2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

A Parallel Distributed Strategy for Arraying a Scattered Robot Swarm

Dominik Krupke[1], Michael Hemmer[1], James McLurkin[2], Yu Zhou[2], and Sándor P. Fekete[1]

***Abstract*— We consider the problem of organizing a scattered group of n robots in two-dimensional space. The communication graph of the swarm is connected, but there is no central authority for organizing it. We want to arrange them into a sorted and equally-spaced array between the robots with lowest and highest label, while maintaining a connected communication network.**

In this paper, we describe a distributed method to accomplish these goals, without using central control, while also keeping time, travel distance and communication cost at a minimum. We proceed in a number of stages (leader election, initial path construction, subtree contraction, geometric straightening, and distributed sorting), none of which requires a central authority, but still accomplishes best possible parallelization. The overall arraying is performed in $O(n)$ time, $O(n^2)$ individual messages, and $O(n)$ travel distance per robot. Implementation of the sorting and navigation use communication messages of fixed size, and are a practical solution for large populations of low-cost robots.

I. Introduction and Related Work

Consider a large company of n robots after deployment. They are scattered across a geometric region. While the swarm is still connected in terms of communication, it lacks central control; and while each of the robots carries a unique ID, none of them has information about the actual range of labels. How can we get the group into an organized arrangement: an equally spaced array between the positions of the robots with minimum and maximum label? (See Figure 1 for an example with 30 robots.) Not only does this demand dealing with the possibly complicated geometric arrangement in a distributed fashion; it also involves sorting them by label, which already requires $\Omega(n \log n)$ in a centralized setting. Furthermore, what are the achievable time until completion, required communication, and distance traveled?

Arranging robots in a specific order is a necessary routine in some applications on multi-robot systems. In addition to scenarios after deployment or perturbation by an uncontrollable event, robots may need to be ordered to go through a narrow passage, or to perform sequential procedures. For homogeneous robots, swapping tasks can solve this problem in some applications, but sorting is required if robots have different structures, are carrying different physical loads that cannot be swapped (and may need to arrive in order), or differ in some other intrinsic quantities, such as remaining battery level that cannot be transferred wirelessly [1].

[1]Sándor P. Fekete, Michael Hemmer, and Dominik Krupke are with the Department of Computer Science at TU Braunschweig, Braunschweig, Germany; s.fekete@tu-bs.de, mhsaar@gmail.com, d.krupke@tu-bs.de

[2]James McLurkin and Yu Zhou are with the Computer Science Department at Rice University, Houston, TX, USA; jmclurkin@rice.edu, yu.zhou@rice.edu

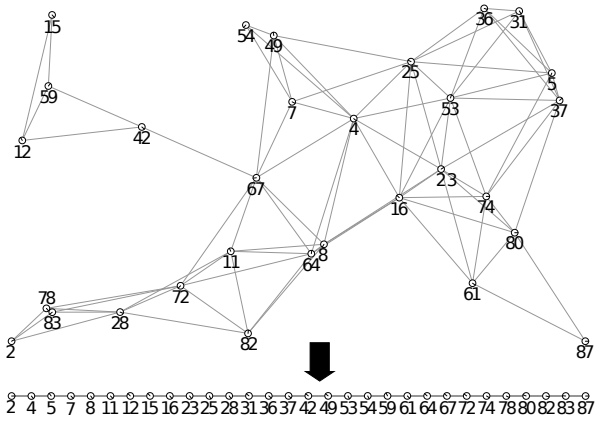

Fig. 1: An example of arraying a scattered swarm of robots. Thirty robots are initially randomly distributed in space, and eventually form an evenly spaced, sorted linear arrangement. A video of this process for 75 robots has been attached and can also be obtained from https://youtu.be/rqAALDwNDkQ.

In this paper, we describe an arraying algorithm to arrange robots in a sorted line. We show that this algorithm has linear completion time and travel distance, which are both optimal. The overall protocol is safe for any arbitrary initial connected robot configuration in Euclidean space. We make minimal assumptions on the robot's communication and computation requirements, so our approach is suitable for simple robots with limited resources.

Related Work

There has been a considerable amount of work on multi-robot navigation for labeled robots; e.g., see [2]–[8], who demonstrate a number of different challenges and approaches for dealing with various aspects of the inherent difficulties. However, there is limited previous work on sorting groups of robots in a way that combines the requirements of two-dimensional geometry, distributed computing, and sorting; moreover, the absence of obstacles makes the problem easier to solve, shifting the focus to achieving optimal efficiency. The most relevant work is of Litus and Vaughan [1], which uses a double bracket flow to build a dynamical system to model the robot's positions. Running this system will drive the positions to a sorted ordering. This is a compact, analytical solution and has provable properties, but it requires that the robots are initially placed among a line parallel to some given axis. Additionally, it requires long-range sensing and communication between robots.

A previous approach to our scenario modeled the system with a discrete model, see Zhou, Li, and McLurkin [9]. This model does not include the dynamic properties of the robots:

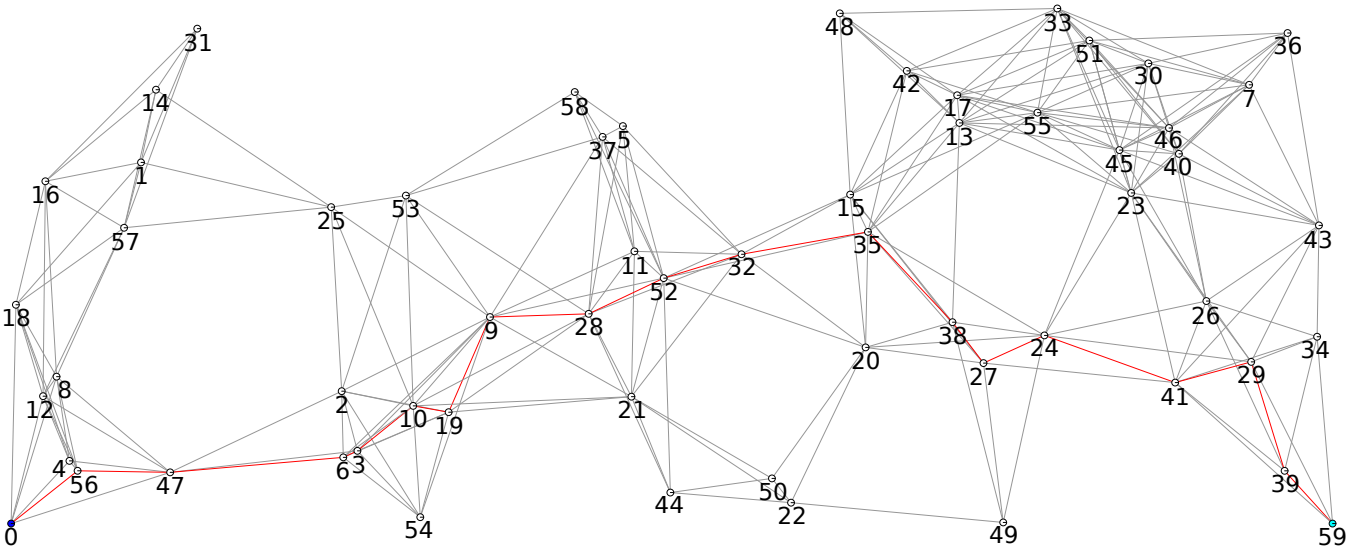

Fig. 2: An initial configuration of 60 robots. Edges of the graph G are indicated in light gray. The central path (Section IV-B) from r_{min} to r_{max} is indicated in red.

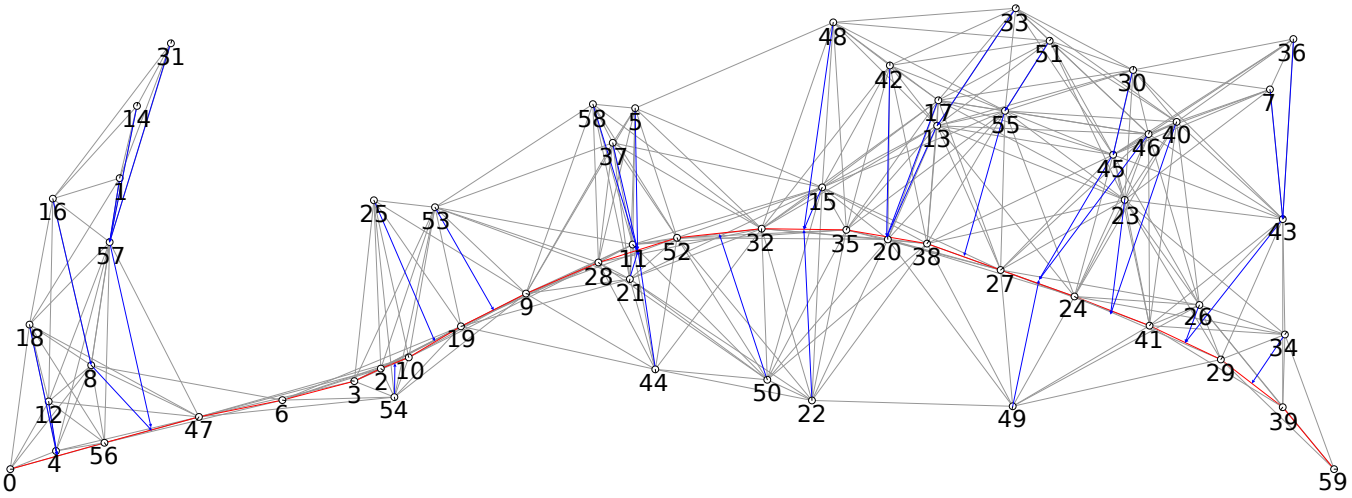

Fig. 3: A snapshot of the contraction phase of swarm already depicted in Fig. 2. Several robots have already moved onto the central path depicted in red. Robot paths for contraction are depicted in blue. It can also be observed that the contraction phase is intervened with the fourth phase as the central path has already straightened significantly.

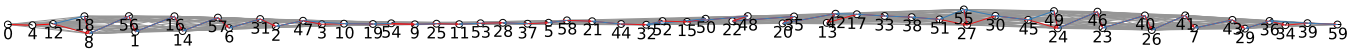

Fig. 4: A snapshot of the wave sort phase. After all the robots have been integrated into the central path during the contraction phase and as soon as the path is considered straight enough r_{min} initializes sorting waves that propagate through the chain of robots.

they simply jump to their destination at each time step. The authors prove convergence with matrix iteration, but also assume a robot can communicate to its final predecessor and successor. As it turns out, this method may get stuck if only local communication is available.

Some components of our approach make use of methods for related subproblems. One of them is the task of straightening a chain of robots in the plane, based on purely local methods; this amounts to our problem for the very special case of a communication graph that is a path, and robots are already sorted by labels. In a considerable sequence of papers, Meyer auf der Heide et al. [10]–[22] studied versions of the strategy GO-TO-THE-MIDDLE (GTM), in which each robot moves to the midpoint between its two immediate neighbors. Some of the underlying models are based on discrete rounds, with robots performing (possibly larger) discrete moves; however, Degener et al. [14] showed that in a setting with continuous motion and sensing, the variant MOVE-ON-BISECTOR produces a straight, evenly spaced chain in time bounded by $O(n)$; more recently, Brandes et al. [22] provided an analysis for continuous GTM and also established an upper bound of $O(n)$ on the distance traveled

by a robot, and thus, the overall time.

Another subproblem is actually sorting the robots along the chain. Here the idea is to exploit parallelism for achieving linear sorting time, thus beating the lower bound of $\Omega(n \log n)$ for comparison-based sorting. This has been studied in the context of a stationary array, with synchronized rounds: a parallel variant of bubblesort called *odd-even sort* [23] achieves a runtime of $\Theta(n)$; see Lakshmivarahan, Dhall, and Miller [24] for a comprehensive analysis. Note that our distributed scenario does not provide using synchronized rounds, but has to work in an asynchronous setting. To this end, we extend the previous work on odd-even sort to a new, asynchronous variant, which we call *wave sort*.

There is some relationship between the paper [25] by Zuluaga and Vaughan, who consider precedence rules for coordinated motion control of multiple robots with nonzero size and the heuristic component of our overall algorithm that aims at collision avoidance. However, Zuluaga and Vaughan give precedence based on energy already spent for motion, while we give precedence based on energy that still needs to be spent.

In addition to these explicit references, there are also several aspects which are dealt with implicitly, including local navigation, and connectivity. Given the limited amount of space, we refrain from providing a survey of these related aspects.

Contribution

This paper presents a distributed algorithm that arranges a set of robots to an uniformly spaced path (implemented as doubly linked list in which each robot knows its predecessor and successor) and sorts them based on some intrinsic property (label). In the end, the robots with minimal and maximal label are the beginning and end of this path, with both of them staying put throughout the protocol. Assuming that robots can communicate when within unit distance, the objective is achieved in total time $\Theta(n)$, and total travel distance $\Theta(n)$ per robot, which are both best possible assuming small robot diameter compared to the final path length. The message complexity is $O(n^2)$ after leader election. As a subroutine, we propose the distributed $\Theta(n)$ sorting algorithm "Wave Sort" that is based on Odd-Even-Sort [23], but works in an asynchronous manner. In addition, we give simulation results for an implementation with up to 130 robots, demonstrating that all components of our overall protocol are indeed linear.

II. Model and Assumptions

We are given a swarm R of n robots r_i, $i \in \{1,\ldots,n\}$, where each robot has a unique ID number $ID(r_i)$, the total set of these labels is unknown. We assume the ID of a robot to be expressable by a constant size bit field. In addition we are given a total order $<$ among them. The ID space need not be known in advance.

Communication is possible whenever two robots are sufficiently close, i.e., they are within communication range of each other. Thus, the robots form the vertices of an undirected communication graph $G = (V,E)$ with n nodes, in which two vertices r_i and r_j are connected by an edge in E, iff r_i and r_j are close enough in order to communicate directly without utilizing other robots.

Let robot r_j be a direct neighbor of robot r_i in G, then r_i can measure the relative position of r_j. We assume that all these measurements are accurate and timely, unless otherwise noted. Furthermore, we assume that each robot can accurately move in the direction of another robot, or towards the midpoint between two visible robots. We also assume that the time to travel a specific distance d is basically linear in d, as the time to reach maximum speed is small compared to travel time.

Now the overall task is to achieve a sorted, evenly spaced arrangement of robots between those with lowest and highest ID-number, i.e., robot r_i must move to position $p_1 + (i-1) \times (p_n - p_1)/(n-1)$, for $i = 1,\ldots,n$, where p_1 and p_n are the initial positions of robot r_1 and r_n, respectively.

Throughout this paper, we do not explicitly deal with message loss, as the use of a communication protocol with acknowledgements [26] makes our algorithm robust against dropped messages. The iteration number can be appended to avoid multiplication of messages due to lost acknowledgements. A robot will only move if it has received all necessary messages, thus if a robot starts to move before we received an acknowledgement from it, we know it has received the message. Overall, delays only slow down execution, but do not harm it.

Note that robot motion in the course of carrying out an algorithm may significantly change the scale of robot distances. Therefore, even every small positive dimensions of robots may cause *qualitative* behavior that is fundamentally different than that for point robots. As a consequence, our algorithms are designed for robots of positive size and with realistic motion, which is also shown in simulation. For the purposes of *quantified* theoretical evaluation, robot dimensions (and other practical parameters like message delays) are treated as small constants, so they are covered by O-notation.

III. Echo Waves

The echo algorithm of Chang [27] is a wave algorithm that is used multiple times in this paper. The initiator starts the algorithm by sending a message to all its neighbors. A robot that receives such a message for the first time broadcasts the message to all its neighbors except for the sender of the message, which is saved as predecessor. This can be implemented by a simple broadcast message with appended information for the predecessor to ignore this message. Only when the robot has also received the message from all its neighbors, it sends the message back (echo) to its predecessor. The algorithm terminates as soon as the initiator has received the message from all its neighbors.

One important aspect of this algorithm is that it spans a tree and each robot finishes before its predecessor. For equal messages delays this tree equals the minimal hop tree

towards the initiator. The algorithm terminates within $O(n)$ time steps and needs $O(n)$ broadcast messages.

IV. Our Algorithm

Our algorithm proceeds in a total of five phases. The first phase is the **leader election** phase, in which all robots identify the extremal ID numbers (Section IV-A). The second phase then establishes an **initial path** within G between the two extremal robots r_{min} and r_{max} (Section IV-B). We then **straighten** (Section IV-C) the central path. Integrating the remaining robots into the path is achieved by **contracting subtrees** (Section IV-D). In the overall protocol (and thus in our simulations), both processes run in parallel; we do present them as two different phases for better readability As soon as all robots are on the central path and as soon as the path is straight enough (Section IV-C) we move on to the fifth and last phase, the **wave sort**, which is a modification of odd-even-Sort, a synchronized, parallel variant of bubble sort. It requires $O(n)$ time and $O(n^2)$ messages (Section IV-E). Overall, the first four phases can be seen as building a robot array implemented as doubly linked list, while the last phase efficiently sorts the array without deforming it.

A. Leader Election

We determine the robots with smallest and largest ID using a simple leader election protocol as described in [28], based on the echo wave. Initially every robot claims to be the minimal/maximal robot and starts an echo wave containing its value. A robot does not propagate a wave if it already knows of a better value. Thus, only r_{min} and r_{max} are able to finish their waves. As soon as r_{max} finishes its wave, r_{max} broadcasts a message to inform all robots. As soon as r_{min} has received this message and has finished its echo wave, it initializes another echo wave. Each robot that finishes this wave can be sure its values are final and transits to the next phase. As the last robot to finish an echo wave is the initiator, r_{min} is the last robot that enters the next phase.

The phase needs at most $O(n^2)$ broadcast messages and at most $O(n)$ time steps. At the end of this phase, no robot has moved, but the robots r_{min} and r_{max} with smallest and largest ID have been identified.

B. Central Path

The goal of this phase is to establish an initial central path from r_{min} to r_{max} within G; see Figure 2. We could save work by choosing a path that already contains many robots; however, this amounts to solving the NP-hard problem of a maximum-length path in G, which may still not contain all robots. We sidestep these difficulties by pulling remaining robots to the initial central path, as described further down. More critical is the property of the path to be free from self-intersections, as these may lead to irresolvable collisions and blocking during straightening, even for arbitrarily small robot sizes. We prove that this can be avoided by choosing a shortest path from r_{min} to r_{max} in G, using the squared Euclidean distance as edge weight.

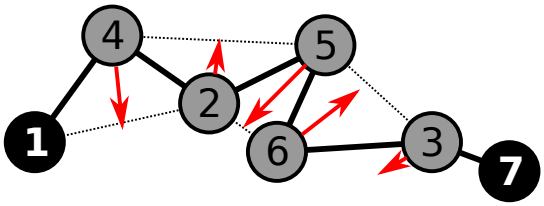

Fig. 5: Straightening using the continuous Go-To-The-Middle (GTM) Method. Not only does it straighten the path, it also makes it evenly spaced, as can be seen by r_3. In a continuous process the path gets straight.

Theorem IV.1 *The path from the root to a leaf in the routing tree $\mathcal{T}^R$ of a Unit-Disc-Graph using the squared Euclidean distances as edge weights is intersection free.*

Proof: Assume there is an intersection of the path's edges (a,b) and (c,d) with $\text{dist}(a) < \text{dist}(b) < \text{dist}(c) < \text{dist}(d)$, where dist refers to the distance to the root. Since (a,b) and (c,d) intersect the robots $\{a,b,c,d\}$ form a convex quadrilateral. There is at least one corner of a robot $x \in \{a,b,c,d\}$ with an angle $\geq 90°$, which implies that the diagonal (either (a,b) or (c,d)) formed by its two adjacent robots is longer than the two quadrilateral edges at x. Hence, this diagonal can not be part of the routing tree, a contradiction. ■

Note that this proof in fact only requires edge weights that reflect the order induced by their Euclidean length.

It is not difficult to see that the path remains crossing free throughout the rest of the algorithm. In order to compute the path we first compute a routing tree $\mathcal{T}^R$ rooted at r_{min} using the Chandy-Misra-Algorithm [29] that can be implemented with $O(n)$ time and $O(n \cdot e)$ individual messages for a graph with e edges, making use of a synchronizer [30]. Based on broadcast messages (and assuming constant effort for each), the total message complexity is $O(n^2)$. After $\mathcal{T}^R$ is established, r_{max} sends a message to r_{min} on $\mathcal{T}^R$. Each robot that receives this message forwards it to its parent in $\mathcal{T}^R$ and becomes part of the initial path. As soon as the message reaches r_{min}, it starts an echo wave on G, telling the remaining robots that they are not on the path. Robots that have finished that wave go over to the next phase. As before, r_{min} is the last robot that enters the next phase.

At the end of this phase, all robots know if they are on the initial path and if so its predecessor and successor, that is, the robots on the path emulate a doubly linked list. The phase has a time complexity of $O(n)$ and a message complexity of $O(n^2)$. There is no motion, yet.

C. Straightening a Path

All robots that are on the central path straighten the path by following the continuous GTM method, for which it is known that the chain of robots converges to a straight line (up to inaccuracy of measurements) [10]–[22]. The rule is simple: every robot moves towards the midpoint of the segment between its two neighbors n_l and n_r in the doubly linked list; see Figure 5. In addition, robots on the path are

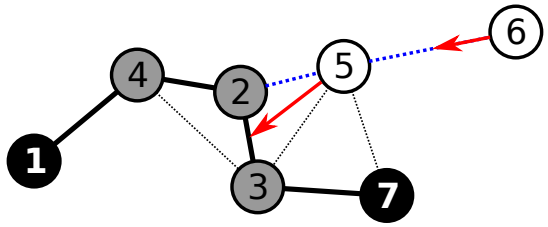

Fig. 6: Example for contraction. Extremal robots are shown in black, central path robots in grey with thick edges, and contraction tree in doted blue. Robot r_6 is not adjacent to the central path and thus moves towards its predecessor. Robot r_5 is adjacent to the path and moves to the closest midpoint of a central path edge to get integrated.

not allowed to cross other path edges in order to ensure that the path remains intersection free.

As mentioned before, this phase is only described on its own for clearer exposition; in the overall algorithm, it runs parallel to contraction (Section IV-D), as well as sorting (Section IV-D). During contraction, the process evens out distances among consecutive pairs of robots, making space for new incoming robots that are integrated into the path. During sorting, this process ensures that robots continue to have sufficient space.

The overall movement in GTM of a robot has been proven to be in $O(n)$ [22]. Note that our proof of total travel distance $O(n)$ for each robot considers a sequence of phases. It is plausible that the distance does not get worse if phases run in a condensed, parallel manner.

D. Contracting Subtrees

This phase integrates remaining robots into the central path; see Figure 6. The idea is that robots that are close to the central path move towards the midpoint of two consecutive robots and get integrated into the path as soon as they have reached that position. At the same time, these robots are the roots of contracting subtrees that organize robots that are further away; see Figure 3. The continued straightening (Section IV-C) ensures that there is sufficient space between consecutive robots. At the end of this phase, there is a doubly linked list that contains all robots with r_{min} and r_{max} at its ends.

a) Contraction Tree: The contraction tree $\mathscr{T}^C$ is built by inducing another routing tree rooted at r_{min}. However, edges on the central path get weight zero, while all other edges in G get their Euclidean length. Thus, we obtain several routing trees that are attached to the central path.

b) Contraction: The following protocol ensures that robots that are further away move first, so parents do not lose connectivity to their children. In order to start a motion, r_{min} induces an echo wave on $\mathscr{T}^C$. However, only robots that have finished this wave are allowed to move. By delaying the finalization of the wave, a child can keep its parent static until it is ready to move. In general, children move towards their parent, while child robots attached to the central path move towards the midpoint of the closest path edge. Contractive motion can be parallelized with straightening the path. Time and travel distance per robot are linear.

We remark that due to contraction, the robots density grows. Thus, in extreme cases a child robot may get stuck, which could cause a loss of connection to its parent. This can be prevented by messaging the parent. However, we do not consider this for the quantified theoretical analysis.

c) Integration: Each path edge is owned by the incident robot closer to r_{min}. If an exterior robot r_{ext}, i.e., a robot that is not yet integrated into the central path yet, is close enough to the midpoint of an edge, the owner r_{own} of the edge can send an *offer*-message containing the ID of itself and the ID of its successor r_{suc} on the central path. Usually, r_{ext} acknowledges this with an *accept*-message to r_{own} and r_{suc}, in which case r_{ext} is integrated into the central path between r_{own} and r_{suc}. However, in rare cases r_{ext} may already have received another offer, in which case r_{ext} replies with a *reject*-message. Assuming that a robot receives at most $O(n)$ offers, the message complexity is $O(n^2)$.

d) Termination: For a cleaner separation of the sorting phase, we enforce all robots to be integrated before we start sorting. The contraction phase terminates as soon as all robots have been integrated into the central path, which is checked as follows. As soon as r_{min} has no child, it periodically sends an echo wave. However, the wave is restricted to the central path and only robots without any children are allowed to forward the wave. Therefore, all robots are integrated as soon as r_{max} sends the echo.

In addition, the wave is used to check that the path is straight enough, i.e., the wave is only sent via robots that are close to the middle of its two path neighbors. This is stable and ensures that robots have sufficient space to move during the next phase.

In $O(n)$ contraction and straightening time, only $O(n)$ waves are initiated by r_{min}. The periodically sent echo waves can be seen as one echo wave that is refreshed until it succeeds. The time is in $O(n)$ consisting of $O(n)$ until the succeeding wave is initiated and its termination time of $O(n)$. At most $O(n^2)$ messages are sent.

E. Wave Sort

As soon as the robots are arrayed on a sufficiently straight line, we enter the actual sorting phase. The idea is to use a variant of odd-even sort [23], [24], a parallel version of bubble sort in which alternately robots of odd and even pairs compare their labels and swap their positions if necessary. With global control, odd-even sort takes $O(n)$ time by carrying out at most $n-1$ parallel rounds, each in time $O(1)$. However, in our distributed setting with no global control, implementing global synchronization for each odd (even) round would take $O(n)$ time, increasing the overall complexity to $O(n^2)$.

Instead, our new *Wave Sort* implements each odd (even) round as a wave that is initialized at r_{min}. If an odd (even) wave reaches and odd (even) pair, they first propagate the signal and then start to swap if necessary. Thus, the minimal frequency of waves is essentially bounded by the time

required to swap two consecutive robots which is $O(1)$. For large n, one can literally see several waves that propagate through the chain of robots—see Figure 4. Overall, the algorithm requires $O(n)$ time. In particular, we require at most $n-3$ waves.

Robots r_{min} and r_{max} are already correctly placed. Consequently, they will not require swaps; r_{min} initializes waves (Algorithm 1), while r_{max} absorbs waves (Algorithm 3). Both algorithms are stated for completeness. In the following we present Algorithm 2 for all other robots. The predecessor in the doubly linked list is denoted by n_l and the successor by n_r.

Algorithm 1 Wave Sort: Min-Robot

```
1: m ← 'Master'
2: while not sorted do
3:     Wait for READY[] from n_r
4:     Send INIT[m, ID] to n_r
5:     Wait for RET[r]
6:     n_r ← min{n_r, r}
7:     m ← m̄
8: end while
9: Sorted!
```

Algorithm 2 Wave Sort: Non-extremal-Robots

```
1: while not sorted do
2:     Wait for n_l in range; Send READY[] to n_l
3:     Wait for INIT[m, l] from n_l
4:     if m = 'Master' then
5:         Send RET[min{ID, n_r}] to n_l
6:         Wait for READY[] from n_r
7:         Send INIT[m̄, l] to n_r
8:         Wait for RET[r] from n_r
9:         if n_r < ID then
10:            n_l ← n_r; n_r ← r
11:            Swap with n_r, sidestep right
12:        else
13:            n_l ← l
14:        end if
15:    else
16:        Wait for READY[] from n_r
17:        Send INIT[m̄, max{ID, n_l}] to n_r
18:        Wait for RET[r] from n_r
19:        Send RET[r] to n_l
20:        if n_l > ID then
21:            n_r ← n_l; n_l ← l
22:            Swap with n_l, sidestep right
23:        else
24:            n_r ← r
25:        end if
26:    end if
27: end while
```

1) Algorithm 2: The message exchange ensures that all robots know their future left and right neighbors, so that they can update the pointers of the doubly linked list before starting the actual swap (if necessary). In general, all incoming messages are stored in a buffer; i.e., if a robot *WAITS* for a message, it checks this buffer until it contains the message (which may already be the case). On success the message is taken and removed from the buffer.

Algorithm 3 Wave Sort: Max-Robot

```
1: while not sorted do
2:     Wait for n_l in range; Send READY[] to n_l
3:     Wait for INIT[m, l] from n_l
4:     Send RET[ID] to n_l
5:     n_l ← max{n_l, l}
6: end while
```

Consider the bottom of Figure 7 depicting a master-slave pair and its two neighboring pairs. The robots of these pairs may also swap. In the left pair, the robot that is going to end up at Position 1 must know $min(r_c, r_d)$, which is going to be its right neighbor. Similarly, the right pair must know $max(r_c, r_d)$. At the same time the robot ending up at Position 2 must know $max(r_a, r_b)$ and the robot that will be at Position 3 must know $min(r_e, r_f)$. After this information is exchanged the doubly linked list can be updated and (if necessary) the robots also change their physical positions (Algorithm 2:Line 11 and Line 22).

2) Example: A detailed example is given in Figure 7. In the top row, robot r_2 and r_3 are already paired. The message exchange of robot r_7 in detail is as follows. Robot r_7 starts its loop by sending a *READY*-message (A2:L2) to its left neighbor r_3. It then waits for a *INIT*-message (A2:L3) from r_3, which turns r_7 into a master for this round and also indicates that the left pair will end the round with r_3 as the robot at Position 1. r_7 then answers with a *RET*-message (A2:L5) indicating the future right neighbor for r_3, namely r_6. r_7 then waits for a *READY*-message (A2:L6) from r_6 and turns it into a slave by an *INIT*-message (A2:L7), which also contains the future left neighbor of this pair, namely r_3. It then waits for r_6 to finish its handshake with the right pair and finally receives a *RET*-message (A2:L8), which contains the future right neighbor for r_7, which is r_4. As master and slave now know their future neighbors, they update their pointers (A2:L10 and L21) and start the actual swap.

The important part of this protocol is that the master of the next pair always immediately answers with a *RET*-message (A2:L8), as soon as it has received its *INIT*-message (A2:L3). This *RET*-message already contains the future right neighbor of the current pair. This information is sufficient for the current pair to update its pointers and to start the swap while the wave continues to propagate to the right.

V. Simulation

We validated our approach by conducting simulations with robot swarms of various size. As we consider continuous sensing and motion, we assume that sensing and motion errors even out. Robots are simulated as disks, i.e., they may collide. The used parameters are inspired by those of

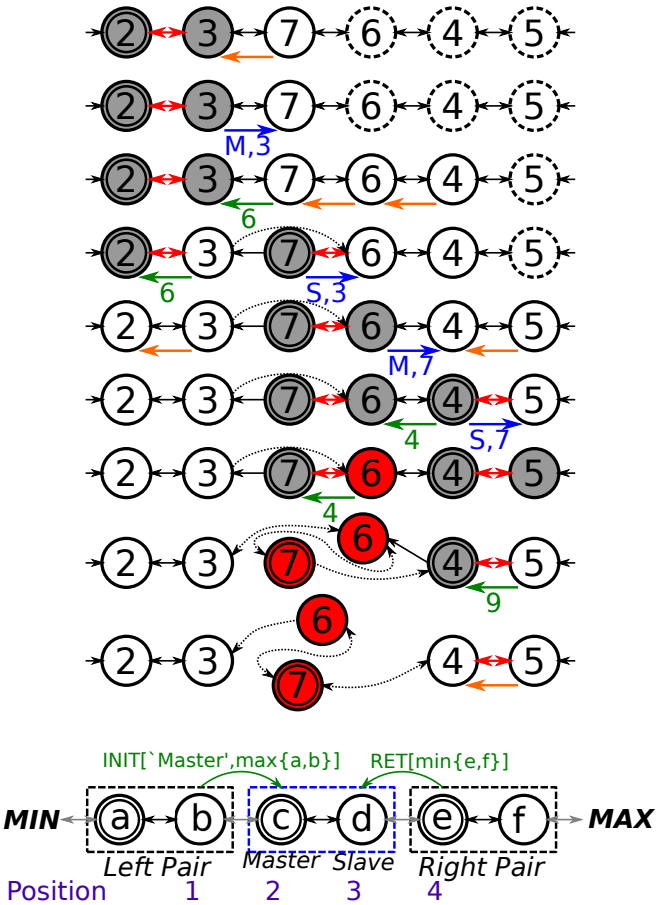

Fig. 7: Bottom: Sketch of a master-slave pair with its two neighbors. Top: A concrete example of a wave. Robots of central pair (r_7, r_6) swap, while left and right pair are not required to swap. Dotted robots are still in the previous wave. Gray robots are in the current wave. Red robots swap. Dotted pointers are already updated. Messages: *READY*—orange; *INIT*—blue; *RET*—green;

the actual r-one robots of Rice University [31]: robot radius is $0.05m$, communication range is $4.5m$, maximal velocity is $1m/s$ and maximal acceleration is $1.8m/s^2$. A robot can perform around 1.6 360° rotations per second, which are necessary to change the direction, as the robots can only drive for- and backwards. Robots are simulated at $60hz$ and messages are received within the next time step, i.e., after $1/60s$.

The experiments where made for n=15,...,130 robots. They were randomly placed into a quadrilateral of length 0.4*n and height 12. The minimal robot was placed at the lower left corner and the maximal at the lower right corner; see Figures 2-4 for an overall illustration. For collision avoidance, we used a heuristic "retreat from neighbors closer to their goal". This heuristic is self-explanatory because adjacent robots tend to aim for the same geometric positions. The comparison is made by exchanging the goal distance. The message complexity for this is not considered by us.

Clearly, the running time is linear.

It is also evident that the time for straightening the chain (the subject of numerous papers dealing with GTM) is almost negligible, i.e., $O(n)$ with a small constant.

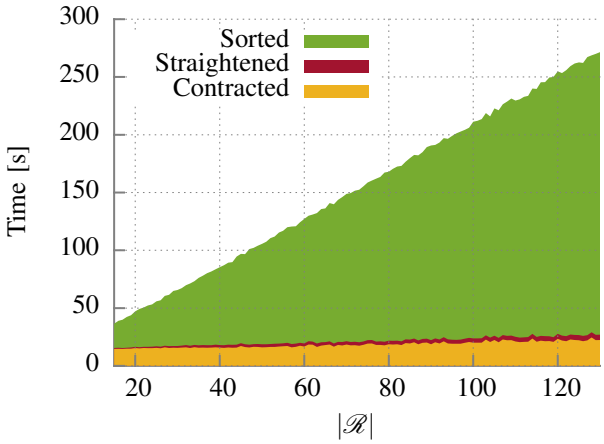

Fig. 8: Simulation runs for $|\mathscr{R}| = 15, \ldots, 130$ with 64 runs each; time includes deadlock prevention during collision-avoiding motion control. Clearly, all program components have linear runtime. The times are the time spans from the beginning until the swarm detected itself to be sorted, straight, or contracted.

VI. Conclusion

We presented a distributed algorithm to sort robots in Euclidean space, in overall time that is linear time. Robots get sorted from arbitrary initial configuration, even if sensor range is limited and there is no central control. Our underlying assumption is that configurations are dense enough for a connected communication graph, but not too dense for feasible arrangements (i.e., available space along the final path is sufficient for accomodating all robots) or for local collision avoidance.

There are many exciting new challenges that lie ahead. The next step is to combine our approach with dense settings in which density along the terminal path is a problem, and multi-robot collision avoidance comes into play. We plan to resolve these by moving global minimum and maximum apart to make the problem feasible. As this generalizes the subproblem of straightening a chain of robots by a local strategy like continuous Go-To-The-Middle or Move-On-Bisector, the mathematical analysis becomes more involved, even if the overall strategy displays benign behavior toward each other to make problem solvable.

At this point, our method is no yet able to recover from failures. There is hope of making this approach fully self-stabilizing.

References

[1] Y. Litus and R. Vaughan, "Fall in! sorting a group of robots with a continuous controller," in *Proc. of Canadian Conference on Computer and Robot Vision (CRV)*, May 2010, pp. 269–276.

[2] P. Svestka and M. H. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125–152, 1998.

[3] G. Sanchez and J.-C. Latombe, "Using a PRM planner to compare centralized and decoupled planning for multi-robot systems," in *Proc. of Int. Conf. on Robotics and Automation (ICRA)*, vol. 2. IEEE, 2002, pp. 2112–2119.

[4] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *Proc. of Int. Conf. on Intelligent Robots and Systems (IORS)*. IEEE, 2005, pp. 430–435.

[5] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. of Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 3260–3267.

[6] G. Wagner, M. Kang, and H. Choset, "Probabilistic path planning for multiple robots with subdimensional expansion," in *Proc. of Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 2886–2892.

[7] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 591–607.

[8] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artif. Intell.*, vol. 219, pp. 1–24, 2015.

[9] Y. Zhou, H. Li, and J. McLurkin, "Physical bubblesort," 2014, poster presented at International Symposium on Distributed Autonomous Robotic Systems (DARS).

[10] M. Dynia, J. Kutyłowski, P. Lorek, and F. M. auf der Heide, "Maintaining communication between an explorer and a base station," in *Biologically Inspired Cooperative Computing*. Springer, 2006, pp. 137–146.

[11] M. Dynia, J. Kutylowski, F. Meyer auf der Heide, and J. Schrieb, "Local strategies for maintaining a chain of relay stations between an explorer and a base station," in *Proc. of the 19th annual ACM symposium on Parallel Algorithms and Architectures*. ACM, 2007, pp. 260–269.

[12] F. M. auf der Heide and B. Schneider, "Local strategies for connecting stations by small robotic networks," in *Biologically-Inspired Collaborative Computing*. Springer, 2008, pp. 95–104.

[13] J. Kutylowski and F. Meyer auf der Heide, "Optimal strategies for maintaining a chain of relays between an explorer and a base camp," *Theor. Comput. Sci.*, vol. 410, no. 36, pp. 3391–3405, 2009.

[14] B. Degener, B. Kempkes, P. Kling, and F. M. auf der Heide, "A continuous, local strategy for constructing a short chain of mobile robots," in *Structural Information and Communication Complexity*. Springer, 2010, pp. 168–182.

[15] B. Degener, B. Kempkes, and F. Meyer auf der Heide, "Building simple formations in large societies of tiny mobile robots," *Procedia CS*, vol. 7, pp. 153–155, 2011.

[16] B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer, "A tight runtime bound for synchronous gathering of autonomous robots with limited visibility," in *Proc. of the 23rd annual ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 2011, pp. 139–148.

[17] P. Brandes, B. Degener, B. Kempkes, and F. M. auf der Heide, "Energy-efficient strategies for building short chains of mobile robots locally," in *Structural Information and Communication Complexity*. Springer, 2011, pp. 138–149.

[18] A. Cord-Landwehr, B. Degener, M. Fischer, M. Hüllmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Märtens, F. M. auf der Heide *et al.*, "A new approach for analyzing convergence algorithms for mobile robots," in *Automata, Languages and Programming*. Springer, 2011, pp. 650–661.

[19] B. Kempkes and F. M. auf der Heide, "Local, self-organizing strategies for robotic formation problems," in *Algorithms for Sensor Systems*. Springer, 2012, pp. 4–12.

[20] ——, "Continuous local strategies for robotic formation problems," in *Experimental Algorithms*. Springer, 2012, pp. 9–17.

[21] B. Kempkes, P. Kling, and F. Meyer auf der Heide, "Optimal and competitive runtime bounds for continuous, local gathering of mobile robots," in *Proc. of the 24th annual ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 2012, pp. 18–26.

[22] P. Brandes, B. Degener, B. Kempkes, and F. Meyer auf der Heide, "Energy-efficient strategies for building short chains of mobile robots locally," *Theor. Comput. Sci.*, vol. 509, pp. 97–112, 2013.

[23] A. N. Habermann, "Parallel neighbor-sort (or the glory of the induction principle)," *CMU Computer Science Report (available as Technical report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Port Royal Rd Sprigfield VA 22151)*, 1972.

[24] S. Lakshmivarahan, S. K. Dhall, and L. L. Miller, "Parallel sorting algorithms," *Advances in Computers*, vol. 23, pp. 295–354, 1984.

[25] M. Zuluaga and R. Vaughan, "Reducing spatial interference in robot teams by local-investment aggression," in *Proc. of Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 2798–2805.

[26] A. Leon-Garcia and I. Widjaja, *Communication Networks*, 2nd ed. New York, NY, USA: McGraw-Hill, Inc., 2004, ch. 5.

[27] E. J. Chang, "Echo algorithms: Depth parallel operations on general graphs," *IEEE Trans. Software Eng.*, vol. 8, no. 4, pp. 391–401, 1982.

[28] W. Fokkink, *Distributed Algorithms: An Intuitive Approach*. MIT Press, 2013, ch. 9, p. 79.

[29] K. M. Chandy and J. Misra, "Distributed computation on graphs: Shortest path algorithms," *Communications of the ACM*, vol. 25, no. 11, pp. 833–837, 1982.

[30] K. Lakshmanan, K. Thulasiraman, and M. Comeau, "An efficient distributed protocol for finding shortest paths in networks with negative weights," *IEEE Transactions on Software Engineering*, vol. 15, no. 5, pp. 639–644, 1989.

[31] J. McLurkin, A. J. Lynch, S. Rixner, T. W. Barr, A. Chou, K. Foster, and S. Bilstein, "A low-cost multi-robot system for research, teaching, and outreach," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 597–609.