Scalable Continuous Sculpting: Adaptive and Persistent Swarm Shape Formation Algorithms with Fixed Memory Dependence

Andrew G. Curtis[1], Michael Rubenstein[1]

[1]Center for Robotics and Biosystems,
Northwestern University, Evanston IL 60208, USA,
agc@u.northwestern.edu, rubenstein@northwestern.edu

Abstract. Robotic swarms often cite scalability as a benefit of their deployment. To realize the full extent of this benefit, swarm behavior algorithms have to be designed so that performance is not inhibited at scale. In this work, we examine a set of persistent and adaptable swarm shape formation algorithms that are limited by onboard memory when executed at scale. We improve these algorithms by using robot communication and collective knowledge to allow robots to form persistent and adaptive shapes with a fixed memory dependence. We demonstrate the improved algorithms in both simulation and on a swarm of mobile robots.

Keywords: Swarms; Path Planning for Multiple Mobile Robots or Agents; Distributed Robot Systems; Shape Formation

1 Introduction

Scalability is often an important aspect of robotic swarms. Swarm behaviors such as shape formation [1]–[3], path planning [4]–[6], foraging [7]–[9], and flocking [10]–[12] can often be executed with swarms ranging from a few robots to a few thousand (or more!). However, in some applications, the performance of the swarm may change with scale. For example, increasing scale may result in a higher resolution shape during shape formation [2], a lower mission success rate for swarm taxis [13], or reduced individual robot efficiency when foraging [7].

In this work, we examine swarm behavior scalability that is limited by the onboard memory ($\mathcal{M}$) of individual robots. Specifically, in [14], robots form dynamic shapes through a process called **continuous sculpting**. These shapes have a constant density per unit area, so "large" shapes are shapes with many robots covering a big area while "small" shapes are shapes with fewer robots covering a smaller area. When forming such shapes, $\mathcal{M}$ scales with the size of the shape by $O(N)$ where N is the number of robots needed to form the shape. To remove this limitation, we developed variations of the algorithms in [14] to allow for **scalable continuous sculpting** where $\mathcal{M}$ is fixed and does not scale with shape (or swarm) size: $O(1)$. This ensures that neither swarm behavior nor robot requirements must change when forming large shapes. This is especially

important as the field of swarm robotics is trending toward smaller, simpler robotic designs with less onboard memory [15]–[17].

After giving some additional background (Section 2), we elaborate on the process by which robots communicate and infer information (Section 3). This communication process allows the swarm to form a path through the shape using the collective knowledge of the swarm instead of individual robot memory. We then describe how robots use this communication process to form shapes that are both persistent (beyond the battery lifetime of each individual robot) and adaptable to shape changes (Section 4). Finally, we demonstrate the new fixed-memory algorithms in both simulation and on a swarm of mobile robots (Section 5) before concluding and identifying future work (Section 6).

2 Background

In a previous work, we introduced continuous sculpting: an activity in which a human can actively change a swarm into shapes that persist beyond the battery lifetime of individual robots (Fig. 1). To perform continuous sculpting, robots execute a set of decentralized algorithms, forming planar (i.e., non-intersecting) Hamiltonian cycles that both approximate the shape and facilitate robot movement through the shape and back to a charging station. This allows the swarm to form shapes persistently as robots can continuously recharge and re-enter the shape. When a human initiates a change to the shape, the swarm detects and reacts to the change, adapting to a new Hamiltonian cycle so that the swarm can continue to persist. The underlying persistence and adaptability algorithms were proven theoretically and demonstrated on a swarm of mobile robots in [14].

In those algorithms, robots navigate through a virtual square grid graph ($G = (n, e)$) overlaid on the environment, occupying grid graph nodes to represent the shape (S) where $S \subset G$. Each shape is constructed from boxes (B) such that each box consists of four grid nodes (n) in a 2 by 2 configuration and each box meets the full side of another box. To navigate through such shapes, robots plan and move one edge (e) at a time using local information and data stored in memory. Specifically, robots store a list of in-shape nodes (S), a list of nodes that the robot has already visited (V_n) such that $V_n \subset S$, and a tree data structure of previously visited boxes (V_B). The required memory for S, V_n, and V_B scales with shape (and swarm) size: $O(N)$.

In this work, we alter the persistence and adaptability algorithms so their memory dependence is fixed: $O(1)$. This concept is not new to robotics research. Others have investigated foraging [7], traversing directed graphs [15], and mobile robot dispersion [18] with limited memory on each robot. In most cases, memory was a variable that could be tuned to study performance against memory size [7], [18]. In other cases, onboard memory was replaced with stigmergy or other environment markers [15], [19]. In this paper, we use robot-to-robot communication to augment onboard memory, allowing robots to infer where they have been based on information received from their neighbors instead of relying on information stored in memory.

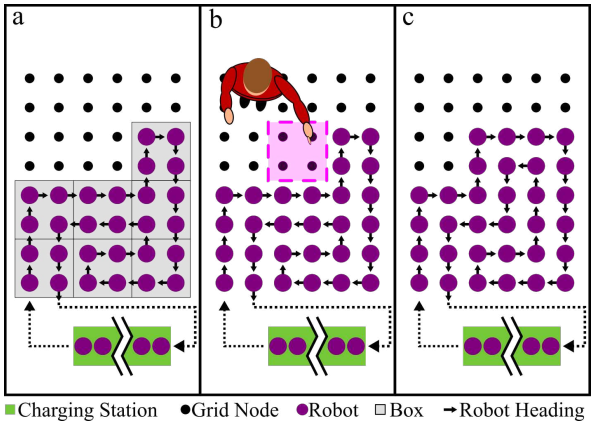

Fig. 1. A graphical representation of continuous sculpting modified from [14]. (a) A shape of 7 boxes. Robots occupy grid graph nodes to form the shape. Robot headings indicate the swarm path through the shape. Dashed arrows show robot paths to and from the charging station. (b) A human initiates an addition to the shape (purple box). (c) The swarm travels a new path through the new shape.

3 Robot Memory

Our goal for this work was not to change the logic of the algorithms in [14], but rather to have robots infer s, v_n, and v_B, subsets of S, V_n, and V_B, respectively, using robot-to-robot communication. These subsets still contain enough information for each robot to follow the continuous sculpting algorithms outlined in [14], but they are limited to a fixed memory size so onboard memory does not scale with the number of grid nodes representing the shape. In this section, we address how these subsets are found and provide an example.

Additionally, we define a **neighborhood** as the eight grid nodes surrounding any given node in G. For example, in Fig. 2a, the robots labeled 0, 1, 2, 14, 15, 20, 21, and 23 are all at grid nodes in the neighborhood of the robot labeled 22. Like our previous work, robots are only capable of communicating with other robots in their neighborhood as the communication range is limited to $\sqrt{2}l$ where l is the length of an edge in G. During the process of finding s, v_n, and v_B, robots will transmit a message to all robots in their neighborhood. We refer to this message as a **broadcast message**. Finally, we make the following assumptions about robot capabilities:

1. Robots can determine the clockwise direction around the center of a box.
2. Robots map the same 2 by 2 configuration of nodes to the same box.
3. Robots are capable of identifying a box given a node (and vice versa) by evaluating its (x, y) position with a modulus of 2.

Algorithm 1 Finding s, v_n, and v_B

Require: $(x_r, y_r), C_r, C_{min_r}$ ▷ Information provided from a neighbor robot
Ensure: s, v_n, v_B, C_{min} ▷ Updated information about the robot and its neighborhood
▷ Assume robot has edge count C and in-box minimum count C_{min}
1: **if** $(x_r, y_r) \notin s$ **then** include (x_r, y_r) in s ▷ Update s
2: **end if**
3: **if** $C_r < C$ **and** $(x_r, y_r) \notin v_n$ **then** include (x_r, y_r) in v_n ▷ Update v_n
4: **end if**
5: **if** $C_r < C_{min}$ **and** (x_r, y_r) in my box **then** $C_{min} \leftarrow C_r$ ▷ Overwrite C_{min}
6: **end if**
7: **if** $C_{min_r} < C$ **and** (x_r, y_r) **not** in my box **and** $\text{Box}(x_r, y_r) \notin v_B$ **then**
8: include $\text{Box}(x_r, y_r)$ in v_B ▷ Update v_B
9: **end if**

3.1 Identifying In-Shape Nodes: $s \subset S$

To make continuous sculpting more scalable and limit the required memory, robots infer s based on the presence of other robots. This process begins with each robot broadcasting its current node location (x, y) to robots in its neighborhood (i.e., its neighbors). Each robot then builds a mental map of occupied nodes (nodes from which a message was received) and unoccupied nodes (nodes from which a message was not received). From this, each robot infers that occupied nodes are a part of the shape and unoccupied nodes are not. Robots then store the list of occupied nodes as s. This process is summarized in Algorithm 1 lines 1-2 where (x_r, y_r) is the node position received from a robot in the neighborhood. Since communication range is limited to a neighborhood, the memory required for s is fixed to no more than eight locations. After moving to a new grid node, each robot clears s and builds a new s in its new neighborhood.

3.2 Identifying Visited Nodes: $v_n \subset V_n$

To understand how a robot can infer v_n, let's consider the path robots travel according to the original continuous sculpting algorithms. The path is a planar Hamiltonian cycle that is broken at adjacent entry and exit nodes such that the entry and exit nodes are in the same box and are on the periphery of the shape. The robot occupying the entry node has yet to traverse any edges in the shape, while the robot occupying the exit node has traversed some number of edges, $\mathcal{E}$, in order to visit each node in the shape exactly once. In fact, every robot along the path from entry node to exit node will have visited some number of edges, μ, such that $0 \leq \mu \leq \mathcal{E}$. If we display the number of edges each robot has traveled, we see that edge counts create a monotonically increasing sequence along the path from entry node to exit node (Fig. 2a).

Robots can use this information to infer which nodes they have visited (and which they have not) instead of keeping that information in memory. When traveling through the shape, robots count the number of edges they traversed. Then, robots broadcast their own edge count in their broadcast messages. Robots

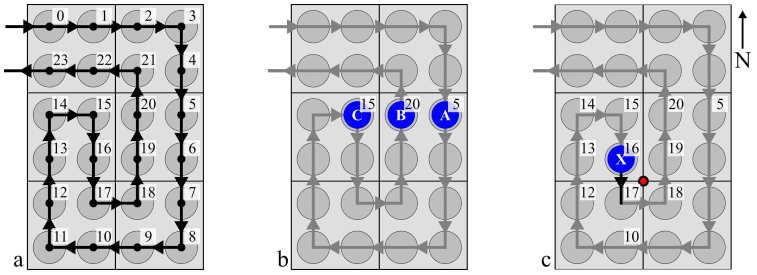

Fig. 2. (a) A monotonically increasing sequence of edge counts along the path through the 6-box shape. Grey circles indicate robots. Arrows indicate path direction. Numbers indicate edge count values. (b) Example to show how robot C identifies that robot B's box has been visited. (c) Robot X identifies the node to its south as the node in its parent box due to a counter-clockwise priority around the red virtual box vertex.

compare their own edge count (C) with each edge count they receive (C_r). If a received edge count is less than their own, then the robot can infer it has visited the corresponding node. Conversely, if a received edge count is greater than their own, then the robot can infer it has not yet visited that node. Since the swarm edge counts are monotonically increasing along the path through the shape, and all robots increase their edge counts at the same rate, it is impossible for two robots to have the same edge count. It is also impossible for a robot to receive an edge count lower than its own from a node it has not already visited or an edge count higher than its own from a node that it has already visited.

Thus, robots create v_n by recording each node from which they received an edge count lower than their own (Algorithm 1 lines 3-4). Since communication range is limited, v_n is fixed to no more than eight locations, and each robot clears v_n and builds a new v_n after moving to each new grid node.

3.3 Identifying Visited Boxes: $v_B \subset V_B$

The process for identifying v_B also uses edge counts. Since robots can identify which nodes belong to which boxes, they can identify if a neighbor is in their own box (in-box) or in a different box. When a robot receives a broadcast message from an in-box neighbor, it evaluates if the received edge count value is less than its own estimated in-box minimum edge count (C_{min}). If the received edge count is less than C_{min}, the robot overwrites C_{min} to the received value. Each robot then transmits C_{min} in its broadcast message.

Robots use minimum in-box edge count values to determine boxes that have been visited instead of keeping that information in memory. If a robot receives a minimum edge count less than its own edge count from a robot in a different box, then the box from which the message originated must be a visited box. Robots can record this information to create v_B. The processes for overwriting C_{min} and updating v_B are given in Algorithm 1 lines 5-6 and 7-9, respectively. Further, since a neighborhood can only include nodes from 4 boxes at most, v_B

is fixed to no more than four boxes. For example, the neighborhood of the robot with edge count $C = 15$ in Fig. 2a consists of robots from its own box (count numbers 13, 14, and 16) and three other boxes (count numbers 19-23). Finally, like s and v_n, v_B is cleared and rebuilt at each new grid node as robots move through the shape. C_{min} is also re-initialized to C after each robot movement.

3.4 Example: Finding s, v_n, & v_B

Consider an example in Fig. 2b. Assume the robots just arrived at their current nodes. Robots A and B are in the same box. To start, both robots believe their own edge count is the smallest in the box, so C_{min_A} is initialized to C_A and C_{minB} is initialized to C_B. Robot A will broadcast its position, its own edge count ($C_A = 5$), and its assumed in-box minimum ($C_{min_A} = 5$). Robot B will broadcast its position, $C_B = 20$, and $C_{min_B} = 20$. Upon receipt of the message from robot A, robot B does three things: 1) it records that robot A's node is within the shape (s), 2) it records robot A's node in v_n (because $C_A < C_B$), and 3) it changes its in-box minimum to $C_{min_B} = C_A = 5$ (because $C_A < C_{min_B}$ and robots A and B are in the same box). Robot B does not need to add the box occupied by robot A into v_B since robot B and robot A share a box. At this point, robot B is now broadcasting its position, $C_B = 20$, and $C_{min_B} = 5$. Then, when robot C receives a message from robot B, robot C processes the message in three parts: 1) it records that robot B's node is within the shape (s), 2) it does **not** append robot B's node to v_n (because $C_B > C_C = 15$), and 3) it records robot B's box in v_B (because $C_{min_B} < C_C$).

Because the communication range is limited to $\sqrt{2}l$, it takes two communication "rounds" for robot C to know that it has visited robot B's box. In the first round, robot B is broadcasting its own edge counter as minimum (as it has not yet heard from robot A), so robot C will assume that robot B's box has not yet been visited (because $C_C < C_{min_B} = 20$). Then, in the second round, robot B has heard from A and is broadcasting a minimum of $C_{min_B} = 5 < C_C$, so robot C will assume that robot B's box has been visited and change v_B accordingly. Given this delay, it is important that robots have enough time to receive all pertinent information before deciding what to do next. For reference, in the simulations and demonstrations presented in this paper, robots transmit messages at approximately 4 Hz which would indicate that robots could move approximately once per 0.5 s (accounting for 2 rounds of communication and algorithm processing per movement). However, our robots move every 12 s to 15 s to account for any dropped messages in our communication system and the velocity and turning constraints of our nonholonomic robots.

4 Persistence and Adaptability

When executing the persistence and adaptability algorithms in [14], robots query data about their neighborhood from S, V_n, and V_B to find and follow paths through the shape. This allows the swarm to form any size valid shape via any

number of additions or subtractions to the shape in any order, as proven in [14]. However, because only neighborhood data is used to inform each robot decision, robots need not store S, V_n, and V_B in memory. Robots can instead use s, v_n, and v_B to get information about their neighborhood. This means the algorithms governing persistence and adaptability in this paper are largely unaltered from their original form with a few minor differences described in this section.

4.1 Persistence

The persistence algorithm in [14] is called the **default behavior**, and consists of three rules, given below, where the **parent box** is the box that the robot had visited before moving into its current box for the first time. The default behavior rules are executed distributively and in real time to form a planar Hamiltonian cycle through the shape called a **preferred path**. In order to execute the default behavior, robots require information about their current box (b), S, V_n, and V_B (s, v_n, and v_B in the fixed memory case). More details can be found in [14].

IF an edge leads to a box not visited, take that edge. (Rule 1)
ELSE IF an edge leads the robot clockwise within its current box, take that edge. (Rule 2)
ELSE IF an edge leads to the parent box, take that edge. (Rule 3)

In order to remove the memory scalability limitation of the default behavior, two minor changes are required: the addition of a message passing period and a new way to identify the parent box. The reason for the first change is trivial; robots need some idea of S, V_n, and V_B to execute the default behavior. Since robots cannot store these lists in memory, they have to infer subset lists from their neighbors using the processes described in Section 3. Therefore, each robot participates in a short communication period to develop s, v_n and v_B before executing the default behavior described in [14].

The only other challenge to the fixed-memory version of the default behavior is distinguishing which previously visited node is in the parent box without storing the visited box tree data structure (i.e., V_B) in memory. To resolve this, robots infer their parent box based on Lemma 8 from [14] which states that, for any given node in a preferred path, there is only one valid outgoing edge that spans into a different box.

For example, consider robot X with edge count $C_X = 16$ in Fig. 2c. Robot X has already visited the nodes to its north (N) and west (W) because their edge counts are lower than C_X. The nodes to its east (E) and south (S) are unvisited because their edge counts are higher than C_X. Both unvisited nodes are in previously visited boxes because the minimum counts in each box (5 and 10) are lower than C_X, so default behavior rule 3 applies. Of the two options, only the S node satisfies Lemma 8, so the robot can correctly infer that the S node is in its parent box. In practice, this is done by choosing the edge that results in counter-clockwise motion around the nearest virtual box vertex (Fig. 2c).

4.2 Adaptability

The adaptability algorithms in [14] govern the swarm's behavior as it adapts to shape changes (i.e., box additions or box subtractions). The adaptability algorithms are divided into three sequential parts: detection, primary changes, and secondary changes. During detection, robots adjacent to the added or removed box identify the change as either an addition or a subtraction. Then, during primary changes, robots either fill in the new box or file out of the removed box. Finally, during secondary changes, the swarm adjusts its path (if necessary) to the preferred path of the new shape. Secondary changes are completed either via robot communication (the communication-based method) or via robot motion (the movement-based method). More details can be found in [14].

For this paper, we alter the adaptability algorithms to permit the use of the fixed-memory subsets s, v_n, and v_B. Details of these alterations are provided in this section, followed by some examples. There are no alterations to detection since shape changes are detected locally (i.e., by robots closest to the change) without requiring S, V_n, or V_B. Finally, for this discussion, we use **upstream** to refer to a robot that has yet to visit a particular grid node, and **downstream** to refer to a robot that has previously visited a particular grid node.

Alterations to Primary Changes Per [14], once a change is detected, it is communicated through the swarm with the coordinates of a critical node, n_{cp}. The critical node is adjacent to the added or removed box at the beginning of primary changes and serves as a reference point for other robots to determine if they are upstream, downstream, or located at n_{cp}.

With unlimited memory, robots can simply check if n_{cp} is in V_n to determine if they are upstream, downstream, or at n_{cp}. To remove this memory dependence, robots instead use $C_{n_{cp}}$ (the edge count of the robot at n_{cp}). Once the change is detected, the robot that detected the change identifies n_{cp} and communicates $C_{n_{cp}}$ through the swarm via repeated robot-to-robot messages. Each robot checks its own edge counter C against $C_{n_{cp}}$. If $C < C_{n_{cp}}$, it is an upstream robot; if $C > C_{n_{cp}}$, it is a downstream robot; and if $C = C_{n_{cp}}$, it is at the critical node. Once they have determined if they are upstream, downstream, or at n_{cp}, robots can execute the default behavior, fill in the new box, file out of the removed box, or remain still per the the primary changes algorithm in [14].

The only other alteration to primary changes is that robots must adjust their edge count values to reflect the change to the shape. In the case of addition, once a new box is filled in, all robots downstream of n_{cp} add four to their edge count (and minimum in-box edge count) so that it appears as if they have already moved through the four nodes of the newly added box before reaching their current position. This maintains a monotonically increasing edge count sequence through the swarm along the interim path that is established once the new box is filled in. In the case of subtraction, robots that file out of the removed box inherit the edge count of the downstream robot that they followed out of the removed box. This ensures downstream robots always have an edge count value higher than (or equal to) $C_{n_{cp}}$ after the removed box is emptied.

Alterations to Secondary Changes In order to describe the alterations to secondary changes, let us first explain the original secondary changes algorithm in more detail. Once the new box is filled in (or the removed box is emptied), secondary changes begin. Secondary changes are responsible for changing the interim path established after primary changes to the preferred path of the new shape. This allows the swarm to execute the default behavior after a shape change is resolved. The behaviors for secondary changes are agnostic of the type of shape change (i.e., addition or subtraction) and are realized via one of two interchangeable methods: communication-based or movement-based.

Both methods operate under the same tenet: incrementally change the path by moving a "change carrier" through the swarm. In the communication-based method, the change carrier is a message that is sent through the swarm robot-to-robot. In the movement-based method, the change carrier is a robot that adjusts the paths of other robots in its neighborhood as it moves. For both methods, the change carrier is established at a secondary change start node (SCSN) immediately after primary changes. The robot at the SCSN represents the first robot that can travel through the new shape using only its default behavior because all robots upstream of its position are not impacted by the shape change, and all downstream robots may be impacted by the shape change.

Instead of focusing on the change carrier as either a message or a robot, we can generalize the discussion by considering the change carrier as a moving point of inflection in the swarm. The change carrier is always at some node between the SCSN and the exit node. When at a node, the change carrier itself and all robots upstream of its position are executing the default behavior, while all robots downstream of its position are executing potentially non-default behavior as they await any path transformations initiated by the change carrier. For brevity, we will consider this general change carrier case to describe how the adaptability algorithms have been altered for fixed onboard memory.

Unfortunately, in the fixed-memory version of secondary changes, robots cannot rely on neighbor edge count values (and minimum in-box edge count values) to infer v_n (and v_B). This is because, at any given point during secondary changes, the robots may not be traveling along a path with a monotonically increasing sequence of edge count values. Thus, we require additional information for robots to infer v_n (and v_B) in the face of transforming edge count sequences. Specifically, we require robots to categorize themselves based on their relative location to the change carrier (i.e., upstream or downstream). Robots can then use these categories to artificially inflate or deflate edge count values received from their neighbors to correctly infer s, v_n, and v_B.

Robots assign themselves a category as follows. By default, all robots are in category 0. Then, after a change carrier is established at the SCSN, it broadcasts a message with its current edge count value (C^*) and the minimum edge count identified within its box (C^*_{min}). Robots re-transmit the message robot-to-robot throughout the swarm, flooding the shape with the information. Robots evaluate their edge count (C) and minimum in-box edge count (C_{min}) relative to C^* and C^*_{min} to put themselves into one of the following categories.

Algorithm 2 Adjusting Received Count Values & Changing Category

Require: $C_r, C_{min_r}, \kappa_r, (x_r, y_r)$ ▷ Information received from a neighbor robot
Ensure: $C'_r, C'_{min_r}, \kappa'$ ▷ Adjusted count values and new category
▷ Assume robot has edge count C, in-box minimum count C_{min}, and category κ
1: **if** ($\kappa = 0$ **or** $\kappa = 1$) **and** $\kappa_r \geq 2$ **and** $C_r \leq C$ **then** $C'_r \leftarrow C_r + C$
2: **end if**
3: **if** ($\kappa = 0$ **or** $\kappa = 1$) **and** $\kappa_r = 2$ **and** $C_{min_r} < C$ **then** $C'_{min_r} \leftarrow C_{min_r} + C$
4: **end if**
5: **if** ($\kappa = 2$ **or** $\kappa = 3$) **and** $\kappa_r \leq 1$ **and** $C_r > C$ **then** $C'_r \leftarrow C - 1$
6: **end if**
7: **if** $\kappa = 2$ **and** $\kappa_r \leq 1$ **and** $C_{min_r} > C$ **then** $C'_{min_r} \leftarrow C_{min} - 1$
8: **end if**
9: **if** $\kappa = 2$ **and** $\kappa_r \leq 1$ **and** (x_r, y_r) in my box **then** $\kappa' \leftarrow 3$
10: **end if**

- **Category 0:** All robots are category 0 by default
- **Category 1:** Robots upstream of or at the change carrier: $C \leq C^*$
- **Category 2:** Robots downstream of the change carrier and in a box the change carrier has not visited: $C > C^*$ and $C_{min} > C^*_{min}$
- **Category 3:** Robots downstream of the change carrier and in a box the change carrier has visited: $C > C^*$ and $C_{min} \leq C^*_{min}$

Once a robot leaves the shape for the charging station, it reverts to Category 0 so that it has the default category when it re-enters the shape after recharging.

Robots include their category ($\kappa = 0, 1, 2$, or 3) in their broadcast message with their own edge count, minimum in-box edge count, and location. Robots log the categories of their received neighbors, but since communication range is limited, the onboard memory required for this is fixed to eight slots (one for each neighbor) and does not scale with shape (or swarm) size.

Robots adjust received edge count values based on robot category via Algorithm 2. These adjustments, for example, make sure that a robot in category 1 (upstream or at the change carrier) will always view a robot in category 2 (downstream of the change carrier) as having a greater edge count than its own to properly infer v_n. Additionally, robot categories may change as the change carrier moves through the shape. A robot in category 2 that passes into the same box as a robot in category 1 or category 0 (either the change carrier or a robot upstream of it) will become a category 3 robot because its minimum in-box edge count value (C_{min}) will be less than (or equal to) that of the change carrier's (C^*_{min}). As such, category 2 robots may convert to category 3 as the change carrier moves per Algorithm 2 lines 9-10. After executing Algorithm 2, robots can execute Algorithm 1 to get s, v_n, and v_B as they perform the secondary change behaviors described in [14]. Note that unless a change has recently occurred, all robots are category 0, and no artificial inflating or deflating is required.

The only other alterations to the secondary change algorithms in [14] are associated with overwriting memory. In the original algorithms, a robot might "pass back" its memory and its position to an upstream robot. The recipient then

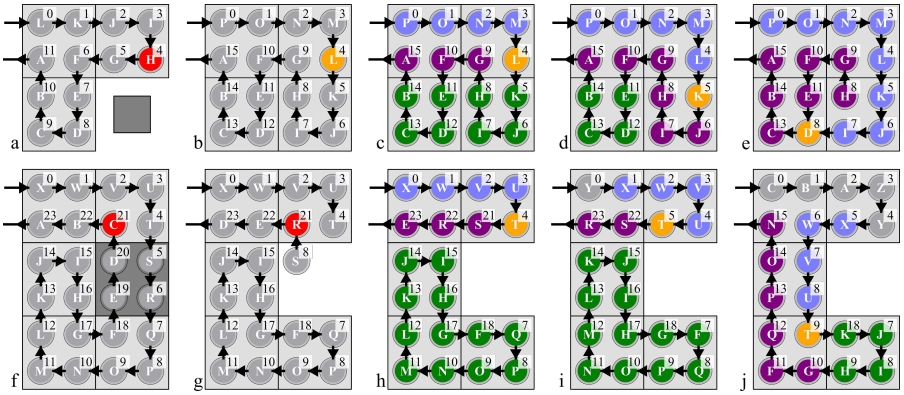

Fig. 3. Example box addition (panes a-e) and subtraction (panes f-j). Robot colors indicate the critical node (red), the change carrier (orange), and robot category (0=grey, 1=blue, 2=green, 3=purple). Arrows indicate path direction. Letters indicate unique robot ids. Numbers indicate non-adjusted edge counts.

follows the "pass back" robot to the pass back robot's node and overwrites its memory with that of the pass back robot. This facilitated non-default behavior of robots downstream of the change carrier in the movement-based method. Likewise, in the communication-based method, robots in receipt of the change carrier message overwrote their memory with that of the senders. They then appended their own location and box information to their new memory before passing the message further downstream. In this paper, we have altered these behaviors to use edge counts instead of memory lists. Specifically, in a pass back scenario, a robot passes back its location and edge count information. The recipient follows the sender and overwrites its edge count to that of the sender. Similarly, in the communication based method, robots overwrite their edge count (and minimum in-box edge count) to that of the sender. They then add one to the edge count value before passing the message further downstream.

Shape Change Examples We provide two shape change examples to demonstrate the preceding adaptability concepts: one addition and one subtraction. A few important snapshots of each change process are provided; the rest are omitted for brevity. Description of the behaviors is in accordance with the original algorithms in [14] and the alterations presented in this paper. The first example is a box addition captured in the top row of Fig. 3.

In Fig. 3a, a box (dark grey) is added to the shape, and robot H is located at n_{cp}, so $C_{n_{cp}} = 4$. Robots with edge count values less than 4 (robots I, J, K and L) are upstream, so they execute the default behavior and file into the new box after passing through n_{cp}. Robots with edge count values greater than 4 (robots A through G) are downstream robots. They remain still as the new box is filled and then inflate their edge count values by 4 so that there is a monotonically increasing sequence of edge count values along the new interim path (Fig. 3b). At

this point, primary changes are finished, and the change carrier is identified at the SCSN. We then employ the communication-based method for this example.

In Fig. 3c, robots determine their category with respect to $C^* = 4$ and $C^*_{min} = 2$. Per the communication-based method, the change carrier then sends a message robot-to-robot through the swarm along the preferred path of the new shape. Fig. 3d shows the state of the swarm after the message has traveled from robot L to robot K. Since robot K is now the change carrier (and category 1), all robots in its box convert to category 3 per Algorithm 2 lines 9-10. Fig. 3e shows the state of the swarm after the message has traveled 3 more edges through robot J and robot I to robot D. As before, category 2 in-box robots convert to category 3. Further, per Algorithm 2 lines 1-2, robot D will inflate the count value of robot H prior to determining v_n. In other words, even though robot H has the same edge count value as robot D (the current change carrier), robot D still evaluates robot H's node as an unvisited node.

The second example is a box subtraction. In Fig. 3f, a box (dark grey) is removed from the shape and primary changes begin. Robot C is at n_{cp}, so $C_{n_{cp}} = 21$. As robots file out of the removed box, they inherit an edge count value of $C = 21$ and continue counting as they move. For example, robot S assumes $C = 21$ as it exits the removed box. Robot R did the same and then shows a count $C = 22$ one step later (Fig. 3g to Fig. 3h).

As secondary changes begin (Fig. 3h), robots determine their category with respect to the edge count values of the robot at the SCSN (robot T in this case). For this example, we employ the movement-based method, so all robots move as the path change is addressed. Fig. 3i shows the state of the swarm after each robot has moved one edge. The category 2 robots are in a loop separate from the rest of the shape due to pass back behavior. Therefore, robot F overwrites its edge count of 19 with the preceding robot's count of 7. Fig. 3j shows the state of the swarm 4 movements later. Robot T receives $C_{r_{min}} = 7$ from robot K which would indicate that robot T has already traveled in robot K's box because $C_T = 9 > 7$. However, because robot K is category 2 and robot T is category 1, robot T inflates the received value per Algorithm 2 lines 3-4 to correctly identify robot K's box as unvisited. Robot T enters the unvisited box its next move.

5 Simulations and Demonstrations

We demonstrated scalable continuous sculpting in both simulation and on a swarm of physical ground robots called *Coachbot V2.0* [3]. Each *Coachbot* is equipped with a two-wheeled differential drive system, onboard position and orientation (x, y, θ) sensing, an onboard battery, an onboard Raspberry Pi computer, and an onboard Wi-Fi module for robot-to-robot communication. The physical experiments were performed with similar parameters as the experiments in [14]: a grid graph of l =0.2 m and a communication range of 0.3 m.

For the physical robot experiments, a human initiated shape changes via both the movement-based and communication-based methods while robots cycled to and from a fixed charging rail (video: [20]). The experiments were similar to those

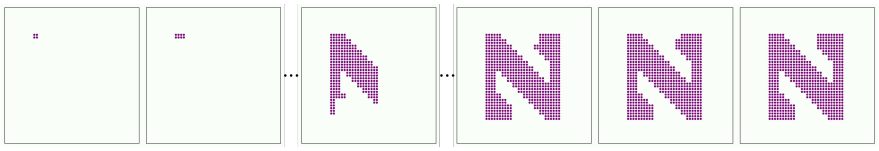

Fig. 4. Sequence of shape additions to build a shape from 4 to 632 robots.

in [14], and the fixed-memory algorithms resulted in nearly identical behavior to the original algorithms. The only minor differences were related to the stochastic nature of robot clock synchronization and human behavior. The near identical behavior indicates that the fixed memory algorithms are just as capable as their predecessors without the required memory overhead.

In simulation, we demonstrated scalability by sculpting a large "N" and a large "U" from just a single box of four robots (video: [21]). Every robot was programmed to know the entry and exit nodes (same as in [14]), and they inferred the initial one-box shape. Once robots began cycling between the initial shape and the charging station, a virtual human began sculpting the shape into the letter "N" formed by 632 robots. Fig. 4 shows the sequence of building the "N." The "N" was then converted through a series of shape changes (using the communication-based method) to a "U" formed by 464 robots. The charging station was omitted for brevity, and robots simply "appeared" at the shape fully charged and "disappeared" when leaving the shape.

The simulation shows the inherent scalability of the fixed-memory algorithms as the swarm scaled from 4 to 632 to 464 robots in a single experiment. Additionally, the swarm demonstrated persistence and adaptability as robots continuously cycled in and out of the shape over the course of 181 additions and 66 subtractions to sculpt the "N" and the "U."

6 Conclusion

The algorithms presented in this paper facilitate adaptive and persistent swarm shape formation with a fixed memory dependence. Since the required onboard memory does not inhibit scalability, practical swarms need not overburden their robots with excessive memory. This reduces cost and complexity, making it easier to employ continuous sculpting algorithms in applications such as agricultural monitoring and emergency response ad-hoc communication networks.

Finally, by reducing the required memory to execute continuous sculpting, we have brought our work one step closer to execution on a swarm of physical flying robots since we can now build a swarm of simple flyers with limited processing and memory capabilities. In addition to execution on a flying swarm, we intend to continue work on a 3D version of continuous sculpting and the challenges associated with a human interacting with flying robots.

References

[1] H. Xu *et al.*, "A multi-robot pattern formation algorithm based on distributed swarm intelligence," in *IEEE ICCSEA*, 2010.

[2] H. Wang and M. Rubenstein, "Generating goal configurations for scalable shape formation in robotic swarms," in *DARS*, Springer, 2022.

[3] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE T-RO*, 2020.

[4] S. Morgan and J. Hereford, "Path formation using a robot swarm with limited sensing capabilities," in *2020 SoutheastCon*, IEEE, 2020.

[5] W. Hönig *et al.*, "Trajectory planning for quadrotor swarms," *IEEE T-RO*, 2018.

[6] B. Araki *et al.*, "Multi-robot path planning for a swarm of robots that can both fly and drive," in *IEEE ICRA*, 2017.

[7] J. P. Hecker and M. E. Moses, "Beyond pheromones: Evolving error-tolerant, flexible, and scalable ant-inspired robot swarms," *Swarm Intelligence*, 2015.

[8] Q. Lu, J. P. Hecker, and M. E. Moses, "Multiple-place swarm foraging with dynamic depots," *Autonomous Robots*, 2018.

[9] Q. Lu *et al.*, "Swarm foraging review: Closing the gap between proof and practice," *Current Robotics Reports*, 2020.

[10] S. Hauert *et al.*, "Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate," in *IEEE IROS*, 2011.

[11] F. Schilling, F. Schiano, and D. Floreano, "Vision-based drone flocking in outdoor environments," *IEEE RA-L*, 2021.

[12] C. Virágh *et al.*, "Flocking algorithm for autonomous flying robots," *Bioinspiration & biomimetics*, 2014.

[13] J. D. Bjerknes and A. F. Winfield, "On fault tolerance and scalability of swarm robotic systems," in *DARS*, Springer, 2013.

[14] A. G. Curtis, M. Yim, and M. Rubenstein, "Continuous sculpting: Persistent shape formation adaptable to human-swarm interaction," *arXiv preprint arXiv:5512447*, 2024.

[15] P. Fraigniaud and D. Ilcinkas, "Digraphs exploration with little memory," in *STACS*, Springer, 2004.

[16] H. Hamann, *Swarm robotics: A formal approach*. Springer, 2018.

[17] W. Cha *et al.*, "Carbon fiber–aluminum sandwich for micro-aerial vehicles and miniature robots," *MRS Advances*, 2021.

[18] J. Augustine and W. K. Moses Jr, "Dispersion of mobile robots: A study of memory-time trade-offs," in *ICDCN*, 2018.

[19] J. Werfel and R. Nagpal, "Extended stigmergy in collective construction," *IEEE Intelligent Systems*, 2006.

[20] A. G. Curtis and M. Rubenstein, *Scalable continuous sculpting with mobile robots*, 2024. [Online]. Available: `https://youtu.be/DXFS2vgs9TA`.

[21] A. G. Curtis and M. Rubenstein, *Scalable continuous sculpting simulation*, 2024. [Online]. Available: `https://youtu.be/yRYwIwJXYtc`.