Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Self-Reconfiguration in Two-Dimensions via Active Subtraction with Modular Robots

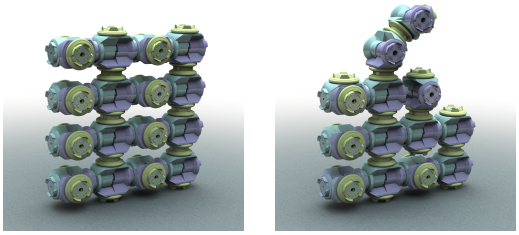
Matthew D. Hall, Anıl Özdemir, and Roderich Groß
Sheffield Robotics, The University of Sheffield, UK
{m.d.hall, a.ozdemir, r.gross}@sheffield.ac.uk

***Abstract*—Modular robotic systems comprise groups of physically connected modules which can be reconfigured to create morphologies that suit an environment or task. One method of reconfiguration is via subtraction, where extraneous modules disconnect from an initial configuration, before being removed by external intervention. In this paper, we consider an approach to reconfiguration in two dimensions, here termed *active* subtraction, in which unwanted modules traverse a configuration in order to remove themselves safely, without the need for external intervention, making it a form of self-reconfiguration. We present a sequential solution that selects suitable extraneous modules that then remove themselves, one by one. We also present a parallel solution that, while being more computationally demanding, allows multiple modules to move simultaneously. Both solutions are proven to (i) be correct for any given non-hollow structure, and (ii) require, in the worst case, quadratic time proportionally to the number of modules. Simulation studies demonstrate that both solutions work effectively for specified and randomly generated desired configurations with hundreds of modules, and reveal a non-monotonic dependence between the performance and the percentage of modules to be removed. This work demonstrates active subtraction as a viable method of self-reconfiguration, without the need for heuristics or stochasticity, and suggests its potential for application in real-world systems.**

I. Introduction

Modular robots are groups of robotic units that combine their abilities and attributes to become a system that is greater than the sum of its parts. Through the collaboration of these lesser parts, the whole can complete tasks that would be challenging for the individual, much like living cells [6] or a colony of ants [1]. Most conventional robotic systems are specialized, with limited use cases. This makes development time-consuming and expensive, as the robots require alterations for each task. It is also more expensive for the end-user, as they would require a multitude of robots for changing demand. Through the development of modular systems, these shortcomings can be negated [20].

For modular systems to become autonomous, it is required that they are able to self-reconfigure, that is, that they can change their morphology without intervention from a user. A number of solutions have been presented for the self-reconfiguration of modular systems [18, 2]. The majority of solutions involve adding modules or rearranging them to form shapes, known as additive self-reconfiguration. Additive self-reconfiguration has been demonstrated with a number of modular systems [17, 16, 19, 4], and is at present the most common form of reconfiguration.



(a) (b)

Fig. 1. Starting from a 4 × 4 solid initial configuration, as in (a), active subtraction is employed to reconfigure modules to a given desired configuration, shown in progress in (b). Illustrated using the *HyMod* platform.

Gilpin et al. [8] developed an alternative approach to reconfiguration, where a desired shape is formed by initiating a group of modules in a given starting formation, before having unneeded modules detach themselves. This is known as subtractive reconfiguration, and has been researched considerably less than additive reconfiguration, despite the promise it holds. Using the *Miche* system, Gilpin et al. demonstrate subtractive reconfiguration in three dimensions. The *Miche* modules are small cubes, equipped with magnetic connectors on their six faces. When initialized in a cubic lattice, the modules are able to determine whether or not they are required in the final configuration and disconnect from their neighbors if they are not. In an effort to minimize the size of the modules, they are incapable of locomotion, so external actuation is required to remove the unnecessary modules, in this case by using gravity. The authors demonstrate the presented concept with 28 *Miche* modules, forming a variety of shapes autonomously, such as a humanoid and a dog shape. Gilpin et al. go on to demonstrate subtractive reconfiguration with the *Robot Pebbles* system [9]. The *Robot Pebbles* are smaller scale robots, inspired by *Miche*, but with connectors on only four of the faces, thereby restricting the lattice to two dimensions. The modules feature no internal locomotion, instead relying on a vibrating table to locomote once disconnected from the structure.

Subtractive reconfiguration is also possible with non-modular robots, as demonstrated by Gauci et al. [7]. In this work, the authors use a swarm of 725 *Kilobots*, which are small-scale mobile robots [15]. The *Kilobots* are initiated in a tightly packed group and given a user-defined goal shape. The robots that are not required in the final configuration remove themselves through the combination of collision avoidance,

phototaxis, and antiphototaxis. The collision avoidance means that the robots will not remove themselves from the structure until there is a clear path ahead, and will not disturb the goal shape in doing so. The work shows the strategy successfully yielding certain types of shapes when the correct combination of phototaxis and antiphototaxis is used.

The motivation behind subtractive reconfiguration comes from the increased reliability it lends. Typically, modules require high precision to form connections between themselves, unless these connections are already present in the starting configuration, as is the case in a subtractive approach (see Figure 1). As discussed by Gauci et al. [7], an initially connected structure can guarantee that the modules are able to communicate from initialization. Because of this, information such as the initial configuration or the control scheme can be easily passed to all members before reconfiguration begins.

The aforementioned work is only concerned with non-modular robots, or non-mobile modules which are removed through external locomotion once disconnected. Employing lattice-based modular systems with internal locomotion, such as *HyMod* [13], *M-TRAN* [12], *SMORES* [5] or *M-Blocks* [14], it could be possible to actively subtract modules from an initial starting cube, leaving behind the desired structure. The problem is particularly challenging in the presence of gravity, as the redundant modules may have to leave in a particular order to prevent the current structure from collapsing. This problem is addressed for the first time in this work.

In this paper, we present a subtractive approach for reconfigurable robotic systems, by which extraneous modules actively remove themselves from a starting configuration, to leave behind a given structure. We refer to this approach as *active* subtraction. We present two solutions: one where modules operate purely sequentially, while the other allows for parallel movement. We prove the correctness and formally characterize the worst-case performance of both solutions. Moreover, we conduct simulations with up to hundreds of modules to evaluate performance over a range of conditions.

The paper is structured as follows. Section II defines the problem to be solved and outlines the capabilities of the modules. Section III presents the two solutions. Their validity and run-time performance are formally analyzed in Section IV. We then simulate a range of systems in Section V. Section VI concludes the paper.

II. Problem Formulation

Consider a group of modules arranged to fill a two-dimensional, rectangular space in a vertical plane. The environment is bounded in one of the four directions by a static surface, which we refer to as the *ground*. In the remaining three directions there exists free space, at least large enough for a module to move into. By default, the modules must remain connected to themselves and to the ground, as otherwise, the structure would collapse due to gravitational forces. The modules, ground, and all connections are considered to be infinitely rigid. A number of modules in this *initial configuration* are required to remove themselves, to leave behind only modules in a predefined configuration, known as the *desired configuration*. As is the case with all existing subtractive reconfiguration, hollow spaces are not permitted in the desired configuration, as modules that are to leave the hollow space behind would not be able to remove themselves [7, 9]. We nominate a *sink* location, which is at ground level and adjacent to the structure. When a module reaches the sink location, we assume it is automatically removed from the configuration.

Fig. 2. Example of an active module employing adjacent movement to travel West, followed by corner movement to travel South-West.

The modules in the desired configuration are henceforth referred to as *included* modules and are stationary throughout the reconfiguration. The excess modules that are to be removed are referred to as *excluded* modules. The task is considered complete when all excluded modules have managed to reach the sink location.

A. Objective

Formally, a robot's configuration is represented as a graph, $G = (V, E)$. Each node $v \in V$ corresponds to a module with coordinates $(v_x, v_y) \in \mathbb{N} \times \mathbb{N}^+$. Edges represent connections between modules; a pair of modules are connected if and only if their Manhattan distance is 1. Let G' denote the *augmented* configuration that includes the original graph, G, as well as an additional node, g, representing the ground, connected to all nodes v where $v_y = 1$. Formally, we have $G' = (V', E')$ with $V' = V \cup \{g\}$ and $E' = E \cup \{\{g, v\} \,|\, v \in V \text{ and } v_y = 1\}$. A configuration G is referred to as *feasible* (i.e., non-collapsing) if the augmented configuration, G', is a connected graph.

We assume that the modules are initially arranged in a rectangular configuration $G_0 = (V_0, E_0)$, of width x_{max} and height y_{max}, where $V_0 = \{(x, y) \,|\, 1 \leq x \leq x_{max}, 1 \leq y \leq y_{max}\}$. Let $G^{\text{inc}} = (V^{\text{inc}}, E^{\text{inc}})$ denote a desired, feasible, non-hollow configuration, where $V^{\text{inc}} \subseteq V_0$ are the included modules. For the robot to self-reconfigure from G_0 to G^{inc}, the excluded modules, $V^{\text{exc}} = V_0 - V^{\text{inc}}$, must be removed. For a module to be removed, it has to reach the sink, which we assume is located at (0,1).

The reconfiguration problem consists of identifying a finite sequence of configurations $G_1, G_2, \ldots, G_T$ such that

- For all $k = 1, 2, \ldots, T$: G_k is feasible;
- For all $k = 1, 2, \ldots, T$: G_k can be reached from G_{k-1} via one module executing a valid movement, which is potentially followed by removing the module if it has reached the sink;
- $G_T = G^{\text{inc}}$.

B. Module Capabilities

The work presented here uses a model of a robot that, while having similar attributes to the aforementioned lattice-based modular systems, implements them in a simplified manner,

akin to the sliding square method of movement [11]. The possible movements can be separated into two types, adjacent and diagonal movement, and are expressed in terms of a module's Moore neighborhood. Adjacent movement is where a module traverses by a distance of one unit along only one axis, that is North (N), East (E), South (S), or West (W). Whereas, diagonal movement is when the module traverses a distance of one unit along both axes, that is, North-West (NW), North-East (NE), South-East (SE), or South-West (SW). Both movement types are illustrated in Figure 2.

Each module is aware of the state of its connectors, that is, whether connected or not. It is assumed that connected modules are able to communicate with one another and can pass messages through the system, including through the ground, as is possible with the *HyMod* surface extension [13]. Through this communication, the modules are able to establish their relative position and heading within a global co-ordinate system. For one of the solutions that is presented, at least one module is capable of computation, and has an internal clock, which is used for implicit synchronization.

III. Solutions

We present two solutions for the problem described in Section II. The first solves the problem while having modules actively remove themselves one at a time, termed Sequential Active Subtraction, or `SAS`. The second builds on the first but grants the modules parallel movement, greatly reducing the overall reconfiguration time, which is Parallel Active Subtraction, or `PAS`.

A. Sequential Active Subtraction (`SAS`)

In this section, we are concerned with creating a pair of algorithms that allow a leader module to select modules to be removed from the structure, and for the selected modules to navigate to the sink location. This will leave behind a set of connected modules that form a given desired configuration. All modules execute an identical distributed algorithm based solely on local knowledge, except for the leader module, the identity of which can be chosen at initialization[1], and which requires knowledge of the initial and desired configurations. As all modules are initially connected, the location of each module can be defined relative to a single point within the system, by passing a message and incrementing the relevant value. The tuple of coordinate values can be used to identify the modules. The leader module is responsible for choosing and informing sequential excluded modules to remove themselves, one at a time. It does so by executing the algorithm presented in Section III-A1. When an excluded module is notified it is to be removed, it becomes the *active* excluded module. The active excluded module removes itself from the configuration, by executing the algorithm presented in Section III-A2.

1) Leader Module Algorithm: Algorithm 1, which is executed by the leader module, requires the module to have knowledge of which modules are excluded (V^{exc}) and which are included (V^{inc}). Assuming that the set of excluded modules is not empty, the leader chooses one module to be removed. The leader can infer from V^{exc} and V^{inc} how many inactive connectors each excluded module has. Let F_v denote the number of inactive connectors for module $v \in V^{\text{exc}}$. For a module v to be considered for removal, we must have $F_v > 0$. The leader module first determines the highest horizontal layer of the structure that contains at least one excluded module with $F_v > 0$. It denotes by V^{top} the set of excluded modules with $F_v > 0$ in this layer (line 2). It then chooses from V^{top} the module furthest West, m, which is also nearest to the sink (line 3). It informs the chosen module to remove itself from the configuration (line 4) and updates the set of excluded modules accordingly (line 5). The leader then waits for the active excluded module to reach the sink, which it can be informed of through the structure, or through the ground (line 6). The process is repeated until V^{exc} is empty. As the process is sequential, modules cannot possibly collide.

2) Active Excluded Module Algorithm: Once a module is informed it is the active excluded module, it performs Algorithm 2. The first step is to determine an order, S_D, of

[1]The most Westerly included module on the ground can be chosen as the leader.

Algorithm 1 Leader module (`SAS`)

Require: V^{exc} and V^{inc}
1: **while** $V^{\text{exc}} \neq \emptyset$ **do**
2: $V^{\text{top}} \leftarrow \{u \in V^{\text{exc}} \mid u_y = \max_{v \in V^{\text{exc}}}\{v_y \mid F_v > 0\}\}$
3: $m \leftarrow \text{argmin}_{v \in V^{\text{top}}} v_x$
4: **notify** m ▷ send signal to activate m
5: $V^{\text{exc}} \leftarrow V^{\text{exc}} \setminus \{m\}$ ▷ update the set of excluded modules
6: **wait** m ▷ wait for m to reach sink
7: **end while**

Algorithm 2 Active excluded module (`SAS` or `PAS`)

1: $S_D \leftarrow$ {S, SW, W, NW, N, NE, E, SE}
2: **while** position $\neq$ sink **do**
3: **for all** $d \in S_D$ **do**
4: **if** $cell(d) = empty$ **then**
5: **if** $d \in$ {N, E, S, W} **then** ▷ space is adjacent
6: $D \leftarrow d$
7: **break** ▷ progress to line 16
8: **else** ▷ space is diagonal
9: **if** $cell(d+1) = empty$ **then**
10: $D \leftarrow d$
11: **break** ▷ progress to line 16
12: **end if**
13: **end if**
14: **end if**
15: **end for**
16: **move** D ▷ move in direction D
17: $S_D \leftarrow \{D-2, D-1, D, D+1, D+2, D+3, D+4, D+5\}$
18: **end while**
19: **notify** leader

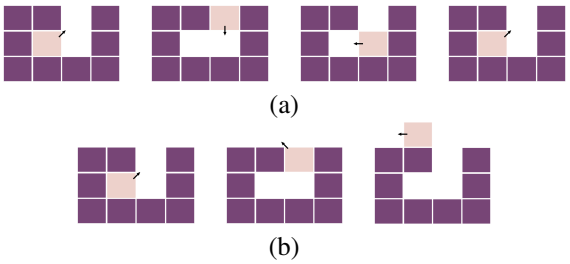


Fig. 3. Example of a stalemate when using static prioritization, (a), and dynamic prioritization avoiding the stalemate, (b).

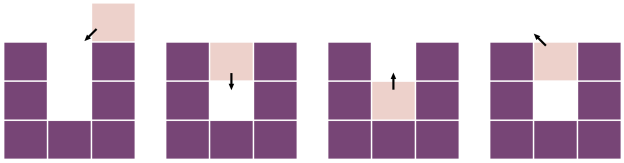


Fig. 4. Example of a module escaping a dead-end by checking all possible directions of movement, including opposite to the direction it already traveled.

directions in which to probe the state of neighboring cells in its Moore neighborhood (given a direction $d \in S_D$, let $cell(d)$ be $empty$ if the corresponding neighbor cell is not occupied). By defining the sink at position (0, 1), the module has a location to aim for, which is always either South, West or South-West. The module therefore first prioritizes a South-bound direction, continuing in a clockwise order around the directions of the Moore neighborhood, as shown on line 1. For example, in Figure 2, the active module first probes whether there is a neighbor to the South, as there is it proceeds to probe to the South-West and then West, where it finds a free space.

For non-rectilinear structures, like the one shown in Figure 3(a), the aforementioned fixed priority sequence may result in a deadlock. The active module would first move to the North-East but is then able to move South, which is the prioritized movement, before moving West as the next highest prioritized movement. It would then repeat these three movements indefinitely, not making progress towards the sink.

To avoid such situations, the movement of the active module is informed by the direction, D, it last traveled. This information is used to alter the priority (i.e., set S_D) by which the next direction is decided, shown on line 17. The numeric alteration consists of steps around the Moore neighborhood in a clockwise direction. The five Moore neighborhood positions that do not constitute a 'backward' movement, that is, $D-2, D-1, D, D+1, D+2$, are given priority over the other three possible directions, $D+3, D+4, D+5$, so as to avoid retreading the same path, where possible. Figure 3(b) shows this dynamic prioritization resolving the previous stale-mate.

In some situations, such as in Figure 4, a module could find itself at a dead-end. Movement directions $D+3, D+4$ or $D+5$ become relevant and are probed in this sequence. This act of dynamically prioritizing directions to move means that the active module can always reach the sink, although not necessarily by taking the shortest possible path.

Each movement that a direction, d, would imply falls into the category of adjacent movement or diagonal movement. For an adjacent movement it is simply necessary that $cell(d)$ be empty, and if this is the case then the direction to move, D, can be set to direction d (line 5). In the case of a diagonal movement, it is required that an adjacent cell be passed through to reach the potential location. The next neighboring cell in the Moore neighborhood is checked (if the previous cell was empty, it would have already been selected), as shown on line 9.

Once D has been set, the module moves to the neighboring cell located in direction D, re-prioritizes the order of directions in S_D and repeats the process. This continues until the module reaches the sink, at which point it can be considered removed, and informs the leader accordingly.

Algorithm 3 Leader module (PAS)

Require: $\varphi(\cdot)$, Δ ▷ Obtained by simulating Algorithms 1, 2

1: $s_1 \leftarrow 0$
2: $a_1 \leftarrow \Delta_1$, $a_{j\neq 1} \leftarrow \infty$
3: **for all** $i = 2$ to $|V^{exc}|$ **do**
4: $\quad s_i \leftarrow \max(0, a_{i-1} + 2 - \Delta_i)$
5: $\quad V^{sim} \leftarrow V^{inc} \cup \{u \in V^{exc} \,|\, a_{\varphi(u)} \geq s_i\}$
6: $\quad$ collision $\leftarrow$ **simulate** V^{sim}
7: $\quad$ **if** collision **then**
8: $\quad\quad s_i \leftarrow s_i + 1$
9: $\quad\quad$ **goto** line 5
10: $\quad$ **end if**
11: $\quad a_i \leftarrow s_i + \Delta_i$
12: **end for**
13: **notify** n at s_n ▷ send signals to modules at start times

B. Parallel Active Subtraction (PAS)

To reduce the time cost of reconfiguration, we present a version of active subtraction that allows modules to move in parallel, PAS. It requires at least the leader module to be capable of conducting simulations. Active modules continue to use Algorithm 2 (see Section III-A2). PAS only differs from SAS in the timings of when the excluded modules are activated. However, it preserves the order in which the modules arrive at the sink.

Algorithm 3, which is executed by the leader module, requires the module to simulate Algorithms 1 and 2 to obtain the order in which the excluded modules reach the sink, $\varphi(\cdot)$, along with the number of time steps that the i^{th} module reaching the sink was active, Δ_i, where $i = 1, 2, \ldots, |V^{exc}|$. For example, if excluded module m was the second module to arrive at the sink and was active for 20 time steps, $\varphi(m) = 2$ and $\Delta_2 = 20$. The start time of the first module, s_1, is set to 0. Its arrival time, a_1, is set to Δ_1, with all other modules assigned an arrival time of infinity. Subsequent modules must arrive at the sink after their predecessor. Moreover, collisions among active modules must be prevented. We define a collision as an active module residing within the Moore neighborhood of another active module. Without collisions, interaction between active modules is avoided, which could otherwise

affect the direction in which they move (see Algorithm 2). As a consequence, the arrival times of subsequent modules must differ by at least two. The earliest possible start time for the module in question is chosen accordingly (line 4). The leader module then simulates the active subtraction process, to determine whether the module in question is successfully removed without any collision (line 6). To lessen the computational load, it is only necessary to simulate the movement of modules that have not reached the sink by the time the newly considered module is deployed (line 5). Excluded modules that are yet to be assessed are included in the configuration, but as they have no designated start time, their movement is not simulated. If a collision occurs, the start time of the module in question is delayed by one time step and the simulation is repeated. Once a valid start time has been found, the arrival time is calculated accordingly (line 11). Once all excluded modules have been assigned a start time, the leader can begin the reconfiguration of the system. It initializes a clock and activates the excluded modules at their corresponding starting times (line 13). Note that multiple modules may be activated on the same time step.

IV. Mathematical Analysis

This section formally analyzes the correctness and run-time performance of both SAS and PAS.

Lemma 1. *If $V^{exc} \neq \emptyset$, lines 2 and 3 of Algorithm 1 identify a module to be removed.*

Proof: Consider the set of excluded modules that have at least one inactive connector, $B = \{v \in V^{exc} \mid F_v > 0\}$. If $B = \emptyset$, then every excluded module is surrounded by other modules. The number of excluded modules is finite, therefore, the excluded modules would have to be fully encapsulated by the included modules. However, this is not possible, as the desired structure (made of the included modules) is non-hollow. Therefore $B \neq \emptyset$. As B is finite, it follows that there exist modules in B that have maximal height, that is, $V^{top} \neq \emptyset$. From these modules, the algorithm chooses the most-Westerly one. The choice is unique, as no two modules have identical coordinates. ■

Lemma 2. *The module chosen by Algorithm 1 can be removed without the configuration becoming infeasible.*

Proof: The included modules form, by assumption, a feasible configuration. Hence, module m is not required to support any included module. Let us assume that removing m would make the configuration infeasible. Consider a vertical block of u excluded modules, that is surrounded by two empty cells at coordinates (x, y) to $(x, y+u+1)$. One can show that u has to be 0:

1) If $(x, y+u+1)$ was removed before (x, y), then $(x, y+u)$ would have been removed before (x, y) as well.
2) If (x, y) was removed before $(x, y+u+1)$, then $(x, y+1)$ would have been removed before $(x, y+u+1)$.

Hence, u must be 0. As a consequence, any excluded module is supported (directly or indirectly) from below by either an included module or the ground, or from above by some included module. ■

Lemma 3. *If Algorithm 1 chooses a module, m, to be removed, Algorithm 2 constructs a valid, finite path from m to the sink.*

Proof: Let G_s be the graph, where the nodes are the free faces of any module in the configuration, excluding the focal module m, and where the edges link any pair of adjacent faces. As m is the module to be removed, it must have at least one free face, implying it is adjacent to one or more faces in G_s. We know that G_s is connected at any time. Each free module face can only be adjacent to one other face per side, that is, each node has two edges, meaning G_s is a path graph of finite length. The sink, s, is located at the ground and adjacent to the configuration. Thus, s is always at one end of the path graph. As m is connected to a module whose free face(s) belongs to G_s and the proposed movement framework allows m to move along a new direction, that is, an edge (see Algorithm 2), m always reaches an end of G_s. If the end that is reached is not s, implying the module was traversing the path in a clockwise direction around the configuration, it will re-traverse G_s in the opposite (counter-clockwise) direction. This causes the module to inevitably reach s. ■

Theorem 4. *By employing Algorithms 1 and 2, all excluded modules are guaranteed to be removed.*

Proof: We prove the theorem by induction. If there are no excluded modules, nothing is to be shown. We assume that the theorem is true if there are $k \geq 0$ excluded modules. Let $|V^{\text{exc}}| = k + 1$. According to Lemma 1, the leader module identifies a module, m, to be removed, and activates this module. It follows from Lemma 2, that removal of module m does not cause the configuration to become infeasible. As stated by Lemma 3, module m traverses a finite path from its initial position to the sink. As only active modules move, and as m is the only active module, module m reaches the sink in finite time. Once the sink is reached, the module is removed, and deactivated, resulting in a configuration of only k excluded modules. For this configuration, we have that all excluded modules are guaranteed to be removed, meaning the proof is complete. ■

Theorem 5. *Using SAS, the number of time steps required to reach the desired configuration is bounded by $O(|V_0|^2)$.*

Proof: While executing Algorithm 2, the active excluded module moves along the perimeter of the remaining structure. Initially, this may happen in a clockwise direction. Once the module moves in a counter-clockwise direction around the structure, however, it keeps doing so until reaching the sink. The path length along the perimeter is bounded by $|V_0| + 1$, where V_0 is the initial configuration (defined in Section II-A). Therefore, the module must reach the sink in at most $2|V_0|+2$ time steps. As at most $|V_0|$ modules are to be removed, the number of time steps to reach the desired configuration is bounded by $2|V_0|^2 + 2|V_0| = O(|V_0|^2)$. ■

Fig. 5. Example of a complex shape with a long corridor that excluded modules must traverse. Dark purple modules are included modules. Light purple and cream colored modules are inactive and active excluded modules, respectively.

Fig. 6. Example of a user-defined U-shape being formed through SAS in simulation. Shown at time steps 0, 16, 26 and 39.

Theorem 6. *PAS guarantees that all excluded modules are removed, each following the same path as in SAS. The order in which modules arrive remains the same as in SAS.*

Proof: We prove the theorem by induction. If there are no excluded modules, nothing is to be shown. We assume that the theorem is true if there are $k \geq 0$ excluded modules. Let $|V^{\text{exc}}| = k+1$ and the modules be labeled in the order by which they would arrive at the sink when employing SAS: $1, \ldots, k+1$. If module $k+1$ was changed to be an included module, all statements would be true. That is, the k excluded modules would reach the sink at times $a_1 < a_2 < \cdots < a_k$, each one following the same path as in SAS. Algorithm 3 determines the arrival times $a_1 < a_2 < \cdots < a_k$ independent from whether module $k+1$ is excluded or not. We have $a_{k+1} = s_{k+1} + \Delta_{k+1} \geq \max(0, a_k + 2 - \Delta_{k+1}) + \Delta_{k+1} = \max(\Delta_{k+1}, a_k+2)$ (see line 4 of Algorithm 3). Hence, module $k+1$ starts no earlier than at time 0, and the order of arrival is preserved. If $a_{k+1} = a_k + 1$, one module would reside within the Moore neighborhood of the other, resulting in a collision. Hence, $a_{k+1} = \max(\Delta_{k+1}, a_k + 2)$ would be the earliest possible arrival time. The leader module emulates the movements of modules 1 to $k+1$ to check for possible collisions. We know that any possible collisions among modules 1 to k have already been resolved. If a collision involving module $k+1$ occurs, s_{k+1} (and hence a_{k+1}) is incremented and the process repeated (see line 8 of Algorithm 3). As $a_{k+1} = a_k + 2 + \Delta_{k+1}$ could not result in any collision, the number of iterations is bounded. In each iteration, at most $\Delta_{k+1} + \max_{j \leq k} \Delta_j$ steps have to be simulated: If module $k+1$ reaches the sink without any collision, arrival times $a_1 < a_2 < \cdots < a_k < a_{k+1}$ have been determined. Moreover, as no collision remains, none of the other excluded modules (1 to k) will ever reside within the Moore neighborhood of module $k+1$. In other words, module $k+1$ follows exactly the same individual path as for SAS The movements of module $k+1$ can hence not render a configuration infeasible, meaning the proof is complete. ■

Theorem 7. *Using SAS or PAS, in the worst-case, the number of time steps required to reach the desired configuration is* $\Theta(|V_0|^2)$.

Proof: According to Theorem 5, the number of time steps required to reach the desired configuration is $O(|V_0|^2)$. In other words, the time grows at most quadratically with the number of modules in the initial configuration. This result equally applies to PAS. What remains to be shown is that quadratic growth is indeed possible, that is, the number of time steps required to reach the desired configuration may be $\Omega(|V_0|^2)$. Consider the initial configuration shown in Figure 5(a). It comprises two horizontal blocks of included modules. Although each block contains 7 modules in Figure 5, in a more generalized configuration, each block contains $\frac{V_0^{exc}}{2}$ modules, where V_0^{exc} can be arbitrarily large. To obtain a lower bound for the number of time steps required to reach the desired configuration, we assume that the excluded modules in the upper block have already removed themselves from the structure, as shown in Figure 5(b). Any excluded module from the lower block will fully explore the upper corridor, as it cannot determine using local knowledge alone that this is a dead-end, seen in Figure 5(c). To prevent collisions, subsequent modules need to be sufficiently separated in time. Formally, the times at which any pair of subsequent modules arrive at the sink have to differ by $|V_0^{exc}|$. The $\frac{|V_0^{exc}|}{2}$ excluded modules in the lower block hence need at least $\big(\frac{|V_0^{exc}|}{2} - 1\big)|V_0^{exc}| = \Omega(|V_0^{exc}|^2)$ time steps to be removed. In this example $V_0 = 4 + 2|V_0^{exc}|$, as such the time steps required corresponds to $\Omega(|V_0|^2)$. ■

V. Simulation Studies

In this section, we present a number of simulation studies, demonstrating the capabilities of SAS and PAS. We implement the algorithms in a combination of Python and Fortran. The implementation allows the user to input a desired configuration (see Section V-A) or generate random configurations (see Section V-B), as well as graphically illustrating the paths of the excluded modules, if required.

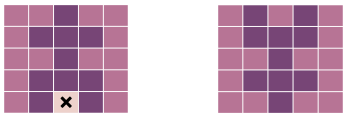
The performance of a simulation trial is quantified by the number of movements required for all excluded modules to be removed, referred to as *time steps*. To account for structures of different sizes, we present normalized values, that is, we divide the time steps by the number of excluded modules in the initial configuration. A simulation is considered successful if all excluded modules are removed, without the configuration becoming infeasible at any moment in time.

A. User-Defined Configurations

In this section, a range of user-defined desired configurations are investigated.

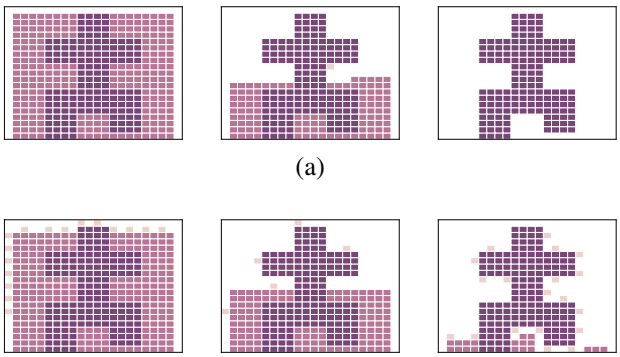
Figure 6 shows how SAS subtracts excluded modules to reconfigure from a 4×4 starting square to a simple U-shape, similar to the one considered by Gauci et al. [7]. Such a shape could be useful for collecting tasks or grasping tasks, once it has been formed. When employing SAS, the reconfiguration takes 39 time steps. The same simulation in PAS takes only 17 time steps, a reduction of 57%.

In the works on *Miche* and *Pebbles* by Gilpin et al. [8, 9], a humanoid structure is formed. As shown in Figure 7(a) this is



(a) (b)

Fig. 7. A humanoid shape configuration that is impossible to form due to the enclosed module indicated, (a), and a feasible rotated version of the same shape, (b).



(a)

(b)

Fig. 8. Example of (a) SAS, and (b) PAS, being used to form a large humanoid structure from a 20×20 grid. Shown at time steps (a) 0, 4000 and 8563, and (b) 13, 400 and 644.

not a feasible configuration in our problem formulation (which involves vertical 2-D structures and a ground), because the indicated module is enclosed by included modules and the ground. One possible solution is to rotate the configuration to a feasible one, shown in Figure 7(b). This reconfiguration from a 5 × 5 starting square takes 100 time steps using SAS, and can be seen animated in the supplementary material provided, along with animations of the other simulations presented here [10]. PAS takes only 40 time steps, a reduction of 60%.

Another solution, shown in Figure 8(a), could be to increase the size of the initial configuration and have one side of the desired configuration slightly removed from the ground. Here the initial configuration is set to 20×20 modules. It takes SAS 8563 time steps to reach this configuration. The same 20×20 humanoid shape was also simulated using PAS. From Figure 8(b) one can see the benefits of multiple modules moving at once. PAS takes only 777 time steps, a reduction of 91%.

B. Randomly Generated Configurations

This section presents studies where the desired configurations are randomly generated, making it possible to systematically characterize the performance under a wide range of conditions. We consider initial configurations of different sizes. Moreover, we consider desired configurations of different density. Density, ρ, is defined here as the percentage of modules within the initial configuration that are retained in the desired configuration, that is, the percentage of included modules. Formally, $\rho = \frac{|V^{\text{inc}}|}{|V^{\text{exc}} \cup V^{\text{inc}}|} \times 100\%$.

To generate a configuration, an initial included 'seed' module is first positioned on the ground. All neighbors of the seed module are checked to see whether they are not yet included, and are within the confines of the environment. From the potential neighbor locations, one is chosen at random to become a new included module. Once multiple included modules are selected, the next module to add a neighbor to is also randomly selected, and its neighbors assessed. This repeats until a configuration of given size is created, and guarantees that the configuration will be connected. However, this method does not ensure that no hollow spaces exist, so further checks must be performed to remove any infeasible desired configurations. Figure 9 shows an example of a randomly generated 10×10 configuration at 40% density.

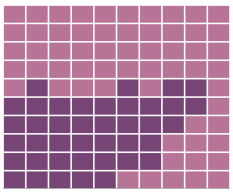


Fig. 9. Example of a randomly generated configuration, of size 10×10 and inclusion density of 40%.

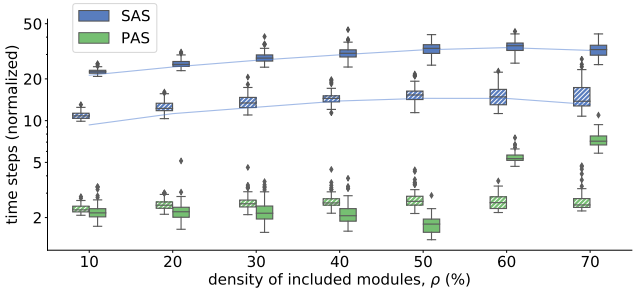


Fig. 10. Box-plots of the normalized time taken by excluded modules to remove themselves, leaving behind desired configurations of various densities. Initial configurations of 10×10 (shown with hatched boxes) and 20×20 (normal boxes). Desired configurations were generated randomly (80 samples per box). Line plots of the median best-case sequential time for each scenario are overlaid for the SAS simulations. Time steps are normalized by the number of excluded modules and shown on a logarithmic scale.

1) Influence of Density of Included Modules: We study the performance of SAS and PAS for different densities of included modules, from 10% to 70%, in steps of 10%. Two initial configurations are considered: 10×10 and 20×20. For each combination of density and initial configuration size, the same 80 randomly generated desired configurations are used for both approaches.

Figure 10 shows the effect the density has on the number of time steps it takes the modules to remove themselves from the structure. It also shows a line plot of the median *best-case sequential time* for the same generated configurations. The best-case sequential time is a theoretical lower bound for a given configuration, which is acquired by determining the shortest possible route that each excluded module could take to reach the sink, via empty cells or those occupied by excluded modules, while still circumnavigating the included modules.

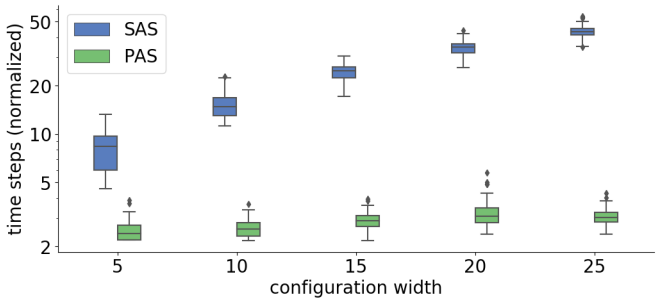


Fig. 11. Box-plots of the normalized time taken by excluded modules to remove themselves from configurations of various sizes. The desired configurations were generated randomly with density 60% (80 samples per box). Time steps normalized by the number of excluded modules.

2) Influence of Configuration Size: We study how the performance of SAS and PAS scale with the configuration size, considering 5×5, 10×10, 15×15, 20×20, and 25×25. For each case, the inclusion density is 60%. The same 80 randomly generated desired configurations are used for both approaches. The results are shown in Figure 11.

C. Discussion of Results

In every case simulated, both user-defined and randomly generated, all excluded modules were able to reach the sink and left behind only the desired configuration, suggesting that the algorithms performed as intended.

In Figure 10, the performance of SAS over varying density exhibits an interesting behavior, where it can be observed to rise to a point and fall again. We hypothesize that this is due to the configurations that can be generated rather than SAS itself. Desired configurations of low density may have less chance of containing overhangs and dead-ends, geometry that is time-consuming to circumnavigate. As the density then increases, so does the likelihood of creating more complex geometry. However, once it becomes very high, areas that featured complex geometry are more likely to be filled again. The non-monotonic behavior is also observed in the analysis of the best-case sequential times. When considering the best-case routes, the excluded modules must still go around the included modules.

PAS far outperforms SAS in terms of time steps taken for the same configurations, requiring the data to be shown on a logarithmic scale in order to be fully assessed. Moreover, the (normalized) reconfiguration time for PAS does not increase with configuration size; in fact, the larger configuration yields the best performance for densities up to 50% (see Figure 10).

In Figure 11 the change in performance attributed to configuration size can be clearly observed. When using SAS, the normalized time that excluded modules take to reach the sink increases with configuration size, though sub-linear, while the theory predicts a linear growth (or quadratic, if not normalized). However, when using PAS, the normalized time remained reasonably consistent. Although each module still individually traveled a further distance in a larger configuration, this also allowed for more modules to move in parallel, almost negating the effects of the increased travel distance. This is corroborated by the increased performance of PAS when simulating the configurations in Figures 7(b) and 8. These findings are promising, given that our theory predicted a linear growth (i.e., quadratic growth, if not normalized) in the worst-case scenario.

The trade-off between computational cost and performance means that both SAS and PAS have potential use cases. SAS is able to run on simple hardware at the expense of reconfiguration speed, yet is still able to reconfigure large structures. Moreover, the relatively small increase in performance that comes with using PAS over SAS in smaller systems may not justify the need for more capable modules. Whereas, for larger scale systems the improvement may be sizeable enough to negate the cost of complex modules. Where PAS proves computationally too demanding for a leader module, the computations could also be off-loaded to an external computer.

VI. Conclusion

In this paper we presented a subtraction approach by which extraneous modules actively remove themselves from a starting configuration, to leave behind a given structure. We refer to this approach as *active* subtraction. We presented two solutions, one with purely sequential, the other with parallel movement. We formally proved the correctness and characterized the worst-case performance of either solution. We conducted simulations that validated the solutions in a wider range of conditions, exploring the effect that varying the sizes and compositions of initial and desired configurations had on the time required for the modules to remove themselves.

In the future, we intend to deploy the algorithms on real-world systems, such as *HyMod*, exploiting its locomotion and in-place rotation capabilities. To achieve this, considerations of the realistic rigidity of the system will be simulated. A shortcoming of subtractive reconfiguration is that desired structures cannot contain hollow structures. However, by altering the algorithms here to allow modules to exchange places, it may be possible to produce shapes with hollow spaces. It is also planned to extend the algorithms to three dimensions. Furthermore, inspired by the concept from Casal et al. [3], reconfiguration between arbitrary structures may be possible using the presented solutions. By reversing the steps produced by the algorithms, and using the initial configuration as an intermediate structure, complete self-reconfiguration in finite time could be achieved, without heuristics or stochasticity.

Acknowledgments

M. D. Hall acknowledges support by the Engineering and Physical Sciences Research Council (EPSRC) through the EPSRC Doctoral Training Partnership National Productivity Scholarship.

References

[1] Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[2] Alberto Brunete, Avinash Ranganath, Sergio Segovia, Javier Perez de Frutos, Miguel Hernando, and Ernesto Gambao. Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems*, 14(3):1–21, 2017.

[3] Arancha Casal and Mark H Yim. Self-reconfiguration planning for a class of modular robots. In *Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 246–257. International Society for Optics and Photonics, SPIE, 1999.

[4] Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo. SWARMORPH-script: a language for arbitrary morphology generation in self-assembling robots. *Swarm Intelligence*, 2(2-4):143–165, 2008.

[5] Jay Davey, Ngai Kwok, and Mark Yim. Emulating self-reconfigurable robots-design of the SMORES system. In *Proc. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4464–4469. IEEE, 2012.

[6] Toshio Fukuda and Seiya Nakagawa. Dynamically reconfigurable robotic system. In *Proc. 1988 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1581–1586. IEEE, 1988.

[7] Melvin Gauci, Radhika Nagpal, and Michael Rubenstein. Programmable self-disassembly for shape formation in large-scale robot collectives. In *Proc. 13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 573–596. Springer, 2016.

[8] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, 2008.

[9] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot Pebbles: One centimeter modules for programmable matter through self-disassembly. In *Proc. 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2485–2492. IEEE, 2010.

[10] Matthew D. Hall, Anıl Özdemir, and Roderich Groß. Self-reconfiguration in two-dimensions via active subtraction with modular robots (supplementary video material), 2020. URL http://doi.org/10.15131/shef.data.12420326.

[11] Kazuo Hosokawa, Teruo Fujii, Hayato Kaetsu, Hajime Asama, Yoji Kuroda, and Isao Endo. Self-organizing collective robots with morphogenesis in a vertical plane. *JSME International Journal on Mechanical Systems, Machine Elements and Manufacturing (Series C)*, 42(1): 195–202, 1999.

[12] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.

[13] Christopher Parrott, Tony J Dodd, and Roderich Groß. HyMod: A 3-DOF hybrid mobile and self-reconfigurable modular robot and its extensions. In *Proc. 13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 401–414. Springer, 2016.

[14] John W Romanishin, Kyle Gilpin, and Daniela Rus. M-Blocks: Momentum-driven, magnetic modular robots. In *Proc. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4288–4295. IEEE, 2013.

[15] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *Proc. 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3293–3298. IEEE, 2012.

[16] Alexander Sproewitz, Philippe Laprade, Stéphane Bonardi, Mikaël Mayer, Rico Moeckel, Pierre-André Mudry, and Auke Jan Ijspeert. Roombots—towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules. In *Proc. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1126–1132. IEEE, 2010.

[17] Kasper Støy and Radhika Nagpal. Self-reconfiguration using directed growth. In *Proc. 7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 1–10. Springer, 2004.

[18] Kasper Støy, David Brandt, and David J. Christensen. *Self-Reconfigurable Robots: An Introduction*. The MIT Press, 2010.

[19] Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. Distributed self-reconfiguration using a deterministic autonomous scaffolding structure. In *Proc. 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 140–148. IFAAMAS, 2019.

[20] Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics and Automation Magazine*, 14(1):43–52, 2007.