Linear Reconfiguration of Cube-Style Modular Robots

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

[1] Université Libre de Bruxelles, Belgique, greg@scs.carleton.ca
[2] Université Libre de Bruxelles, Belgique, sebastien.collette@ulb.ac.be
[3] Villanova University, Villanova, USA, mirela.damian@villanova.edu
[4] Massachusetts Institute of Technology, Cambridge, USA, edemaine@mit.edu
[5] Siena College, Loudonville, N.Y., USA, flatland@siena.edu
[6] Université Libre de Bruxelles, Belgique, stefan.langerman@ulb.ac.be
[7] Smith College, Northampton, USA, orourke@cs.smith.edu
[8] Rutgers University, Camden, USA, rsuneeta@camden.rutgers.edu
[9] Universitat Politècnica de Catalunya, Barcelona, Spain, vera.sacristan@upc.edu
[10] Carleton University, Ottawa, Canada, 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 [1–3], 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.

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

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

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 [4] and telecube atoms [5]. 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 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[11], 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 [1–3], 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. Connectedness of the reconfiguration space is guaranteed for robots composed of *meta-modules* [1, 2], where a meta-module is a connected set of k^3

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

atoms arranged in a $k \times k \times k$ grid. It is desirable that meta-modules be composed of as few atoms as possible. In our reconfiguration algorithms, meta-modules are of minimum size consisting of a $2 \times 2 \times 2$ grid of atoms [6, 2].

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 Fig. 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

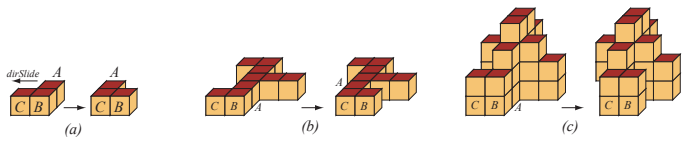

Fig. 1. Examples of SLIDE(x^-): (a) Meta-module A slides alone, (b,c) A carries adjacent meta-modules.

sliding meta-module to "carry" other attached meta-modules (as in Figs. 1b-c), 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.

k-TUNNEL(***sPos***, ***ePos***)**.** Pushes the meta-module located at *sPos* into the robot, and pops a meta-module out of the robot in 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 Fig. 2). This move performs an "inchworm" move between successive turns. Thus the contracted "mass" of *sPos* is transferred between turns using $O(1)$ motions.

In [7], sequences of atom operations implementing SLIDE and k-TUNNEL for cube-style robots are illustrated. 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

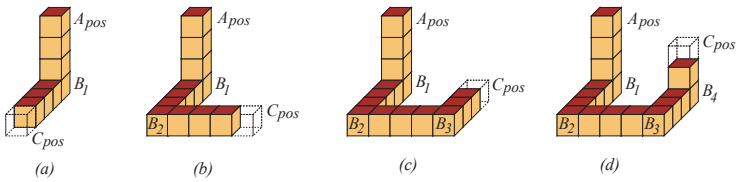

Fig. 2. Examples of TUNNEL(A_{pos}, C_{pos}) with orthogonal turns at B_i, $i = 1, 2, 3, 4$. (a) 1-TUNNEL (b) 2-TUNNEL (c) 3-TUNNEL (d) 4-TUNNEL.

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

We consider both centralized and distributed models of computation. In the centralized model algorithms (described in Sect. 3), computation is performed only by a central processing unit in order to determine the sequence of reconfiguration moves for each meta-module. In Sect. 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 $O(n^2)$ 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 (Sect. 3.1). We then generalize this to 3D robots (Sect. 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*.

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 Fig. 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 Fig. 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^-).

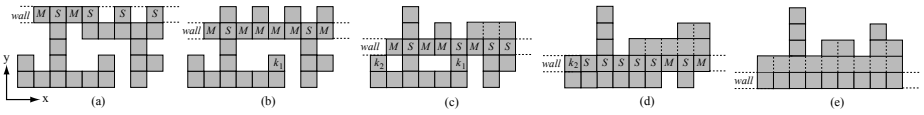

Fig. 3. The initial configuration is converted into a comb as it is swept by the wall.

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 Fig. 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 Fig. 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 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 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). Then each stationary meta-module (now in row w^+) detaches itself from any

adjacent meta-modules to its left and right. Finally, all meta-modules in w^- that are now adjacent to a wall meta-module attach to this wall meta-module.

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

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

Proof. Omitted.

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 $S_1, S_2, \ldots, S_h$ 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'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 ith 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, Figs. 4a and 4b show C_S and C_T. In Fig. 4d, C_U consists of all the shaded and unshaded meta-modules; the common comb is all the shaded boxes.

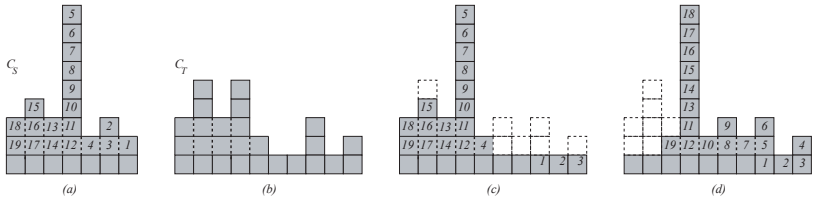

Fig. 4. (a) C_S, with meta-modules labeled in reverse lexicographical order. (b) C_T (c) Shaded meta-modules are C_S after extending its handle's length to match C_U. C_U consists of all shaded and unshaded boxes. Labels indicate which meta-modules moved to form the handle. (d) Shaded meta-modules form 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 Fig. 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 $S_1, \ldots S_i$ 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 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 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. 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 $O(n)$ 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. 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 Fig. 5. The

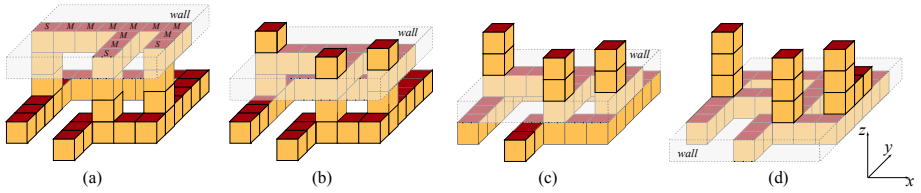

Fig. 5. The 3D-Combing algorithm. (a) Meta-modules labeled M form one F-shaped connected component. (b, c, d) Robot configuration after (1, 2, 3) algorithm iterations. (d) Final terrain configuration.

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.*

3D Terrain to 3D Comb. A 3D Terrain I is reconfigured into a 3D comb by applying the 2D-Combing algorithm of Sect. 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 2D-Combing($B(I)$) using the Slide operation that carries towers (see Fig. 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^i_j be the jth tooth of C^i_S, and let $|S^i_j|$ be the number of teeth meta-modules in tooth S^i_j (not counting the handle module at its base). Let h^i_S be the length of C^i_S'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^i_U, $i = u \ldots 1$, append to a list I the handle meta-modules $(i, 2, 1)$ to $(i, h^i_U, 1)$ of C^i_U, followed by the teeth meta-modules of C^i_U 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^i_S to the 2D common comb determined by $C^i_U = C^i_S \cup C^i_T$ using Algorithm 2. Since C^i_S and C^i_T may not contain the same number of meta-modules, there may not be enough meta-modules in C^i_S to fill the entire handle of C^i_U, in which case C^i_S will become only a portion of the handle that starts with module $(i, 1, 1)$. See Fig. 6a.

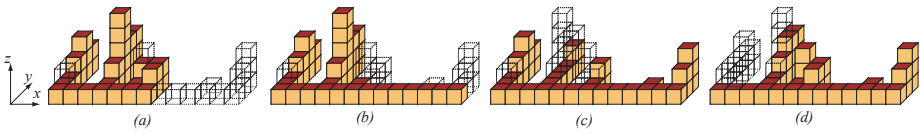

Fig. 6. (a) Solid meta-modules are C_S after each z-comb is converted to a common comb. C_U consists of the solid and the wireframe boxes. (b) C_S after extending its xy-comb handle to match that of C_U. (c) C_S during the execution of Step 4.3 of Algorithm 3, as it lengthens the teeth of C^7_S by tunneling meta-modules from C^4_S. (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. Figure 6b shows the results of this step. The order of the meta-modules in O ensures that each leg of the 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. 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 Fig. 6c, with Fig. 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 $O(n)$ atom operations.*

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

1. For $i = 1 \ldots s$
   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 $k \in \{3, 4\}$

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 $O(n)$ atom operations.*

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. 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 is easily adaptable to the synchronous distributed model. During an initialization phase, each meta-module is sent its starting (x, y, z) location and the wall's starting position. 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 state by just checking if it is attached to a module below. The reverse of this algorithm can be made distributed in a similar way, sweeping the wall up instead of down.

The Comb-To-Common-Comb algorithms can also be distributed, albeit with some stronger requirements. First, the initial and final configurations S and T are 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 an 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. The reverse of this algorithm is similarly distributed.

Acknowledgments. We thank Thomas Hackl for his suggestion on how to reduce the size of the meta-modules [6]. 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. Rus, D., Vona, M.: Crystalline robots: Self-reconfiguration with compressible unit modules. Autonomous Robots **10**(1) (2001) 107–124
2. Vassilvitskii, S., Y., M., Suh, J.: A complete, local and parallel reconfiguration algorithm for cube style modular robots. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation. (2002) 117–122
3. Butler, Z., Rus, D.: Distributed planning and control for modular robots with unit-compressible modules. The Intl. Journal of Robotics Research **22**(9) (2003) 699–715
4. Butler, Z., Fitch, R., Rus, D.: Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting. IEEE/ASME Trans. on Mechatronics **7**(4) (2002) 418–430
5. Suh, J.W., Homans, S.B., Yim, M.: Telecubes: Mechanical design of a module for self-reconfigurable robotics. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation. (2002) 4095–4101
6. Hackl, T.: Personal communication (2007)
7. Aloupis, G., Collette, S., Damian, M., Demaine, E.D., Flatland, R., Langerman, S., O'Rourke, J., Ramaswami, S., Sacristán, V., Wuhrer, S.: Linear reconfiguration of cube-style modular robots (2007) www.cs.siena.edu/papers/cbots_long.pdf.
8. Arkin, E., Bender, M., Mitchell, J., Polishchuk, V.: The snowblower problem. In: Proc. 7th Intl. Workshop on the Algorithmic Foundations of Robotics. (2006)