An Asymmetric Distributed Method for Sorting a Robot Swarm
Yu Zhou\textsuperscript{1}, Ron Goldman\textsuperscript{1}, and James McLurkin\textsuperscript{1}

Abstract—We consider the problem of sorting a swarm of labeled robots in Euclidean space. Our goal is to organize the robots into a sorted and equally-spaced straight line between the robots with lowest and highest labels, while maintaining a connected communication network. We break the symmetry between the minimum and maximum, in order to keep time, travel distance, and communication costs low, without using central control. We run in parallel a set of functions, including leader election, tree formation, path formation, path modification, and geometric straightening. We show that our algorithm is safe, correct, efficient, and robust to changes in population and connectivity. We validate our theoretical results with simulations. Our implementation uses messages of fixed size and robots of constant memory, and is a practical solution for large populations of low-cost robots.

Index Terms—Distributed robot systems, path planning for multiple mobile robots or agents, swarms, physical sorting, distributed algorithm.

I. INTRODUCTION AND RELATED WORK

ARRANGING robots into a specific order is a necessary routine in multi-robot applications. If physically homogeneous robots need to perform sequential procedures, they can simply swap tasks instead of moving physically; however, swapping tasks requires consensus, and transferring data is expensive. Sorting is required if robots have different structures, are carrying different physical loads that cannot be swapped, and need to arrive in some order to remotely build an assembly line. Sorting is also required if the robots differ in other intrinsic quantities, such as remaining battery level that cannot be transferred via communication [1].

We describe a sorting algorithm to arrange robots into a sorted line, and demonstrate that this algorithm is robust and safe for any arbitrary initial robot configuration in Euclidean space. Because we make minimal assumptions on the robots’ ability to communicate and compute, our approach is suitable for simple robots with limited resources. Fig. 1 shows a snapshot of sorting a scattered collection of nine robots.

Our algorithm is based on asymmetric tree topologies, where we can extract a specific main path from the trees. We change this main path to reflect the relative order of the robots, and bring in more robots to join the main path. We introduce a set of atomic operations to make the changes, each triggered by different conditions, and these operations perform at all the possible locations in parallel. Finally all the robots join the main path with the correct order, and the main path forms a line with robots evenly distributed.

Our algorithm requires only that robots are able to measure the relative directions of their neighbors in their local coordinates, and that robots initially form a connected graph. The overall worst- case wall-clock time complexity is $O(n^2)$, travel distance for each robot is $O(n^2)$, and memory required for each robot is $O(1)$, where $n$ is the number of robots. The operations use constant message size and constant memory, and this algorithm is a practical solution for large populations of low-cost robots.

Previous work on sorting groups of robots is limited. The most relevant work is of Litus and Vaughan [1], which uses a Double Bracket Flow to build a dynamical system to model the robots’ positions. Evolving this system drives the robots to a sorted ordering. This is a compact and analytical solution with provable properties, but requires that the robots are initially placed along a line parallel to some given axis. Additionally, this model requires robots to sense over arbitrarily long distances, which is impractical in some applications.

McLurkin, Li, and Embree [2] solve the sorting problem with a discrete model. In this model each robot simply jumps to the midpoint of its predecessor and successor at each time step. The authors prove convergence with matrix iteration, but assume that a robot can communicate with its final predecessor and successor. These assumptions are weaker than global sensing, and the correctness of the procedure is proven mathematically, but the algorithm is not designed for wheel-based robots that cannot jump.

Krupke et. al. [3] invented wave sort based on parallel odd-even sort, which is able to arrange robots into a chain
and sort in \(O(n)\) time complexity, where \(n\) is the number of robots. Odd-even sort takes \(n\) passes of \(n/2\) swaps, and the wave provides a synchronization method to run \(n/2\) swaps in a pipeline. This algorithm is quite efficient with embedded collision avoidance, but requires global stages, point to point communication with handshake, mutual exclusion (a robot swapping with one neighbor cannot accept a request from another), and consensus (a pair of robots agrees on when to start and stop swapping). If new robots are added during sorting, the global stage needs to be explicitly reset to the very beginning.

Our previous work [4] by Zhou, Li, and McLurkin builds a symmetric stepping model. In addition to moving to the midpoint of the predecessor and successor, this algorithm allows the local minimum (or maximum) to navigate to the global minimum (or maximum) in a symmetric approach. However, the network may disconnect and cause failure with limited sensing range. Implementing geometry-based safety constraints could solve this problem in most cases and make this algorithm practical, but there still exist specially constructed counterexamples [5]. In addition, correctness of this algorithm is very difficult to prove.

To straighten the sorted line and avoid disconnecting the network during the algorithm, we are also inspired by papers related to chain formation [6] [7] [8] [9].

II. MODEL AND ASSUMPTIONS

We have a set of \(n\) robots. The robots form the vertices \(V\) of an undirected communication graph \(G = (V,E)\), in which two vertices \(u\) and \(v\) are connected by an edge in \(E\) iff \(u\) and \(v\) are close enough to communicate directly without utilizing other robots, i.e., \(E\) is the set of all robot-to-robot communication links. A path is a set of continuous connected edges in the graph, and the number of hops means the number of edges between two given robots along a path. We assume that the communication range of each robot \(v\) is an open disk or open ball with radius \(r_{\text{max}}\), and all other robots within the disk or ball can bidirectionally communicate with robot \(v\). We call all the robots within the communication range neighbors of robot \(v\). This assumption also results in a monotonic communication graph, in which two connected robots continue to be connected if one moves directly toward the other.

If the robots can measure only bearing without distance, moving along the angle bisector could be an approximation of moving to the midpoint [7].

Robots are distinguishable with unique serial numbers, where robot \(v\) has serial number \(\text{SERIAL}(v)\). Each robot \(v\) also has a label or value \(\text{VALUE}(v)\) by which we sort. We can either assume no two robots have the same value, or we can use the serial number to break ties, i.e., we always compare lexicographically the pair \((\text{VALUE}(\cdot), \text{SERIAL}(\cdot))\) whenever we compare two robots. All the robots can be totally ordered as \(\{v_1, \ldots, v_n\}\), i.e., \((\text{VALUE}(v_1), \text{SERIAL}(v_1)) < (\text{VALUE}(v_{i+1}), \text{SERIAL}(v_{i+1}))\) iff \(\text{VALUE}(v_i) < \text{VALUE}(v_{i+1})\) or \(\text{VALUE}(v_i) = \text{VALUE}(v_{i+1})\) and \(\text{SERIAL}(v_i) < \text{SERIAL}(v_{i+1})\). Each robot knows only its own value and serial number, but has no knowledge about the number of robots \(n\) or its relative order \(i\). The robot \(v_1\), which has the lowest value, is called the global minimum. The robot \(v_n\), which has the highest value, is called the global maximum.

The position of each robot \(v_i\) is described by \(p_i\). Now the overall task is to achieve a sorted, evenly spaced arrangement of robots between the global minimum and the global maximum, i.e., robot \(v_i\) must move to position \(p_i = p_1 + (i - 1) \times (p_n - p_1)/(n - 1)\), for \(i = 1, \ldots, n\).

Algorithm execution occurs in a series of rounds. Each robot has a local timer with a fixed interval, which triggers round updates. All the robots share the same fixed interval, although their round updates need not to be synchronized. These rounds greatly simplify analysis and are straightforward to implement in a physical system [10]. At the end of each round, every robot broadcasts a message to all of its neighbors, and also collects messages received from all the neighbors during this round. Each message contains a set of variables, including the sender’s value and unique serial number. The remaining variables will be defined later, but the total message has constant size. Each robot may move up to a distance \(d_{\text{step}}\) in each round.

III. ALGORITHM

Our goal is to form a sorted path among all the robots, while distributing robots evenly on a straight line. We accomplish this task with a set of atomic operations that are fully distributed and do not require mutually exclusive decisions between robots.

The procedure builds a physical data structure called the main path, a path in \(G\) from the global minimum to the global maximum. The main path is initially a shortest path in all those possible paths. We apply topological operations to modify the main path: insertion and deletion add and remove edges so that we can enlarge sorted segments and eliminate unsorted segments on the main path, and navigation changes local topology to move robots to their correct positions. In addition, we perform geometric operations: move-to-midpoint straightens the main path, and move-to-parent brings those robots not on the main path closer to the main path. These operations run in parallel on all the robots. When the algorithm finishes, the main path will contain all the robots sorted and evenly distributed along a straight line. We describe these operations in this section, and present our analysis in the next section.

In order to describe the algorithm and analysis more clearly, we list here all the assumptions and limitations:

- The communication network is initially connected and remains connected. We only recover from failures that do not violate this rule.
- If a robot fails, the robot is immediately and permanently removed from the set of all the robots.
- All the robots have a circular range with the same radius. Bidirectional communication is guaranteed if two robots are located within this radius; otherwise there is no direct communication.
- Values are totally ordered. Serial numbers are unique and also totally ordered.
A robot can follow another robot at the same speed and needs no time to accelerate or rotate.

- Rounds take the same amount of time for all the robots, although rounds do not have to be synchronized.
- There are no sensing errors, no collisions, and no communication interference. A robot does not block the communication between two other robots.

### A. Main Path Formation

![Fig. 2](image.png)

Fig. 2: The primary tree is drawn in solid arrows (both bold and thin) and the feedback tree is drawn in thin dashed arrows. An arrow from A to B means A selects B as its parent in either tree, while messages are relayed in the opposite direction. The main path is highlighted as bold solid arrows. Neighbor connections not used to form trees are drawn in thin dashed lines.

The main path always connects the global minimum and the global maximum. We construct the main path using two trees: the primary tree and the feedback tree. The primary tree is a spanning tree rooted at the global minimum; the feedback tree is a tree with the same set of edges but rooted at the global maximum (Fig.2).

The primary tree is represented by robots (as vertices) and parent selections (as directed edges from each child to its parent). The global minimum does not have a parent. Each of the other robots chooses a neighbor as its parent, and stores the parent’s serial number. The algorithm maintains appropriate parent selections so that all the robots are singly connected as a tree rooted at the global minimum.

We construct the primary tree as a distributed Breadth-First-Search spanning tree [10], but we disable the automatic refreshing to shorten paths between the root and the leaves. The current parent is preserved unless modified by one of the topological operations described later in this section, or under one of the following conditions:

- some robot with lower value than the smallest known value is discovered,
- the robot with the smallest known value is removed, or
- a robot loses connection to its parent.

If any of these conditions happens, the robot selects a parent from its neighbors which claims to have the fewest number of hops to the root (as a result of timeout mechanism in [10]).

There is one path in the primary tree that connects the global minimum and the global maximum. We use this path as the main path, and we let robots know whether they are on the main path by introducing a feedback mechanism. The feedback tree is another spanning tree overlaying the primary tree but rooted at the global maximum. We use the same method [10] to construct the feedback tree, but limit the parent selections to those edges selected by the primary tree, i.e., if robot \( u \) selects robot \( v \) as its parent in the feedback tree, robot \( u \) must be robot \( v \)'s parent or child in the primary tree. Automatic refreshing is enabled in the feedback tree so that the feedback tree changes in the same way as the primary tree.

The feedback tree provides feedback from the global maximum to the global minimum so that:

- a robot finds itself on the main path if its parent in the primary tree and its parent in the feedback tree are different, and
- the other robots know that they are not on the main path (they form the branches), because the parent in the primary tree and the parent in the feedback tree are the same (Fig.2).

Since the feedback tree merely provides feedback for the robots to detect whether or not they are on the main path, and is refreshed in every round, our topological changes operate only in the primary tree. In the remainder of this paper, when we talk about parent, child, and sibling, we refer to those in the primary tree unless otherwise noted.

### B. Geometric Operations (Robot Motion)

Robots have different geometric behaviors based on whether on the main path or not:

- The global minimum and the global maximum remain stationary.
- The other robots on the main path move to the midpoint between the parent in the primary tree and the parent in the feedback tree. This operation straightens the main path and distributes the robots evenly [10].
- The robots in the branches move toward their parents (the parents in both trees are the same). This operation brings these robots toward the main path.

These motions change the geometry of the network, and since scattered robots move closer to become neighbors, geometric changes may cause topological changes.

### C. Main Path Topological Operations

![Fig. 3](image.png)

Fig. 3: Main path operations.

There are two kinds of main path operations: insertion and deletion (Fig.3).
Insertion adds one robot with correct local order onto the main path (Fig.3(a)). If robot \( u \) is on the main path, and there exist neighbors \( p \) and \( s \) such that:
- \( p \) is \( u \)'s parent (\( p \) must be on the main path),
- \( s \) also selects \( p \) as its parent (\( s \) is \( u \)'s sibling in a branch), and
- \( p < s < u \),
then \( u \) selects \( s \) as its parent instead of its current parent \( p \). If there is more than one robot that can be selected as \( s \), robot \( u \) simply picks one.

Deletion removes one robot with incorrect local order from the main path (Fig.3(b)). If robot \( u \) is on the main path, and there exist neighbors \( p \) and \( g \) such that:
- \( p \) is \( u \)'s parent (\( p \) must be on the main path),
- \( g \) is \( p \)'s parent (\( g \) is \( u \)'s grandparent and must be on the main path), and
- \( g > p \) or \( p > u \),
then \( u \) selects \( g \) as its parent instead of its current parent \( p \). We need deletion because the main path is initialized regardless of order, and we fix the order by deleting locally disordered robots. Those robots deleted will finally get back onto the main path at appropriate positions with navigation (described below) followed by an insertion.

D. Branch Topological Operations

Robots in the branches also have operations to change branch topology. Thus they can move topologically and geometrically to appropriate positions to get inserted onto the main path. Navigation to minimum and navigation to maximum (Fig.4) allow branches to move along the main path and seek positions in which they can be inserted.

Navigation to minimum moves a branch robot toward the global minimum (Fig.4(a)). If robot \( u \) is in a branch, and there exist neighbors \( p \) and \( g \) such that:
- \( p \) is \( u \)'s parent and is on the main path,
- \( g \) is \( p \)'s parent (\( g \) is \( u \)'s grandparent and must be on the main path), and
- \( u < p \),
then \( u \) selects \( g \) as its parent instead of its current parent \( p \).

This operation changes branch topology to let a low-value robot get closer to its grandparent. If \( g < p < u \), robot \( u \) gets inserted onto the main path. Otherwise, this process repeats after \( u \) geometrically moves toward \( g \) to become a neighbor of \( g \)'s parent. This operation also eliminates all the branches connected to the global maximum, as they cannot get inserted since they do not have a sibling on the main path.

Navigation to maximum moves a branch robot toward the global maximum (Fig.4(b)). If robot \( u \) is in a branch, and there exist neighbors \( p \) and \( s \) such that:
- \( p \) is \( u \)'s parent and is on the main path,
- \( s \) is on the main path and \( s \) selects \( p \) as its parent (\( s \) is \( u \)'s sibling on the main path), and
- \( u > p \) and \( u > s \),
then \( u \) selects \( s \) as its parent instead of current its parent \( p \).

This operation changes branch topology to let a high-value robot move toward its sibling on the main path. Robot \( u \) continues to move toward the global maximum until its sibling on the main path has a higher value than itself. Finally we have another parent \( p' \) and sibling \( s' \) with \( p' < u < s' \), and sibling \( s' \) performs an insertion to add \( u \) onto the main path.

![Branch collapse](Fig.5.png)

Fig. 5: Branch collapse. \( m \) can be any robot on the main path (not necessarily \( p \)'s parent or ancestor), and \( p \) can be any number of hops away from the main path.

An optional branch collapse reduces the depth of branches to accelerate navigation (Fig.5). If robot \( u \) is in a branch, and there exist neighbors \( p \) and \( m \) such that:
- \( p \) is \( u \)'s parent and in a branch, and
- \( m \) is any robot on the main path,
then \( u \) selects \( m \) as its parent instead of its current parent \( p \). This operation avoids the whole branch moving back and forth while navigating, since there are no other topological operations for robots more than one hop away from the main path. If more than one robot can be selected as \( m \), \( u \) simply picks one.

E. Movement of Endpoint

![Unsortable](Fig.6(a).png)

(b) Sortable after the global maximum moved.

Fig. 6: (a) This configuration cannot be sorted unless we shorten the main path to gain at least one more connection. (b) Sortable after the global maximum moves (degenerate triangles shown with curved edges). Detailed sorting process is shown in Fig.8.

In some cases the robots on the main path are too sparse so no sorting operations can be made (Fig.6(a)). Robots can still form an unsorted evenly distributed line. In order to solve the sorting problem, we allow the global maximum to move. The
global maximum moves toward its parent when its grandparent is not a neighbor. At the same time, we apply the geometric operation move-to-midpoint to the other robots on the main path. Eventually all the robots on the main path will have a connection of at least two hops in either direction along the main path. These connections enable the robots to delete disordered robots from the main path.

F. Implementation

Algorithm 1 SORTING(u)

1: PRIMARYTREE(u)
2: FEEDBACKTREE(u)
3: if u.primary.parent ≠ u.feedback.parent then
4: u.status ← on the main path
5: for each robot v in u.neighbors do
6: INSERTION(u, v)
7: DELETION(u, v)
8: end for
9: if u.primary.parent ≠ ∅ and u.feedback.parent ≠ ∅ then
10: MOVE2MIDPOINT(u.primary.parent, u.feedback.parent)
11: end if
12: else
13: u.status ← in the branches
14: for each robot v in u.neighbors do
15: NAVIGATION(u, v)
16: COLLAPSE(u, v)
17: end for
18: MOVE2TOWARD(u.primary.parent)
19: end if

Each robot needs to enumerate all the messages received in its current round, decide whether on the main path or in the branches, see if any topological operation can be performed, and perform a geometric operation. A straightforward version of the main algorithm (Algorithm 1) requires processing the neighbor messages multiple times, in order to update the trees and perform the operations. This requires each robot to have $O(n)$ memory to store the list of its neighbors.

Operations described in Algorithm 1 are performed if any condition in Fig.3, Fig.4, or Fig.5 is satisfied. If multiple operations qualify to be performed in the same round, the robot simply chooses one.

A modified version of this algorithm with $O(1)$ memory for each robot can also be constructed, if each robot can process each message promptly and discard after receiving another constant number of messages. Detailed construction is left to the reader.

IV. ANALYSIS

In this section we show that this algorithm is safe, correct, efficient, and robust.

A. Safety

Safety is the most important issue for sorting robots. Once the connected network breaks and some robots are apart, the algorithm fails since no distributed algorithm works with a disconnected network. Our old symmetric algorithm [4] is unsafe: local extrema in that algorithm are very likely to tear the network apart, and we even provided an example to show that applying only move-to-midpoint could leave someone behind. But in our new asymmetric algorithm, we can prove that the procedure is intrinsically safe and does not disconnect the robots.

Theorem 1. All the topological operations are safe.

Proof. We begin with a theorem in graph theory [11], if an undirected graph has $n$ nodes and $n-1$ edges, and there is no cycle, the graph must be connected. We have $n$ robots with $n-1$ child-to-parent connections associated to them (the global minimum has no parent), assuming that communication is bidirectional, then we fall into this case. Therefore, if the graph breaks, there must have been a cycle.

Here we show that any combination of operations can never generate a cycle, even if some robots make competing decisions at the same time. We have only four topological operations (insertion, deletion, navigation, and collapse), and none can directly select a descendent as a parent. If multiple robots make decisions and perform operations on their own judgment, a robot being selected as a parent can change its parent selection at the same time. Deletion and navigation to minimum always make a robot select a parent with fewer number of hops to root than its current parent, therefore these operations cannot generate cycles. Insertion and navigation to maximum select a parent with more number of hops to root than current parent, but our algorithm guarantees that these operations do not generate cycles:

- Concurrent multiple insertions do not influence each other, since they can happen only between different parent-and-child pairs on the main path.
- Concurrent multiple navigations to maximum do not influence each other, since the topology of the main path does not change, and navigating robots do not select each other as parents since they are in branches.
- Concurrent insertion and navigation to maximum: insertion requires that the sibling on the main path has a greater value than the sibling in a branch, while navigation to maximum requires the opposite.

Theorem 2. All the geometric operations are safe.

Proof. There are only two kinds of geometric operations: move to the midpoint of two robots, and move toward one robot. Here we prove that neither of these geometric operations causes disconnection.

- Only robots on the main path move to the midpoint of two other robots, and the main path forms a chain. As proven by Degener [7], the distance between any parent and child on the main path cannot increase, therefore the main path cannot break.
- Only robots in the branches move toward another robot. If a robot chases another robot with equal or greater speed, the distance between the two robots cannot increase, therefore branches cannot break [9]. We can also employ recovery algorithms [12] to enhance safety.
- The global maximum moving toward its parent cannot move outside of the boundary set by [9], thus this movement cannot cause disconnection.
We denote by \(\{1,2,3\}\) the ascending relative order of three robots’ values. There are only six permutations of \(\{1,2,3\}\) (enumerated in the row headers), and three different types of triangles adjacent to the main path (enumerated in the column headers), so we have eighteen local triangle states. Each row contains one or two stable states.

B. Correctness

**Theorem 3.** There are no loops of operations causing deadlock.

**Proof.** All the topological operations (except branch collapse) are based on triangles adjacent to the main path, which have one or two edges on the main path. (They can be geometrically degenerate triangles.) We enumerate the only eighteen local triangle states in Fig.7 and show which operations are applied to change the topology. Unstable triangles become topologically stable after one or two operations, where stable means no more topological operations can be applied in this triangle to revert to a previous state.

We are also concerned about concurrent operations in nearby triangles. We can enumerate all the combinations:

- Concurrent insertions. Multiple robots can be inserted onto the main path at different positions, but only one can get inserted at the same position in one round. Once inserted, the robot cannot be deleted unless its parent or child is deleted.
- Concurrent deletions. A robot being deleted may also delete its parent at the same time. As we mentioned in Subsection IV-A, the main path is still connected and multi-hop branches may be generated. Robots in the branches navigate and get inserted in some other positions. They cannot be inserted at their previous positions.
- Concurrent navigations. Multiple branch robots navigate independently and do not change the topology of the main path. Navigating to minimum and navigating to maximum for the same robot have different conditions, so a robot cannot navigate back and forth between two positions.
- Insertion and deletion. A robot being deleted may insert another robot between itself and its parent. This generates a multi-hop branch and cannot be reversed.
- Deletion and navigation. A robot may navigate toward some robot that is being deleted. This generates a multi-hop branch and cannot be reversed.
- Navigation and insertion. A robot being inserted cannot navigate: navigation and insertion have mutually exclusive requirements.

**Fig. 7:** We denote by \(\{1,2,3\}\) the ascending relative order of three robots’ values. There are only six permutations of \(\{1,2,3\}\) (enumerated in the row headers), and three different types of triangles adjacent to the main path (enumerated in the column headers), so we have eighteen local triangle states. Each row contains one or two stable states.

**Fig. 8:** Robot 2 and robot 4 may perform operations sequentially or concurrently. All three cases result in the same correctly sorted main path.

Therefore, even if multiple operations take place at the same time, this algorithm does not generate any loop of topology changes. Fig.8 shows an example that different sequences of operations yield the same correct result.

**Theorem 4.** When the algorithm terminates with no more operations, all the robots are on the main path.

**Proof.** If the algorithm terminates with one or more robots in some branches, there must be at least one robot \(u\) having a parent on the main path. Geometric operations should have moved \(u\) toward its parent \(p\), in order to have its grandparent \(g\) (if exists) and main-path sibling \(s\) (if exists) as neighbors. If \(s\) exists and \(p < u < s\), \(s\) inserts \(u\) onto the main path. Otherwise, \(u\) must select either \(g\) (if \(g\) exists and \(u < p\)) or \(s\) (if \(u > p\) and \(u > s\)) as its parent. These comparisons are both inclusive and exclusive, therefore an operation must be performed immediately. This contradicts with the assumption that the algorithm has terminated.
Theorem 5. The main path gets sorted in ascending order.

Proof. If we do not allow the global maximum to move, this algorithm may terminate unsorted because there are no triangles on which to perform operations (Fig. 6(a)). Allowing the global maximum to move toward its parent compresses the main path, and finally all the robots on the main path can directly communicate with their grandparents and grandchildren. So we assume there always exist triangles with two edges on the main path at every possible position.

The relative order of robots’ values on the main path forms a sequence of natural numbers. This sequence starts from the global minimum, contains one or more increasing pairs and zero or more decreasing pairs, and ends at the global maximum. Decreasing pairs should be avoided, and once eliminated, the main path is sorted. Here we prove that the algorithm removes all decreasing pairs.

We present two metrics to measure the disorder:

• The number of decreasing pairs. For example, in the sequence \( \{1, 7, 5, 2, 3, 6, 4, 8\} \), three natural numbers \( \{5, 2, 4\} \) are smaller than their predecessors, and \( \{7, 5\}, \{5, 2\}, \{6, 4\} \) are the three decreasing pairs.

• The summation of decreasing pairs. In the same example, \( (7 - 5) + (5 - 2) + (6 - 4) = 7 \).

From any initial case, those two metrics are natural numbers. Either metric being zero results in the other also being zero, and this happens only when these natural numbers are reduced to zero or more decreasing pairs, and ends at the global maximum. Decreasing pairs should be avoided, and once eliminated, the main path is sorted. Here we prove that the algorithm removes all decreasing pairs.

We present two metrics to measure the disorder:

• The number of decreasing pairs. For example, in the sequence \( \{1, 7, 5, 2, 3, 6, 4, 8\} \), three natural numbers \( \{5, 2, 4\} \) are smaller than their predecessors, and \( \{7, 5\}, \{5, 2\}, \{6, 4\} \) are the three decreasing pairs.

• The summation of decreasing pairs. In the same example, \( (7 - 5) + (5 - 2) + (6 - 4) = 7 \).

From any initial case, those two metrics are natural numbers. Either metric being zero results in the other also being zero, and this happens only when these natural numbers are sorted in an ascending order. In our algorithm, neither of these metrics can increase. Insertion does not create or enlarge any decreasing pair, for we require \( p < u < s \). Navigation and collapse do not change the main path. For deletion, we have two situations:

• The natural number deleted is a local extremum (at the boundary between an increasing pair and a decreasing pair or vice versa). The summation of decreasing pairs must decrease, since a local extremum is removed. For the example above, removing \( \{2\} \) changes the decreasing pair \( \{5, 2\} \) into \( \{5, 3\} \) and reduces the summation of decreasing pairs.

• The natural number deleted is not a local extremum (must be between two decreasing pairs). The number of decreasing pairs must decrease. In the same example, removing \( \{5\} \) merges two decreasing pairs \( \{7, 5\} \) and \( \{5, 2\} \) into one \( \{7, 2\} \).

As we assume triangles adjacent to the main path always exist, there must be a deletion operation if a local extremum exists. Each deletion either reduces the number of decreasing pairs or reduces the summation of decreasing pairs. Since these metrics are natural numbers, they must reduce to zero. Therefore all the decreasing pairs will be removed and leave the main path ascending ordered.

Theorem 6. All the robots on the main path will form a line and distribute evenly.

Proof. Since each robot on the main path moves only toward the midpoint between its parent and child on the main path, the main path forms a chain, and the chain will be straightened and the robots will be evenly distributed along the chain as proven in [7] [9].

As a consequence, this algorithm sorts all robots regardless of initial conditions, provided that the initial communication network is connected.

C. Efficiency

We show that both the time spent and the distance traveled for each robot are bounded by \( O(n^2) \).

First, we assume that all the robots are either on the main path or only one hop away from the main path. For each robot \( u \) (except the global minimum and maximum), there are \( k \) ascending pairs formed by the other robots in which robot \( u \) falls into the range. Robot \( u \) may navigate and insert into one of these ascending pairs, but once inserted, robot \( u \) cannot be deleted until robot \( u \) becomes a local extremum. Once deleted, robot \( u \) no longer falls into the new ascending pair between its parent and its previous child. No operations can increase \( k \). Thus, each robot \( u \) can only get deleted and inserted at most \( k \) times. Since \( k \) can be proportional to \( n \) and each navigation travels at most \( n - 2 \) hops, the distance traveled for navigation is bounded by \( O(n^2) \).

Second, a robot \( u \) may change from one hop to two hops away from the main path, if its parent is deleted. Robot \( u \) travels a constant distance to get back to one hop from the main path, and this can happen at most \( n - 3 \) times. Thus, each robot travels \( O(n) \) total distance to get back within one hop from the main path.

Third, to straighten the main path, each robot moves a distance of up to \( O(n) \), for the distance between any two robots must be less than \( (n - 1)r_{max} \).

Fourth, multi-hop branch collapse takes at most \( n - 2 \) hops from any leaf to the main path, so each robot in the branches moves a distance of up to \( O(n) \) to get within one hop from the main path.

Therefore, the worst case distance complexity for each robot is \( O(n^2) \) if we decompose this algorithm into serialized stages. Movements are optimized more in this algorithm than in our previous work [4]. This algorithm actually runs in parallel and robots usually do not have to navigate back and forth, so we expect a robot to move a distance of \( O(n) \) in the average case (shown in Section V).

D. Robustness

Since the underlying tree-formation algorithm [10] provides a timeout mechanism, our sorting algorithm is robust to these swarm modifications:

• Adding a new global minimum or maximum. The new global minimum or maximum is recognized by neighbors, and the primary and feedback trees are updated.

• Adding general robots to the communication network. A new robot becomes a leaf of both trees.

• Removing some robots that does not disconnect the communication network. Trees are updated, and a new global minimum or maximum may be elected.
• Permanent failure of any rate if the failure can be treated as robot removal and not cause disconnection.
• Temporary failure or communication loss (the allowed rate can be calculated from [10]).

V. SIMULATION RESULTS

We have simulated this algorithm with our simulation software. Our simulation software supports up to 150 robots and updates all movements and communications every 20 milliseconds. The simulation software can provide precise sensor readings and tree structures, as well as model some physical effects, such as errors, collisions, obstructions, and communication delay.

![Area of Convex Hull versus time. Ten initial configurations](image)

Fig. 9: Area of the convex hull versus time. Ten initial configurations (shown in different colors) are simulated with both algorithms.

To analyze our procedure of sorting robots, we measure the area of the convex hull of the robot configuration, where convex hull is the smallest polygon covering all the robots [13]. For each vertex robot, all the other robots appear in a sector smaller than 180 degrees. Since we have only two kinds of movements, move-to-midpoint and move-to-parent, there is no movement for a vertex robot to move outside the convex hull. Thus, the area of the convex hull cannot increase. If the area of the convex hull becomes zero, all the robots must be in a line. Fig. 9 compares the new algorithm (always succeeds) and our previous algorithm [4] (often fails without range-based safety).

![Initial configuration and average sorting time](image)

(a) The asymmetric algorithm. All (b) The symmetric algorithm in our previous work. Converging to non-zero means disconnected and failed.

Fig. 10: Experiments suggest that the average time is proportional to the number of robots for a constant density.

We ran simulations with different numbers of robots, and recorded the time required to sort randomly generated initial configurations with similar density. The results suggest $O(n)$ average time complexity (Fig.10). Traces of robots in one of those experiments are shown in Fig.11.

VI. CONCLUSION

We presented a distributed algorithm to sort robots in Euclidean space. Robots that form a connected graph are sorted from arbitrary initial configurations, even if sensor range is limited and distance measurement is absent. This algorithm is intrinsically and provably safe, correct, and efficient, and is a good solution for sorting a large swarm of low-cost robots in motion. Our future work will be moving to non-flat spaces with obstacles and to efficiently handle collisions for real robots.

REFERENCES