Complete Heterogeneous Self-Reconfiguration: Deadlock Avoidance Using Hole-Free Assemblies

Daniel Pickem [*] **Magnus Egerstedt** [*] **Jeff S. Shamma** [*]

[*] *Georgia Institue of Technology, GA 30332, USA,*
{daniel.pickem, magnus, shamma}@gatech.edu

Abstract: In this paper, we present a novel approach for heterogeneous self-reconfiguration of a modular robot comprised of heterogeneous cubic modules. We allow an arbitrary number of modules and module classes and show that the proposed self-reconfiguration algorithm can guarantee completion of heterogeneous self-reconfigurations by avoiding so-called hole obstructions. We introduce a hole-detection algorithm to avoid creating holes in connected sets of modules (furthermore called configuration) and an assignment resolution algorithm that prevents deadlocks. Using these algorithms, we show that this approach yields provably successful reconfiguration sequences from any heterogeneous initial configuration to any heterogeneous target configuration as long as the initial and the target configuration are hole and enclosure-free.

1. INTRODUCTION

Self-reconfigurable robots are comprised of individual modules and have the ability to change their morphology, structure, and functionality through changing the relative position of the modules. These modules can disconnect and reconnect to other modules and assemble into a larger robot. Additionally, their modular structure allows self-reconfigurable robots to adapt to a given task (see Yim et al. (2007) or Gilpin and Rus (2010)).

Most of the existing literature on self-reconfiguration focuses on homogeneous self-reconfigurable robots, where all modules are identical and interchangeable (see for example Rus and Vona (2001) or Vassilvitskii et al. (2002)). Whereas homogeneity generally reduces the complexity of reconfiguration algorithms (see Balch and Parker (2002)), homogeneous reconfiguration can not guarantee absolute module position meaning that a module can be moved to any location in the target configuration (see Fitch et al. (2007)). Since modules are interchangeable, this restriction does not matter in a homogeneous system.

Heterogeneous robots on the other hand are comprised of modules with different capabilities and potentially shape and size. In this case, not all modules are interchangeable anymore and it is conceivable that absolute placement of modules is important. Therefore, a heterogeneous reconfiguration algorithm is required to be able to move any module x to a specific desired target position y (where y is any position that is suitable for module x). For example, a position in the target configuration could be reserved for a battery module. Therefore, a heterogeneous reconfiguration algorithm has to guarantee that a battery module is placed at that position.

Another restriction of homogeneous modular robots is their limited extendability of functionality (see Balch and Parker (2002) or Fitch et al. (2007)). One can of course add more modules to a homogeneous robot, but extending module capabilities is difficult because either all modules have to be extended by the same capability (e.g. an additional sensor) or the basic assumption of homogeneity is violated. As soon as a single module differs from all others, we have to treat the robot as a heterogeneous system.

For the purpose of this paper, the relevant characteristics of a heterogeneous system are summarized as follows:

- Heterogeneous robots are comprised of (groups of) modules with distinct properties.
- Absolute positioning of modules needs to be guaranteed for heterogeneous robots.
- Extending the capabilities of a heterogeneous robot is cheaper and adds less hardware complexity since only modules with the desired capabilities have to be added instead of extending the functionality of every module.

Note that the loss of interchangeability in heterogeneous systems introduces additional complications such as the potential for creating deadlocks (see Butler et al. (2002)). In the presented algorithm, the avoidance of deadlocks translates into the avoidance of holes (positions in the target configuration that cannot be reached) and enclosures (positions in the current configuration that cannot be reached) during the reconfiguration process. Additionally, we have to ensure the existence of valid assignments of modules to their respective target positions.

This paper presents a novel approach for automatically reconfiguring heterogeneous modular robots from any given initially hole and enclosure-free configuration into any desired hole and enclosure-free target configuration. The main contribution of this paper is a provably complete algorithm for heterogeneous configuration that avoids deadlocks.

[⋆] This research was sponsored by AFOSR/MURI Project #FA9550-09-1-0538

2. RELATED WORK

While homogeneous self-reconfiguration has been studied in depth in the literature, heterogeneous self-reconfiguration has not enjoyed the same attention. This section gives an overview of relevant work in the area of heterogeneous and homogeneous systems.
In Pickem and Egerstedt (2012), we presented an approach for homogeneous self-reconfiguration that was based on the interchangeability of modules. In this paper, we extend this system to heterogeneous modules.
Fitch et al. (2003) took a different approach to heterogeneous reconfiguration and introduced the melt-sort-grow algorithm that decomposes heterogeneous reconfiguration into three steps. Their approach requires the assembly of an intermediate configuration, which the algorithm in this paper avoids. In a later paper, Fitch et al. (2007) focus on space-constrained heterogeneous reconfiguration. They treat heterogeneous reconfiguration as a two step process, where the first step accomplishes shape-forming (as a homogeneous reconfiguration step) and the second step performs a sorting of modules (ensuring heterogeneous module placement). Their approach decomposes the heterogeneous reconfiguration problem into a two-step process, whereas the algorithm in this paper combines both steps.
Gilpin and Rus (2010) show an algorithm for creating tightly packed configurations for self-disassembly. Their algorithm is designed to create solid blocks of modules with no internal holes, but is presumably not suited for creating arbitrary configurations as shown in this work.

3. SYSTEM REPRESENTATION

In this paper, we employ a model commonly referred to as sliding cube model (see Pickem and Egerstedt (2012) or Rus and Vona (2001)) that employs cubic modules embedded in a discrete three-dimensional unit lattice (see Figure 1). We impose no physical constraints such as gravity, module masses, or forces on the system and assume the reconfiguration process happens in free space.
The instantiation of the sliding cube model in this paper uses modules (furthermore also called cubes) of unit dimensions that move through a cubic lattice in discrete steps of size one. A motion applied to a cube moves the cube from its current position $x \in \mathbb{Z}^3$ to $x + m$ with $x, m \in \mathbb{Z}^3$. Each cube is able to perform two primitive motions - sliding and corner motions. A sliding motion is defined as a motion m_s where $|m_s| = 1$. Applying a sliding motion m_s changes a single coordinate of the cube position by one unit and results in the cube sliding along a surface made of other cubes. A corner motion is defined as m_c where $|m_c| = 2$ with $m_{c,x}, m_{c,y}, m_{c,z} < 2$. Applying a corner motion m_c changes two coordinates by one unit at the same time and transitions a cube to an orthogonal surface. We impose a feasibility constraint on a motion meaning that a motion requires a connected substrate of other cubes (also called a configuration) and has to obey connectivity constraints as well as collision constraints. Furthermore, a configuration $\mathscr{C}$ is composed of N cubes and is a subset of the representable space $\mathbb{Z}^{3N}$.
A heterogeneous configuration is represented as a set of cubes in which each cube is assigned an origin $x \in \mathbb{R}^3$, an ID, and various properties. Without loss of generality, we represent the heterogeneity of cubes with different colors. Path planning as shown in Section 4 relies on this representation of a configuration as a set of cubes. For analytic purposes and foremost for hole detection we use another representation of the configuration - the connectivity graph.

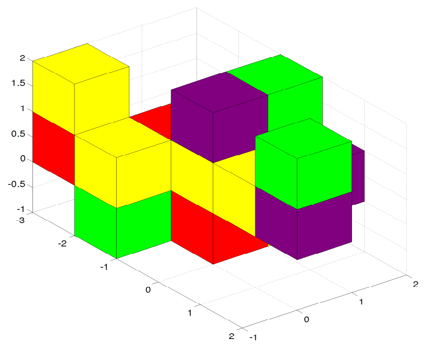


Fig. 1. Example of a heterogeneous configuration comprised of colored unit cubes.

Definition 1. The connectivity graph $G_C = (V, E)$ of a configuration $\mathscr{C}$ is an undirected graph represented by a node set V and and edge set E. V is a finite set of nodes corresponding to the cubes in the configuration, i.e., $V = \{1 \dots N\}$ where N is the total number of cubes. $E \subseteq V \times V$ is the edge set of the graph and contains edges for any two cubes $c_i, c_j, \forall i \neq j$ for which $dist(c_i, c_j) = 1$ as measured by the Manhattan distance (i.e., cubes are connected if they share a common face).

Generally, a connectivity graph can be computed for any set of cubes. It is noteworthy that not just the configuration itself can be represented as a set of cubes but also, for example, the hull of a configuration (see Section 4), which is a set of empty lattice positions. Therefore, one can represent sets of empty lattice positions as a connectivity graph (which will become important in Section 4 and Section 5).

3.1 Assumptions

Before we describe the reconfiguration algorithm in Section 4, we clarify the assumptions made for this work.

- The initial configuration $\mathscr{C}_I$ and the target configuration $\mathscr{C}_T$ are known.
- $\mathscr{C}_I$ and $\mathscr{C}_T$ contain the same number of modules of each property.
- Without loss of generality we assume that there exists an overlap between $\mathscr{C}_I$ and $\mathscr{C}_T$ of exactly one cube. This cube matches the properties of its position (i.e., it is already at its target position).
- The configuration is initially enclosure-free and remains enclosure-free throughout the reconfiguration. As shown in Pickem and Egerstedt (2012), this assumption is required to ensure that the planning space $\mathscr{N}(\mathscr{C})$ (see Section 4) remains connected.
- The overlap of initial and target configuration is hole-free. This assumption is required to show completeness of the reconfiguration algorithm in Section 5.

Holes and enclosures are unreachable positions in the target and current configuration, respectively, and are formally defined in Section 5. They will become important in showing completeness of the reconfiguration algorithm. It is also noteworthy that for every enclosure-free reconfiguration sequence there also exists a sequence that allows enclosures. In this sense, the requirement of enclosure-freedom is not a limitation to the presented reconfiguration approach.

3.2 Constraints

We impose a series of constraints on our system.

- *Collision*: Motions of cubes are only allowed if they do not cause collisions between cubes.
- *Connectivity*: The configuration of cubes needs to remain connected at all times meaning that the connectivity graph can only contain one connected component.
- *Mobility*: Cubes are only allowed to perform sliding and corner motions.
- *Permanence*: Once a cube reaches a position in the target configuration, it remains fixed to that position until the end of the reconfiguration sequence.

As we shall see in Section 5, permanence is used to show completeness of the heterogeneous reconfiguration algorithm. Before we define permanence, we introduce the notion of time for the presented reconfiguration algorithm. The current configuration at the initial time t_0 is $\mathscr{C}(t_0) = \mathscr{C}_I$. A time step $\triangle t$ is the time required to move a cube $c_i \in \mathscr{C}$ from its current position to a target position $r_i \in \mathscr{C}_T$. The current time t is therefore defined as $t = t_0 + n\triangle t$, where n is the number of cubes moved to their respective target position. In this sense, the final time t_f is the time at which every cube $c_i \in \mathscr{C}$ has been moved to its respective $r_i \in \mathscr{C}_T$, i.e., $t_f = t_0 + (N-1)\triangle t$, where N is the number of cubes in the configuration $\mathscr{C}$. Note that only $N-1$ cubes have to be moved because of the initial overlap of size one. In other words $t_f = \{t | c_i(t) = r_i, \forall c_i \in \mathscr{C}, r_i \in \mathscr{C}_T\}$.

Definition 2. Permanence requires that once a module reaches its target position $r_i \in \mathscr{C}_T$ it remains fixed to that position until the target configuration has been fully assembled, i.e., until $\mathscr{C} = \mathscr{C}_T$. More formally, we define permanence as

$$c_i(t) = r_i \in \mathscr{C}_T, \forall t \in [t, t_f]$$

where t is the current time step and t_f is the final time.

As we will show in Section 5, permanence implies that once a hole is created it cannot be undone.

4. RECONFIGURATION PLANNING

Accomplishing a heterogeneous reconfiguration requires to move all cubes in the initial configuration $\mathscr{C}_I$ to matching positions in the target configuration $\mathscr{C}_T$. A matching position $r_i \in \mathscr{C}_T$ must have the same properties as the cube c_i that will occupy it. Note that the permanence constraint implies that we only need to move cubes $c_i \in \mathscr{C}_I$ that are not already at their matching target positions. Because we assume an initial overlap of one cube, $N-1$ cubes have to be moved to their respective target positions. The connectivity constraint prohibits moving cubes whose movement would disconnect the configuration. These cubes can be identified in the connectivity graph of the configuration as articulation points or cut vertices. Note that the connectivity constraint implies that the configuration can only contain one connected component. To guarantee connectivity, we exclude articulation points from the movable set, which is the set of all cubes that are movable at any given time in the reconfiguration process.

Definition 3. The movable set $\mathscr{M}$ is the set of all cubes that can be moved without violating the collision, connectivity, and permanence constraints during the current time step:

$$\mathscr{M} = \{c_i | c_i \in \mathscr{C} \setminus (A(\mathscr{C}) \cap I \cap \mathscr{C}_T)\}$$

Here, $A(\mathscr{C})$ is the set of all articulation points of the current configuration and I is the set of immobile cubes (cubes with six neighbors).

Note that $\mathscr{M} \subset \mathscr{C}$ and that we consider all $c_i \in \mathscr{C}_T$ as immobile because of permanence (an example of a movable set is shown in Fig. 2(b)). Moving a cube from $\mathscr{C}_I$ to $\mathscr{C}_T$ requires an assignment. But before the algorithm can compute a valid assignment (see Algorithm 1, line 5), we have to define which positions of $\mathscr{C}_T$ can be reached within one time step of the current time and configuration, i.e., which positions $r_i \in \mathscr{C}_T$ are adjacent to $\mathscr{C}$. Therefore we introduce the hull $\mathscr{N}(\mathscr{C})$ and the target successor set $\mathscr{R}$.

Definition 4. The hull $\mathscr{N}(\mathscr{C})$ is defined as all positions adjacent to $\mathscr{C}$:

$$\mathscr{N}(\mathscr{C}) = \{c_j | dist(c_i, c_j) = 1, c_i \in \mathscr{C}, c_j \in \mathbb{Z}^3 \setminus \mathscr{C}\}$$

Here, c_i is a cube in the current configuration and c_j is an empty lattice position.

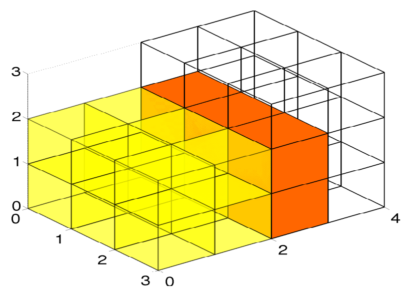
Note that a hull $\mathscr{N}(S)$ can be computed for any set of cubes S. The hull $\mathscr{N}(\mathscr{C})$ in particular represents all positions adjacent to the current configuration and is used as the planning space for the path planner. It also plays an important role in defining which positions can be reached from the current configuration $\mathscr{C}$. The target successor set $\mathscr{R}$ incorporates $\mathscr{N}(\mathscr{C})$ as follows.

Definition 5. The target successor set $\mathscr{R}$ is the set of all positions adjacent to the current configuration that are also in the target configuration, or more formally:

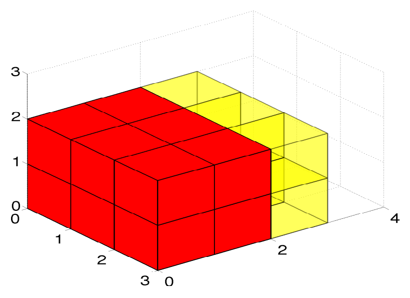
$$\mathscr{R} = (\mathscr{C}_T \cap \mathscr{N}(\mathscr{C}))$$

Note that $\mathscr{R}$ is the set of all target positions that can be reached within one time step (an example is shown in Fig. 2(c)). Also note that positions in the target configuration $r_i \in \mathscr{R}$ have properties just like actual cubes $c_i \in \mathscr{C}$. Empty positions $m_i \in \mathscr{N}(\mathscr{C})$, however, do not have any properties (beyond unit dimensions), i.e., any module can occupy a position in $\mathscr{N}(\mathscr{C})$, which is crucial for path planning. The planning approach presented so far is identical to the one presented in Pickem and Egerstedt (2012) for homogeneous systems. The main difference, however, lies in the assignment of a cube $c_i \in \mathscr{M}$ to a target position $r_i \in \mathscr{R}$. Whereas for homogeneous systems, any cube in $c_i \in \mathscr{M}$ can be moved to any target position $r_i \in \mathscr{R}$, a heterogeneous system requires matching properties (i.e., $p_k(c_i) = p_k(r_j)\ \forall p_k \in P$ where P is the set of all properties defined for cubes).

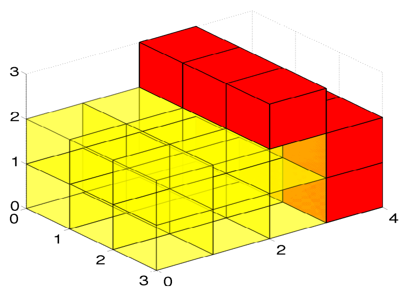
Before the notion of valid assignment is introduced, we define two sets. The set of positions in $\mathscr{R}$ that would create a hole if occupied at the current time step is denoted as H_t.



(a) Initial configuration (transparent), target configuration (wireframe), and overlapping nodes (opaque).



(b) Movable nodes (opaque) as part of the initial configuration (transparent).



(c) Immediate target successor positions (opaque) as neighboring positions of the initial configuration (transparent).

Fig. 2. Representation of the initial and target configuration, movable, and immediate target successor set. For clarity, we show a homogeneous case here. The concepts are the same for the heterogeneous case.

The set of positions in $\mathcal{N}(\mathscr{C})$ that would create enclosures if occupied at the current time step is denoted as E_t (holes and enclosures are formally defined in Section 5). Based on these sets, a valid assignment is defined as follows.

Definition 6. A valid *assignment* a is given by a set $\{c_i, r_j\}$ with $c_i \in \mathscr{M}$ and $r_j \in \mathscr{R}$ where the properties of c_i and r_j have to match. More formally an assignment a is defined as:

$$a = \{c_i, r_j\}, \text{ with } c_i \in \mathscr{M}, r_j \in \mathscr{R} \setminus (H_t \cup E_t)$$
$$\text{and } p_k(c_i) = p_k(r_j), \forall p_k \in P$$

As shown in Pickem and Egerstedt (2012), by construction $\mathscr{M}$ and $\mathscr{R}$ are nonempty unless the target configuration has been assembled, i.e., $\mathscr{C} = \mathscr{C}_T$. For homogeneous reconfiguration, there exists a module that can be moved from its initial to its target position at any time. For heterogeneous self-reconfiguration, this is not generally true even for nonempty sets $\mathscr{M}$ and $\mathscr{R}$. If no cube $c_i \in \mathscr{M}$ matches the properties of an $r_i \in \mathscr{R}$ then no valid assignment can be found. An example is shown in Fig. 3 where $\mathscr{M}$ contains only red cubes and $\mathscr{R}$ only yellow ones. To avoid deadlocks created by a lack of valid assignments, an algorithm called assignment resolution is used (see Algorithm 1, line 10).

Definition 7. Assignment resolution computes a valid assignment as follows.

$$a = \{m_i, t_i\}$$
$$\text{with } t_i = \text{rand}(\mathcal{N}(\mathscr{C}) \setminus (\mathscr{R} \cup \mathcal{N}(\mathscr{R}) \cup E_t))$$

Instead of moving a cube $m_i \in \mathscr{M}$ to a target position, it is moved to a random temporary position t_i in the hull of the current configuration which is neither a target position nor in the neighborhood of a target position. Additionally, m_i can not be picked such that its occupation will create an enclosure.

The basic idea of assignment resolution is that as long as cubes $m_i \in \mathscr{M}$ are moved randomly to positions given by the rule in Def. 7, assignment resolution will eventually make a cube movable whose properties match those of a position $r_i \in \mathscr{R}$. At that point, a valid assignment can be computed and the assembly of the target configuration can continue. Note that assignment resolution will never obstruct the assembly of $\mathscr{C}_T$ because it will never move a cube to a target position nor into the neighborhood of a target position. Therefore, it will never create a hole.

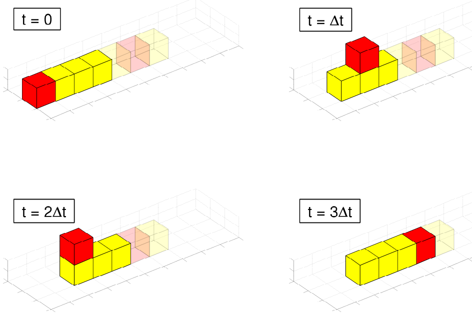


Fig. 3. Example of a reconfiguration sequence using assignment resolution. Opaque cubes represent the current configuration, transparent cubes the unoccupied target positions. The goal is to move the line configuration three steps to the right. Shown is only one assignment resolution step, in which the red cube has to be moved out of the way before the reconfiguration can continue.

Another consequence of assignment resolution is that a target position will never be occupied by a non-matching cube.

Theorem 1. Assignment resolution will enable the computation of a valid assignment with probability 1.

Proof. This result is shown in two steps. First we show that a deterministic sequence of moves can reconfigure all cubes that are not at their target position into an intermediate configuration in which all cubes are movable and a valid assignment can be computed. Then we show that the random cube motions of assignment resolution will assemble such a configuration with probability 1.

(1) Starting in any nonterminal configuration, an intermediate configuration in which a valid assignment can be computed is reachable. We pick the intermediate configuration to be a double line, which consists of two connected linear one-dimensional chains of cubes (such as the one shown in Fig. 3). Any configuration can be reconfigured into a double line (see Pickem and Egerstedt (2012)). and in such a configuration, every cube is movable without disconnecting the configuration. Therefore, the movable set contains every cube

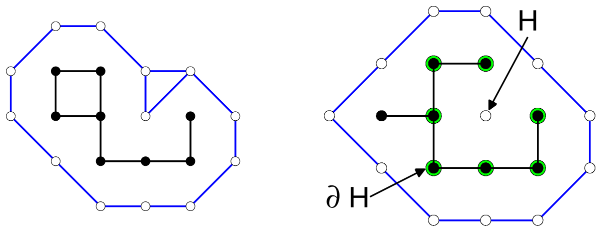
Algorithm 1 Heterogeneous Reconfiguration

```
Require: input $\mathscr{C}_I$, $\mathscr{C}_T$
Ensure: $|\mathscr{C}_I| = |\mathscr{C}_T|$
1: Set $\mathscr{C} = \mathscr{C}_I$
2: while $\mathscr{C} \neq \mathscr{C}_T$ do
3:     Compute $\mathscr{M}$
4:     Compute $\mathscr{R}$
5:     Compute assignment $a = \{c_i, r_i\}$
6:     Compute planning space $\mathscr{N}(\mathscr{C})$
7:     while !isValid(a) do
8:         Remove $r_i$ from $\mathscr{R}$
9:         if isEmpty($\mathscr{R}$) then
10:            $a$ = assignResolution($\mathscr{M}$, $\mathscr{N}(\mathscr{C}), \mathscr{R}, E_t$)
11:            break
12:        else
13:            Recompute $a$
14:        end if
15:    end while
16:    $p$ = planPath($a$, $\mathscr{N}(\mathscr{C})$)
17:    executePath($p$)
18: end while
```

$c_i \in \mathscr{C} \setminus \mathscr{C}_T$, i.e., every cube that is not at a target position. Because $\mathscr{R}$ is nonempty unless $\mathscr{C} = \mathscr{C}_T$ (see Pickem and Egerstedt (2012)), at least one cube $m_i \in \mathscr{M}$ matches a position $r_j \in \mathscr{R}$ and a valid assignment can be found.

(2) Using only random cubes motions, the probability of reaching a double line configuration is approaches 1 as $t \to \infty$. That is because N cubes can only be arranged in a finite number of configurations assuming that at least one cube is fixed (see assumptions made in Section 3). Therefore, we can interpret a reconfiguration process as a finite state machine, in which each state corresponds to a configuration and each transition to a motion of a cube. In such a state machine the probability of reaching a double line configuration from any state (i.e. configuration) is non-zero. Therefore, the random cube motions of assignment resolution will reconfigure the current configuration into a double line as $t \to \infty$, in which a valid assignment can be computed.

Note that in most practical cases a double line will not actually be assembled but a assignment resolution will enable the computation of a valid assignment before. The reconfiguration process can then proceeds towards assembling the target configuration. Referring to Fig. 3, the only reachable position in the first frame is a yellow position. Yet the only movable cube is red. Therefore, assignment resolution has to be applied to the red cube to move it to a random position. Once a yellow cube becomes movable the reconfiguration can proceed to assemble $\mathscr{C}_T$.
After a valid assignment has been computed, the algorithm plans a path from m_i to r_i through planning space $\mathscr{N}(\mathscr{C})$. A path is the concatenation of primitive sliding and corner motions that move cube m_i from its current position to its target position r_i (or to an intermediate position in case of assignment resolution). The path planner obeys collision, connectivity, and mobility constraints. The complete algorithm is outlined as Algorithm 1.



(a) Connectivity graph of configuration $\mathscr{C}$ (filled nodes) and hull $\mathscr{N}(\mathscr{C})$ (hollow nodes) before a hole is created.

(b) Connectivity graph of configuration $\mathscr{C}$ (filled nodes) and hull $\mathscr{N}(\mathscr{C})$ (hollow nodes) after a hole is created. ∂H is indicated by green nodes.

Fig. 4. Hole detection using connectivity graphs of the configuration $\mathscr{C}$ and the planning space $\mathscr{N}(\mathscr{C})$.

5. HOLE DETECTION

The heterogeneous self-reconfiguration algorithm shown in Section 4 potentially creates holes in the target configuration. A hole is a position $r_i \in \mathscr{C}_T$ that can not be reached by any cube $c_i \in (\mathscr{C} \setminus \mathscr{C}_T)$ at the current time t or any future time because of permanence. As we will show, holes obstruct the completion of a reconfiguration because they create permanent deadlocks and must therefore be avoided at all cost. The algorithm presented in this section provably detects holes and can invalidate assignments that would create either. Before we present the algorithm, we first define what a hole and its boundary are.

Definition 8. A *hole* H is an unoccupied target position (or a set of adjacent unoccupied target positions) $c_i \in \mathscr{C}_T$ that is surrounded on all adjacent sides by occupied target positions such that no path exists between any cube $c_i \in \mathscr{C} \setminus \mathscr{C}_T$ and any position in the hole. More formally, a hole H is defined as

$$H \in \mathscr{C}_T \setminus \mathscr{C} \text{ s.t. } \forall h \in H \text{ and } c_i \in \mathscr{C} \setminus \mathscr{C}_T \ \nexists! \ \text{path}(c_i, h_i)$$

The creation of a hole in a two-dimensional case is shown in Fig. 4, which illustrates that the existence of a hole implies that the planning space $\mathscr{N}(\mathscr{C})$ becomes disconnected. Fig. 4 also shows the boundary of a hole ∂H.

Definition 9. The *boundary of a hole* ∂H is defined as $\partial h \in \partial H \ \forall \partial h \in \mathscr{C}_T$ if $\text{dist}(\partial h, H) = 1$. In other words, the boundary of a hole is defined as all occupied target positions that are adjacent to any position $h_i \in H$.

Since both $H \subset \mathscr{C}_T$ and $\partial H \subset \mathscr{C}_T$, once a hole is created, it can not be resolved because of the permanence constraint. Hole-like structures can also occur in the current configuration because of the assignment resolution algorithm. We call these temporary holes enclosures and they are also detected by the hole detection algorithm (Algorithm 2). However, permanence does not apply to these enclosures and their boundary. Therefore, they do not obstruct the successful completion of a reconfiguration sequence. Nonetheless, assignments that create enclosures are not allowed because they would disconnect the planning space $\mathscr{N}(\mathscr{C})$. A connected $\mathscr{N}(\mathscr{C})$, however, is required for proving the detectability of holes (see Theorem 4). To show the importance of hole avoidance, we show that holes obstruct the completion of the reconfiguration process.

Theorem 2. A hole obstructs the successful completion of a heterogeneous reconfiguration.

Proof. The proof follows directly from the definition of a hole, where $H \in \mathscr{C}_T$, i.e. H contains unoccupied target positions. By definition, $\partial H \in \mathscr{C}_T$, which means that because of the permanence constraint none of the cubes $p_i \in \partial H$ will be moved until the reconfiguration process is completed. However, the reconfiguration process will never terminate because it contains a hole H and its boundary ∂H that blocks the occupation of empty positions $h_i \in H \subset \mathscr{C}_T$. Therefore, a reconfiguration sequence can never be completed when a hole is exists.

Theorem 2 shows that a successful reconfiguration cannot contain any holes. In general, however, we cannot conclude that the absence of holes implies a successful reconfiguration. In a heterogeneous self-reconfigurable system, deadlocks can happen that prevent the completion of a reconfiguration. A deadlock occurs when no module in $\mathscr{M}$ matches the properties of a position in $\mathscr{R}$. Without any further measures, the algorithm would get stuck in such a case. However, with the assignment resolution algorithm introduced in Section 4 we can resolve the issue of deadlocks and guarantee successful heterogeneous reconfiguration.

Theorem 3. Using the assignment resolution algorithm, the absence of holes guarantees a successful reconfiguration.

Proof. As shown in Theorem 1, assignment resolution ensures that a valid assignment can be found as long as the reconfiguration has not been completed (because $\mathscr{M}$ and $\mathscr{R}$ are nonempty as long as $\mathscr{C} \neq \mathscr{C}_T$, as shown in Pickem and Egerstedt (2012)). A valid assignment guarantees that progress is made towards assembling $\mathscr{C}_T$, i.e., the number of cubes occupying positions $r_i \in \mathscr{C}_T$ is strictly monotonically increasing in time. This property follows from Pickem and Egerstedt (2012), where we showed for homogeneous systems that a valid assignment ensures that a path can be computed from $c_i \in \mathscr{M}$ to $r_i \in \mathscr{R}$. For heterogeneous systems, this assertion holds as long as holes and enclosures are avoided in the reconfiguration process. Given valid assignments, the only way a heterogeneous reconfiguration can fail is if unreachable positions occur in the target configuration. An unreachable position in $\mathscr{C}_T$ will only occur if a hole is created because by definition, the existence of a hole implies that there does not exist a path from any $c_i \in \mathscr{C} \setminus \partial H$ to a position $h_i \in H$. Therefore, the absence of holes guarantees that there exists a path to any $r_i \in \mathscr{R}$ at the current time step (if a matching cube is movable) and to any $r_j \in \mathscr{C}_T$ for all future time steps $t \leq t_f$. Given that we can always find a valid assignment using assignment resolution and a valid assignment means progress towards assembling $\mathscr{C}_T$ is made in the absence of holes, we can guarantee that $\mathscr{C}_T$ is indeed assembled.

The above proof relies on the ability to detect and therefore avoid holes. As long as $\mathscr{C}_I$ is a hole and enclosure-free initial configuration, the reconfiguration algorithm (Algorithm 1) will guarantee hole and enclosure-freedom throughout the configuration because Algorithm 2 will detect any hole and enclosure in the reconfiguration sequence and invalidate assignments that create either. The hole detection algorithm computes the number of connected components of the connectivity graph of the hull $\mathscr{N}$, which is initially connected (because we know that $\mathscr{C}$ is hole and enclosure-free). After computing the hull $\mathscr{N}$, its connectivity graph G_C, and the associated graph Laplacian L, the algorithm determines the eigenvalues of L. The number of zero eigenvalues indicates the number of connected components (according to Mesbahi and Egerstedt (2010)). If G_C is disconnected to begin with, the algorithm will report the detection of a hole or an enclosure for any assignment and will essentially stop the reconfiguration. Therefore, it is crucial that hole and enclosure-freedom is maintained throughout the reconfiguration.

Algorithm 2 Hole Detection

```
Require: input $a = \{c_i, r_i\}$, $\mathscr{C}$
Ensure: $r_i \in \mathscr{G}_{\mathscr{C}}$ and $r_i \in \mathscr{R}$
 1: Compute $\mathscr{N}(\mathscr{C})$
 2: Compute $G_C$ of $\mathscr{N}(\mathscr{C})$
 3: Compute $L$ of $G_C$
 4: if $|\lambda_i = 0| > 1, \forall \lambda \in eig(L)$ then
 5:     Return true
 6: else
 7:     Remove $r_i$ from $\mathscr{N}(\mathscr{C})$
 8:     Update $c_i$'s origin to $r_i$ (in $\mathscr{C}$)
 9:     Recompute $\mathscr{N}(\mathscr{C})$, $G_C$, and $L$
10:     if $|\lambda_i = 0| > 1, \forall \lambda \in eig(L)$ then
11:         Return true
12:     else
13:         Return false
14:     end if
15: end if
```

If G_C is connected, Algorithm 2 will proceed to determine if the given assignment a (i.e., the movement of cube c_i to position r_i) will create a hole or an enclosure. The algorithm executes a hypothetic move of cube c_i to position r_i (lines 7 to 9) and recomputes $\mathscr{N}$, G_C and L as well as the multiplicity of zero eigenvalues of L. If more than one zero eigenvalue is found, the algorithm concludes that the last motion has created a hole or an enclosure. Differentiating a hole and an enclosure can be done using the definition of a hole. If $h_i \in \mathscr{C}_T \; \forall h_i \in (H \cup \partial H)$ then a hole has been detected, otherwise an enclosure is reported. Essentially, the algorithm checks whether the hole and its boundary are in the target configuration or not. The following theorem shows that this algorithm indeed guarantees the detection of holes and enclosures.

Theorem 4. The proposed hole detection algorithm (Algorithm 2) will detect a hole (or an enclosure) if and only if there exists a hole (or enclosure).

Proof. The proof is based on the assumption that the current configuration $\mathscr{C}$ is hole and enclosure-free. Therefore, the connectivity graph G_C of $\mathscr{N}$ is connected.

- Necessity (D $\rightarrow$ H): The detection of a hole is based on the eigenvalues λ_i of the graph Laplacian L, where the multiplicity of $\lambda_i = 0$ (the number of zero eigenvalues) indicates the number of connected components of the graph. Therefore, we can conclude that G_C is disconnected if the multiplicity of $\lambda_i = 0$ is larger than 1. By construction, $\mathscr{N}$ is connected, which means that the occurrence of two or more connected

Table 1. Reconfiguration planning results for reconfigurations from random to box configurations

Size	Steps	Detected Holes	# of Resolutions
10	33	0	3
20	69	0	1
30	107	0	0
40	150	0	0
50	233	0	1

components implies the existence of either a hole or an enclosure.

- Sufficiency (H $\rightarrow$ D): Starting with an initially connected $\mathscr{N}$ and using the fact that a hole or an enclosure increases the number of connected components of G_C, the hole-detection problem is reduced to determining how many connected components G_C contains. If the configuration $\mathscr{C}$ contains a hole or an enclosure, L has more than one zero eigenvalue, and we immediately detect the hole or the enclosure.

Given that the current configuration $\mathscr{C}$ is hole and enclosure-free we can guarantee that $\mathscr{C}$ will be hole and enclosure-free after moving the current cube according to assignment a because the hole detection algorithm only allows valid assignments. Hole and enclosure freedom of the entire reconfiguration sequence follow by induction.

6. RESULTS AND CONCLUSIONS

This section shows numerical results based on reconfiguration sequences from random three-dimensional initial configurations to box configurations (see Table 6) as well as a full reconfiguration sequence from a random configuration to a layered pyramid (Fig. 5). In Table 6, the field *Size* refers to the size of the configurations, *Detected Holes* reports the number of holes and enclosures detected during reconfiguration, and *# of Resolutions* denotes the number of assignment resolutions during each reconfiguration. The results indicate that the number of assignment resolution steps is larger for small configurations and decreases as the size of the configuration increases. This can be attributed to the larger number of movable and reachable cubes in larger configurations, which makes it very likely that a valid assignment can be found.

The complete absence of holes and enclosures is an artifact of this specific set of randomly generated problem instances. Note, however, that the occurrence of holes and enclosures is very unlikely in general and requires multiple consecutive assignment or assignment resolution steps to arrange cubes in specific shapes. Nonetheless, holes and enclosures are theoretically possible and need to be avoided in a complete heterogeneous self-reconfiguration algorithm as we have shown in this paper.

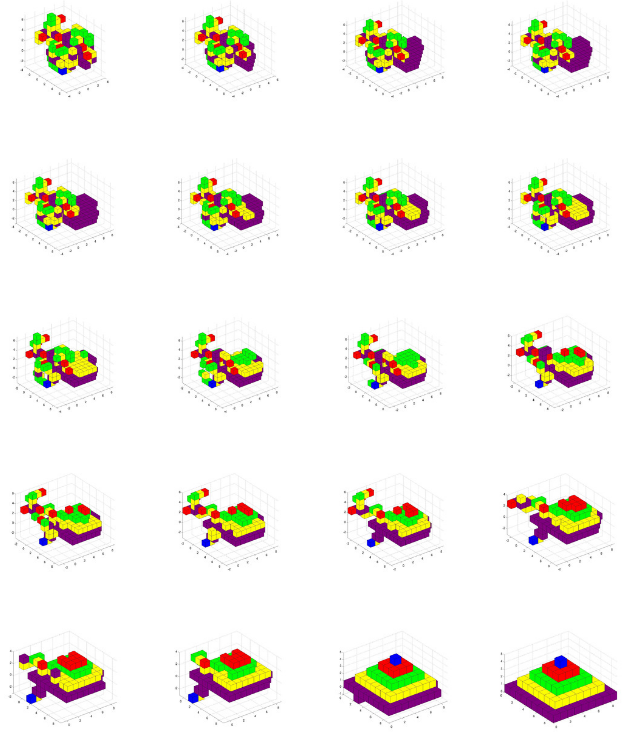


Fig. 5. Example of a heterogeneous reconfiguration from a random three-dimensional configuration to a layered pyramid.

REFERENCES

Balch, T. and Parker, L.E. (eds.) (2002). *Robot Teams: From Diversity to Polymorphism.* A. K. Peters, Ltd., Natick, MA, USA.

Butler, Z., Murata, S., and Rus, D. (2002). Distributed replication algorithms for self-reconfiguring modular robots. In *In Proceedings of Distributed Autonomous Robotic Systems (DARS).*

Fitch, R., Butler, Z., and Rus, D. (2003). Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, 2460 – 2467.

Fitch, R., Butler, Z., and Rus, D. (2007). In-place distributed heterogeneous reconfiguration planning. In R. Alami, R. Chatila, and H. Asama (eds.), *Distributed Autonomous Robotic Systems 6*, 159–168. Springer Japan.

Gilpin, K. and Rus, D. (2010). Modular robot systems. *IEEE Robot. Automat. Mag.*, 17(3), 38–55.

Mesbahi, M. and Egerstedt, M. (2010). *Graph Theoretic Methods in Multiagent Networks.* Princeton University Press.

Pickem, D. and Egerstedt, M. (2012). Self-reconfiguration using graph grammars for modular robotics. IFAC Conference on Analysis and Design of Hybrid Systems, Eindhoven, Netherlands.

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

Vassilvitskii, S., Yim, M., and Suh, J. (2002). A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, 117 – 122 vol.1.

Yim, M., Shen, W.M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., and Chirikjian, G.S. (2007). Modular self-reconfigurable robot systems – challenges and opportunities for the future. *IEEE Robotics and Autonomation Magazine*, March, 43–53.