Applied Intelligence (2020) 50:1316–1326
https://doi.org/10.1007/s10489-019-01602-5

Collision avoiding decentralized sorting of robotic swarm

Utkarsh Kumar[1] · **Adrish Banerjee**[1] · **Rahul Kala**[1]

Published online: 13 January 2020

Abstract

Sorting the swarm of robots is required when the robots are carrying different loads and they can not simply swap the loads. In 2016, Zhou et al. presented an interesting algorithm to sort a swarm of robots, wherein the authors made a main tree and a feedback tree to assign a topology to the robots, based on which the robots moved while arranging themselves in a sorted order. While the approach was very interesting and the results were critically analyzed by the authors, we see a critical problem that the approach did not account for collisions because of which the results can be very different. In this paper, we extend the work of Zhou et al. by enabling the robots to avoid collision by using a geometric approach called as "follow the gap" method. Together both the algorithms allow robot swarm to sort themselves in a straight line while avoiding collision simultaneously.

Keywords Swarm robotics · Obstacle avoidance · Sorting

1 Introduction

Swarm robotics is the study of coordination among a large group of simple and low cost robots [1]. The coordination is achieved by using some local behavior of individual robots. Swarms robotics is highly influenced by nature. In nature we see several insects form a swarm and they collectively perform more complicated tasks which they are not capable of performing individually, like ant or bee colonies, fish schools etc [2]. Swarm robotics is also influenced by the same idea. Instead of using expensive robots we will use several low cost robots which will come together to carry out a specific task. Swarm behavior is highly influenced with swarm intelligence and centralized or decentralized control over them for their coordination.

Research supported by the Science and Engineering Research Board, Department of Science and Technology, Government of India through Research Grant ECR/2015/000406.

✉ Utkarsh Kumar
utkarsh.dpsvns@gmail.com

Adrish Banerjee
adrish.banerjee24@gmail.com

Rahul Kala
rkala001@gmail.com

[1] Indian Institute of Information Technology, Allahabad, India

Following are few characteristics of swarm robotics [3]:

1. Robots in the swarm are autonomous.
2. Robots can act to change the environment.
3. Robots cooperate to perform a task.
4. Robots do not have global knowledge.
5. Generally the robots are homogeneous.
6. The system is scalable. [1]

Due to low cost and high effectiveness, swarm robots have various applications in defense, olfaction detection, smart cropping, mapping of environment, space exploration etc.

Since there are various applications of swarm of robots, they need to cooperate to perform the task. To cooperate they must form some kind of ordered or unordered pattern. Since the robots are very simple they are equipped with very simple resources that is they have low memory, they have short lifespan (battery). To utilize robots in the most efficient manner, sometimes we need to sort the robots based on some specific attribute(s). The attribute(s) may be the amount of battery left in robots, number of wheels of robots working correctly, etc.

Many robotic tasks require robots to be present in a desired order. For instance, a group of robots boarding a vehicle may need to do so in a particular order so as to utilize the space in the vehicle more efficiently. Similarly, there can be a certain preference when deploying or ejecting robots from that vehicle. A priority queue of agents may be required when the agents need to line up for refuelling or service.

There are numerous applications wherein sorting and orderings are important. Consider the problem of prediction of protein complex as done by Lei et al. [4] in protein–protein interaction. The authors first make a protein–protein interaction network as a dynamic graph to capture the temporal information. The core of such graphs act as a virtual heat centre to attract the artificial moths, which approach the heat source in a spiral motion. This makes the protein complex. The salps are another swarm agents that show interesting swarm behaviors. Imitating their motion behavior, Faris et al. [5] designed an algorithm for feature selection for machine learning algorithms.

Our objective is to organize the swarm robots in sorted and equally spaced sequence (from lowest to highest labels). While doing so we want to maintain the connected communication network among the robots.

Sorting is important because of the following reasons:

1. In case of homogeneous robots we can obtain sorting behavior by simply swapping the tasks, however swapping tasks requires agreement (consensus) among the robots and transferring data is expensive.
2. Sometimes robots might be carrying different loads and because of the structural difference loads can not be swapped.
3. When robots differ in intrinsic quantities (e.g. battery level) sorting is required as these quantities can not be swapped.

Zhou et al. [6] solved the problem of sorting of robots in a swarm in a decentralized manner. They proposed a graph based algorithm with topological and geometrical operations to sort the swarm of robots. The algorithm takes $O(n^2)$ time and $O(n^2)$ distance to sort the robots. Yet the algorithm is scalable and only requires knowledge of the relative direction of neighbours with respect to their local coordinates. However the authors do not account for collisions between the robots. This creates a severe problem in the analysis and results presented by the authors. Collision accounts for a major share of computational time and travel time in any multi-robot system. A deliberative motion planning algorithm spends most of the time in collision checking of the prospective states, which are used to create a solution. In reactive navigation of robots, mutual collision between robots restricts robots from moving at their top speeds, forces a robot to take a sub-optimal path so as to avoid collision with some other robot, makes a robot wait for other robot to create way, introduces the dilemma of cooperation and competition between the robots. Unlike humans, robots have difficulty handling dynamic obstacles. A human may use common sense to perceive and overcome a deadlock, however very few robots can easily cause deadlocks to each other. The underlying algorithm only sorts the robot in a straight line, but introduction of collision avoidance makes it more realistic.

The paper is organized as follows: Section 2 discusses the related work done so far, Section 3 describes the sorting algorithm of Zhou et al. [6], Section 4 describes collision avoidance technique Follow the Gap Method [7], Section 5 discusses the experiment performed and results, Section 6 compares the presented technique along with key discussions about the presented algorithm, then we finally conclude paper in Section 7.

2 Related work

There is very limited work available on sorting a swarm of robots. Litus and Vaughan [8] worked on using Double Bracket Flow to build a dynamical system to model the robots' positions. Though this yielded compact and analytical solutions, there were certain constraints that made this approach infeasible in most situations. Firstly, the robots need to be placed on a straight line parallel to some axis initially and the sensors should be able to sense over arbitrarily large distances, which is not practical is many situations.

Krupke et al. [9] used echo (wave) algorithm [10] to develop Parallel Distributed Sorting Algorithm. The algorithm was able to sort the robots in $O(n)$ time complexity where n is the number of robots. But the algorithm was not scalable.

In a recent work Kala [11] showcased navigation of a very large number of robots using collision avoidance. The robots used a geometric method to compute the best heading direction, while also used a deadlock detection and avoidance algorithm.

Golas et al. [12] also proposed a solution for collision avoidance between a large number of agents. The authors also considered for long-range collision avoidance. The collision avoidance in all these algorithms is applied to heterogeneous multi-robot motion planning problem rather than robotic swarms.

Shang et al. [13] proposed a method to sort swarm of robots based on their hardware variation by categorizing their behavior. They have modeled robots with IR sensor, PI controller and differential drivetrain. 210 robots were trained and tested in two different arenas (differ by the reflective pattern). The authors intend to use their technique in automatic assignment of tasks to the robots based on their behavior.

Kumar et al. [14] proposed decentralized self sorting technique for heterogeneous robots, which uses differential artificial potential field. Their approach is able to segregate robots of type A from the robots of type B.

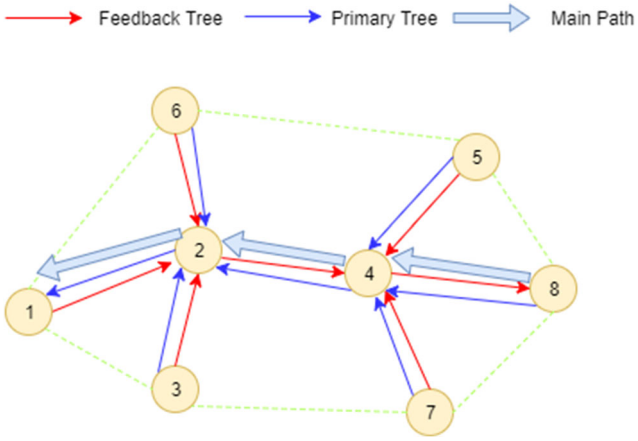

Fig. 1 Main Path Construction

Saad et al. [15] proposed an algorithm for distributed network topology design. The algorithm is inspired by the bee colony and was formulated as multi-objective optimization problem.

Nitschke et al. [16] presented that Collective Neuro-Evolution (CONE) helps better in evolving controller in simulated robots in a better way as compared to other methods. They tried to study the evolution in structure building task in which robots have to co-operate and put block in a sequential manner to build a structure.

Ding et al. [17] proposed a gossip based sorting technique based on cluster size estimation. The approach sorts different set of robots (type A from type B) from mixed group. In this approach robots exchange information with their neighbour to find a majority, when majority exceeds a threshold limit, majoritarianism is said to form a barrier and minorities are asked to leave to another free space. Authors have used ARGoS simulator to validate their approach and experimented with two groups of robots.

Kalles et al. [18] proposed *Emerge Sort* which follows ant based algorithms. The overall sorting is achieved by using the local operators. They have used triplets of robots with direction bias as pheromones to achieve the overall sorted behavior. One advantage is that there is no need to tell in advance which way to sort, direction bias takes care of it.

3 Sorting algorithm

Zhou et al. [6] proposed distributed sorting algorithm. In the algorithm, the authors have defined two operations: *topological* and *geometric*. Topological operations are responsible for maintaining the communication among the robots while the geometric operation is responsible for the actual movement of the robots. Model assumption and the operations of the algorithm as defined by the authors in [6] are described in the following subsections.

3.1 Model assumptions

We have n number of robots. Initially all the robots form an undirected communication graph $G(V, E)$. The robots form the vertices (V) of the graph. There exists an edge between the two robots a and b if the robots are close enough so that they can communicate with each other without the help of any other robot c. We assume that the communication range is a disk of radius r and there exist bidirectional communication between the two robots. A robot is in the neighbourhood of the other robot if both of them can communicate directly. Each robot has a unique serial number. Robots have some attribute value v. If there is a tie between the attribute values of the two robots the tie is broken with the help of serial number sn, i.e. comparison is based upon ordered pair $< v_i, sn_i >$.

The robot with minimum value is known as the *global minimum* and one with the maximum value is known as the *global maximum*.

All the robots are equipped with a timer and all of them share the same fixed time interval. After each interval all the robots exchange their information with their neighbors.

The assumptions made can be summarized as follows:

1. Initially the communication network is connected and remains connected till the execution of the algorithm. If this rule is violated we can not recover from the failure.
2. A robot neither collides nor it fails. If it fails it is completely removed from the system.
3. All the robots travel at the same speed.

Since the algorithm presented here is a distributed algorithm, no such type of algorithm works with a disconnected network and hence network connectivity has to be maintained while the robots move. The assumption that robots do not fail means that any robot which fails, is removed from the system and hence no failed robots participate in the sorting. The third assumption that all the robots travel at the same speed is just to keep the model simple and an easy implementation in the simulated environment. Similar assumptions are made in [19] and [20].

3.2 Algorithm

The algorithm presented in [6] is the modified version of *Breadth First Search* (BFS) and is divided into two types of operations: *topological operations* and *geometric operations*. The algorithm proceeds by making a data structure called the *main path*. Topological operations are

Fig. 2 Insertion operation if $p < s < u$, here $p = 1$, $s = 3$ and $u = 5$

used for the formation of the main path, deleting and adding the edges as required. Geometric operations deal with the actual motion of the robots.

3.2.1 Constructing the main path

Main path consists of the edges which connects the global minimum, the global maximum and is initially the shortest path between the two. The main path is constructed using BFS algorithm with the help of two spanning trees *primary tree* and the *feedback tree*.

Primary tree construction starts from the global minimum and requires parent selection by each robot from one of its neighbouring robots. The selected parent is called the *primary parent*. Global minimum has no primary parent. The robots represent the vertices and parent selection (directed edges) represents the edge. The robot chooses one of its neighbor as parent and stores its serial number. These parents may change as the algorithm proceeds. We use BFS spanning tree to construct the primary tree.

Feedback tree construction uses the same approach as described above except that it starts from global maximum and parent selected is known as *feedback parent*. Global maximum has no feedback parent. Node a selects node b as a parent if node b is either parent of node a in primary tree or node b is child of node a in primary tree.

Based on the above two spanning trees we construct the main path as follows:

a. A robot is said to be on the main path if both of its parents (primary and feedback) are different.
b. A robot is said to be in branch if both of its parents (primary and feedback) are same.

Figure 1 shows a connected graph of eight robots, annotated with the values according to which they are to be sorted. Robot with value 1 is global minimum and robot with value 8 is global maximum. Blue arrows represent the primary tree rooted at the global minimum and the red arrows represent the feedback tree rooted at global maximum. Arrow direction tells about the parent information, for example, 1 is parent of 2 in primary tree.

3.2.2 Geometric operations

Geometric operations are responsible for the motion of robots. Following are the two types of motion defined for the robots in the main path and in the branches:

a. A robot in the main path moves towards the mid point of its primary and feedback parent.
b. A robot in the branch moves towards its parent.

Global minimum and global maximum do not move.

3.2.3 Main Path Topological Operation

There are two main operations defined on main path: *Insertion* and *Deletion*.

Insertion operation adds a branch robot to the main path. The operation is defined such that a robot is inserted to the main path in the correct local order. If u is a main path robot and it has two neighbors p and s, if:

a. p is parent of both u and s i.e. u and s are siblings
b. s is in branch
c. p is in main path
d. $p < s < u$

then, u chooses s as its parent and s is inserted into the main path.

Figure 2 depicts the insertion operation. Black arrows represent the main path and red arrows represent that the robot is in the branch. Robot 1 (p) is parent of both robots 3 (s) and 5 (u). Robot 3 is inserted into the main path according to the condition mentioned.

Deletion operation deletes a locally unordered robot from the main path. If u is a robot on the main path, p and g are neighbours of u and are on the main path, if:

a. p is parent of u
b. g is parent of p i.e. g is grandparent of u
c. $g > p$ or $p > u$

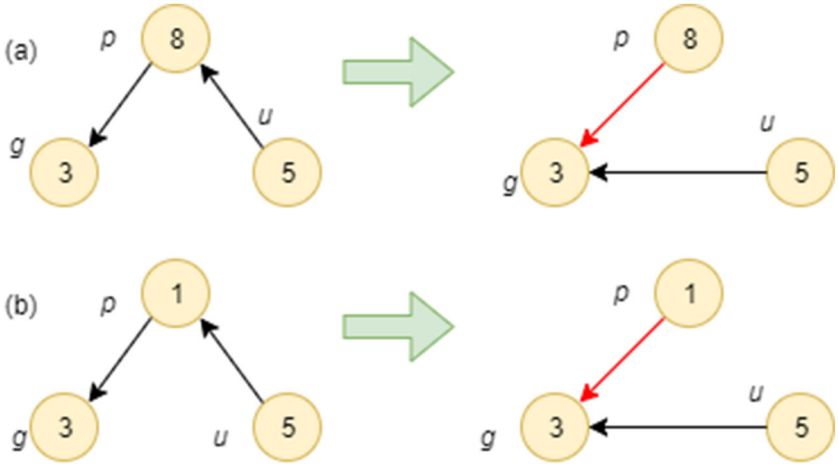

Fig. 3 Deletion operation if $g > p$ or $p > u$ in (a) $p = 8$, $g = 3$ and $u = 5$ in (b) $p = 1$, $g = 3$ and $u = 5$

then, u selects g as its parent instead of p and p is deleted from the main path.

Figure 3 depicts the deletion operation from the main path. (a) part of the Figure shows the case when $p > u$ where robot 8 is p, robot 3 is g and robot 5 is u. (b) part of the Figure shows the case when $g > p$ where robot 3 is g, robot 1 is p and robot 5 is u. Black arrows represent the main path and red arrows represent branches.

3.2.4 Branch topological operation

Algorithm completes when all the robots are inserted into the main path and are arranged in order. To insert the branch robots into the main path, two branch topological operations: *Navigation to Minimum* and *Navigation to Maximum* are defined.

Navigation to Minimum operation is responsible for the movement of a branch robot towards the global minimum. In the local sense it makes a robot move closer to its grandparent. If u is in a branch and it has neighbouring robots p and g, if:

a. p is on main path and p is parent of u
b. g is parent of p i.e. g is grandparent of u and g is on main path
c. $u < p$

then, u selects g as its parent.

Figure 4 shows *navigation to minimum* operation. Here robot 5 is p, robot 3 is g and robot 4 is u. Black arrows represent the main path and red arrows represent the branch.

Navigation to Maximum operation is similar to the *Navigation to Minimum* operation except it allows a robot to move towards the global maximum. If u is a branch robot, p and s are neighbors and are on the main path, if:

a. p is parent of u
b. p is parent of s i.e. s and u are siblings
c. $u > p$ and $u > s$

then, u selects s as its parent.

Figure 5 shows *navigation to maximum* operation. Here robot 3 is p, robot 8 is u and robot 5 is s. Black arrows represent the main path and red arrows represent the branch.

4 Collision avoidance

The greatest limitation of the work of Zhou et al. [6] is that the authors do not incorporate any collision avoidance algorithm that makes the results unrealistic. The purpose of this section is to propose integration of collision avoidance in the same approach.

The foremost decision is the selection of the collision avoidance algorithm. The artificial potential field [21] and fuzzy based algorithms are not good candidates since the

Fig. 4 Navigation to minimum operation, here $p = 5$, $g = 3$ and $u = 4$

robots should finally make a straight line even at high densities, while the potential based algorithms can make it very difficult for a robot to squeeze itself in-between two robots already in a sorted line. Similarly, the velocity obstacle approach [22] is not a good candidate because the simple robots may not be in a position to estimate the other robot's speed.

Especially considering that the robots need to ultimately make a line, the geometry based algorithms are a good choice. Since the robots are cheap and proximity sensing of a limited resolution, approaches (e.g. [23]) assuming precise locations of the other robots are undesirable. The most relevant work is hence by Sezer et al. [7] which attempts to maintain a very high clearance from other robots and can thus work with very limited sensing and very large errors.

Sezer et al. in [7] proposed an obstacle avoidance method named as *Follow the Gap Method*. The method works by first constructing a gap array in the proximity of the agent which can be sensed using the appropriate sensors, and then it calculates heading angle to avoid the obstacles while considering the goal position at the same time. Authors have assumed robots and obstacles to be spherical in shape with radius r and R respectively. To calculate the head angle, authors have first assumed the agent as a point robot and inflated the obstacles by a radius r, hence the obstacles are now of radius $r + R$. A reference line is chosen along which all the angular measurements are to be done. From the point robot, left (θ_l) and right (θ_r) border angles are measured for all the obstacles. Figure 6 shows angular measurement for an obstacle where $\angle bcx = \theta_r$ and $\angle acx = \theta_l$.

All the angular ranges excluding the border angle range are included in the gap array. Then from the gap array the direction with maximum gap is chosen. If there are two obstacles A and B, then heading angle (θ_h) is calculated using the approximation in (1).

$$\theta_h = \frac{\theta_r^A + \theta_l^B}{2} \tag{1}$$

Figure 7 shows an illustration of heading angle calculation where $\angle bcx = \theta_l^B$ and $\angle acx = \theta_r^A$. The algorithm can be summarized as follows:

a. First make the robot a point robot and inflate all the obstacles.
b. Maintain an array of all the obstacle angles.
c. Using the array of obstacle angles, find the largest gap.
d. Using the obstacles which inscribe the largest gap, calculate the heading angle.

Algorithm 1 shows the pseudocode for sorting algorithm and algorithm 2 shows the pseudocode for calculating the new position (towards the mid point of primary and feedback parents). To check if a position x is collision prone or not, we can assume that the center of a robot is occupying position x. Then we can compute *Euclidean Distance* ($dist(x, y)$) from the center point y of all the other robots. Here, r is radius of the robot. From (2) if $dist(x, y)$ is less than or equal to $2 * r$ then there is collision between the two robots centered at position x and y respectively. Suitable changes can be made in Algorithm 2 to make it *moveTowardsParent*.

$$dist(x, y) \leq 2 * r \tag{2}$$

Algorithm 1 Sorting(robots).

```
Result: Robots sorted in a straight line
Construct primary tree;
Construct feedback tree;
foreach robot i in the connected graph do
    if robot(i).primary_parent
    ≠ robot(i).feedback_parent then
        robot(i).status = main_path;
        foreach robot j adjacent to robot i do
            insert(i, j);
            delete(i, j);
        end
        if robot is neither global_minimum nor
        global_maximum then
            moveToParentMidPoint
            (robot(i).feedback_parent_pos,
            robot(i).primary_parent_pos);
        end
    else
        robot(i).status = branch;
        foreach robot j adjacent to robot i do
            navigationToMinimum(i, j);
            navigationToMaximum(i, j);
        end
        if robot is neither global_minimum nor
        global_maximum then
            moveTowardsParent
            (robot(i).feedback_parent_pos);
        end
    end
end
```

Fig. 5 Navigation to maximum operation, here $p = 3$, $s = 5$ and $u = 8$

Algorithm 2: moveToParentMidPoint (primary_parent_pos, feedback_parent_pos).

Result: New position of robot
mid_point = calculate mid point of the parent's position;
action = a step towards mid point;
if *action is collision prone* **then**
  gap_array = calculate gap array by sensing around the proximity;
  largest_gap = find the largest gap from the gap array;
  head_direction = calculate heading direction using the largest_gap;
  action = take a step in head_direction;
end
return action;

5 Experiment and result

To perform the experiment we have developed simulation codes. The appropriate parameters, for example, communication range and radius of robots can be varied as per requirement. We have also assumed a grid of 100 × 100 as workspace for robots. We have implemented the algorithm in [6] and have tested it with 15, 30, 45 and 60 robots.

The number of iterations an algorithm takes to terminate both in the case of collision detection and no collision detection are compared. Figure 8 represents the initial configuration of fifteen robots in their workspace. Attribute value of robots (according to which sorting is to be done) is annotated. Blue and black lines represent primary and feedback trees respectively. Figure 9 shows algorithm in action after few iterations. Figure 10 shows the algorithm's output a few iterations before its termination. Table 1 shows the comparison between the algorithms with and without collision detection.

From Fig. 11, we observe that as the number of robots increase there is approximately linear increment in the number of iterations of the algorithm. While from Fig. 12, we observe that the number of iterations is increasing exponentially with an increase in the number of robots. As the number of robots increases so does the number of collision which resembles the real world for example; if there are more people in the same workspace, chances of a collision increases. Hence to avoid collision, robots may divert and may take longer route to reach their goal position leading to the more number of iterations.

6 Discussion

The algorithm presented in this paper is highly adaptive as compared to algorithm in [9]. If a new minimum or maximum value robot is added to the system, Wave algorithm [9] has to exchange $O(n^2)$ messages in order to find out the new minimum and maximum robot, while

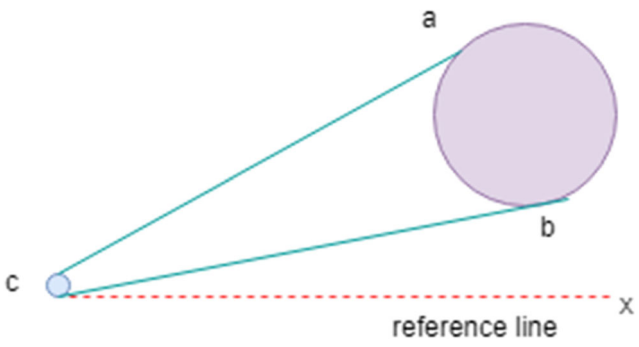

Fig. 6 Angle Measurement

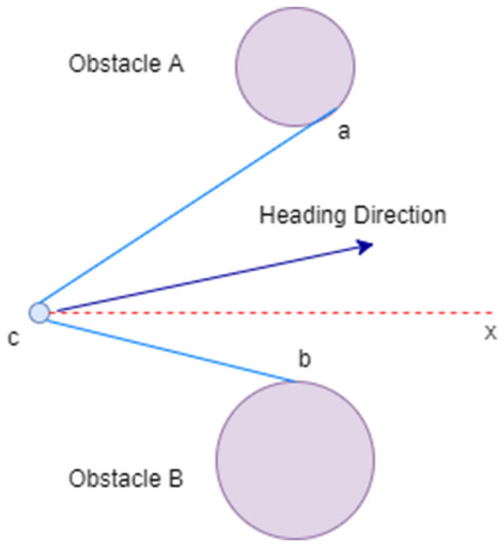

Fig. 7 Heading Direction

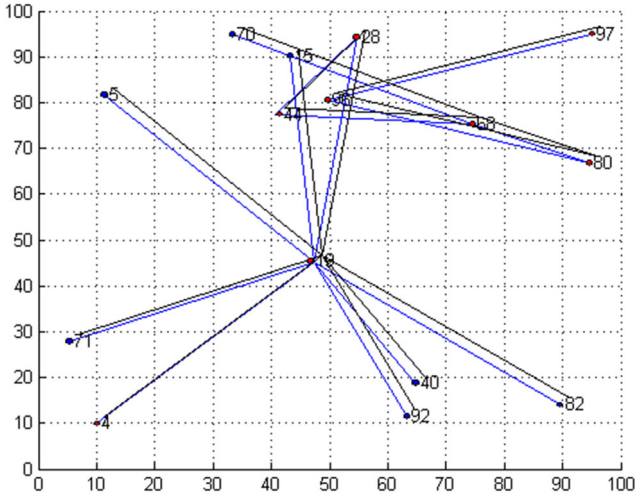

Fig. 8 Initial Configuration

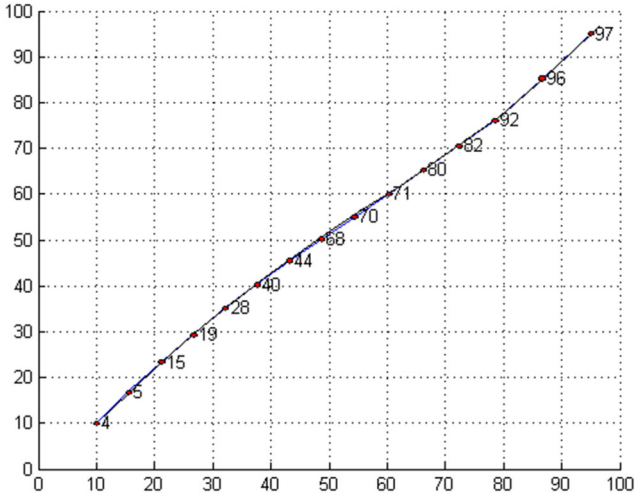

Fig. 10 Sorted robots almost in a straight line

in our case topological operations will take care of it. For topological operations in the presented algorithm there are n robots with n-1 parent child connections and since the graph is bidirectional there can not be a cycle. Out of insertion, deletion and navigation, no topological operations produces a cycle as none of them are allowed to select a descendant as a parent, similarly there are only two geometric operations: move towards parent, and move towards the mid point of primary and feedback parent. Move towards the mid points is a main path operation and in doing so the distance between parent and child does not increase [24], hence the main path does not break. Move towards the parent is a branch operation and hence the distance between the parent and child will always be decreasing. In conclusion both the operations are safe. The feedback tree is generated after the primary tree and the tree is made in such a way (by restricting the selection of parents or expansion, explained in Section 3.2.1), such that the main path is present in the feedback tree as well. The purpose of the feedback tree is to allow the robots to know if they are in the main path. Hence the cycles will never be formed.

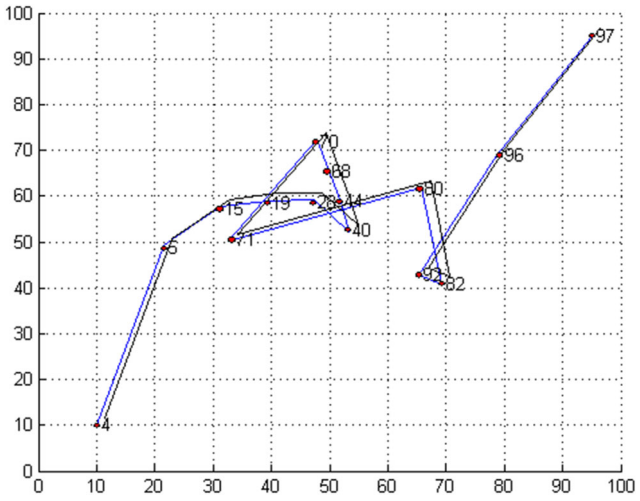

Fig. 9 Sorting algorithm in action

A robot in the correct order and close enough to main path is accepted by it. Once in main path robot tries to reach to the mid point of its parents (primary and feedback) which places it in visual main path (straight line joining global minimum and maximum) and therefore for large number of robots main path will have all the robots connected in order of their priority. However robots may not be able to fit in physically and hence they may form a zig-zag path. The situation is shown in Fig. 13.

When two branch robots are on different sides of the path, both the robots will keep moving towards their parent, but only one of them can be inserted at the same position in one round. Multiple insertions can occur at the same time but with different parent child pair. If both of them try to get inserted at the same position then one request has to be dropped. This is implemented using a handshaking between every pair of robots that form a parent child relationship. Assume the robots i and j simultaneously request the robot k to act as the parent. k will only accept the request that comes first and will perform the handshaking (say i). Since

Table 1 Comparison of Algorithms without collision and with collision detection

Number of robots	Number of iterations (No Collision Avoidance)	Number of iterations (Collision Avoided)
15	132	138
30	155	230
45	174	1252
60	196	2960

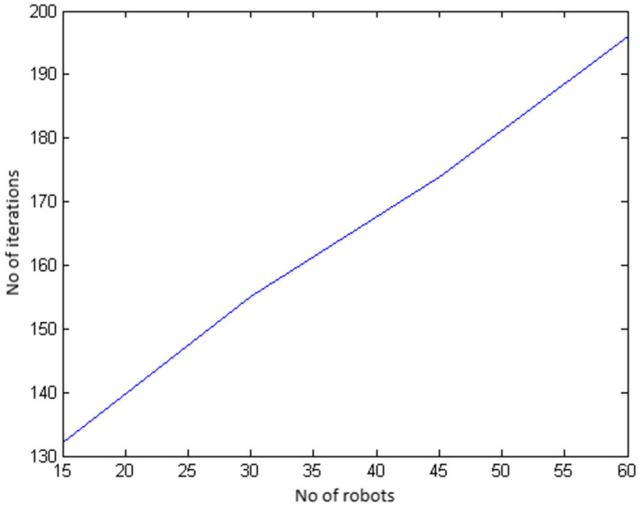

Fig. 11 Number of iterations of algorithm without collision detection

j does not get a positive response, it knows that it could not get inserted in the path and looks for an alternative attempt as per the functioning of the algorithm. It must be stressed that the term lying on the main path means symbolically in the path represented by the trees and not geometrically to line in the straight line joining the parents. The symbolic joining is hence an atomic operation.

As long as the communication graph does not break, follow the gap method for collision avoidance can take care of obstacles other than robots itself. It must be noted that a reactive planning technique is used for the physical navigation of the robot, which cannot counter complex obstacles and especially the concave ones. These limitations apply to the proposed approach as well. As a simple example, consider the problem shown in Fig. 14. The robot will select a gap as shown in Fig. 14. The robot at every step selects this gap, which makes it away from the obstacle

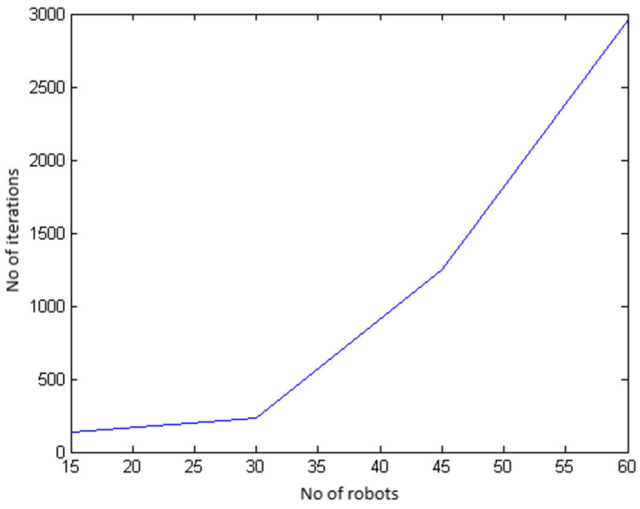

Fig. 12 Number of iterations of algorithm with collision detection

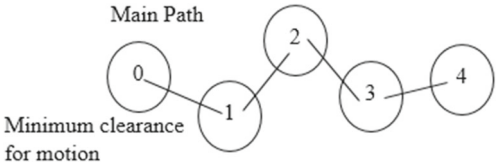

Fig. 13 A case when the distance between minima and maxima is not large enough to accommodate all robots

(other than the robot shown in black). After some time, the robot will be sufficiently away from the obstacle, wherein its direction of motion to goal is clear and the robot can safely move towards the goal.

What is wrong with just doing collision avoidance reactively when the robots try to go to their pre-allocated places? This can be one of the possible argument against the work presented in this paper, but, in the presented work, robot's positions are not pre-allocated. The positions are initialized randomly, their position in the final sorted sequence is the result of the algorithm. The algorithm can be terminated when the distance between the mid-point of parents and robot's position is below a threshold. The sorting implemented is purely decentralized. In a centralized scheme it is possible to get all IDs, to sort them and to interpolate a line to get their final positions; making it a multi-robot motion planning problem. However, in this approach there is no centralized robot that has the capacity to do a central sorting. The sorting happens in a purely decentralized manner as the robots move, based on their neighbors. The collision avoidance is done reactively. The algorithm without collision checking has a $O(n^2)$ complexity in the worst case, while the average case complexity is near-linear, also clear from the graphs. In case of collision checking, with every move associated with a robot, a robot may have to wait for all the other robots to clear one after the other. So, the complexity becomes $O(n^3)$ in the worst case, while it is near-quadratic in the average case. This is the complexity shown.

The controller presented in [14] is able to sort the heterogeneous robotic swarm but it uses artificial potential field methods which are easy to implement but we can not guarantee that it will always work, these methods can converge to a local minima instead of a desired goal. Graph based algorithm is robust, ensures a sorted sequence in ascending order as proved in [6].

Gossip based sorting technique [17] segregates robots of type A from robots of type B but the algorithms don't have any on the fly collision avoidance mechanism rather robots stop and wait for others to clear the arena in order to move. While collision avoidance method in this paper calculates the heading direction dynamically based on current position of the robots.

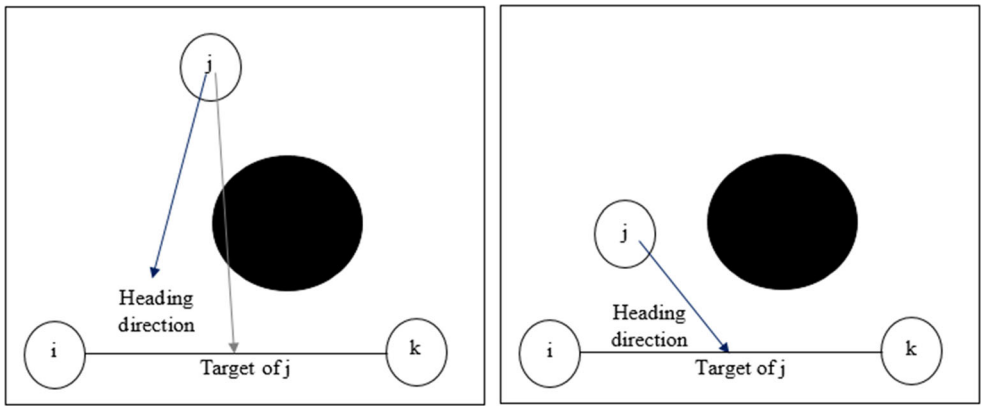

Fig. 14 Collision avoidance in case of obstacles other than robots itself

7 Conclusions

There is no collision avoidance in the baseline algorithm of this experiment, the introduction of collision avoidance is the contribution of this paper. The original paper modelled swarms where the individual robots were assumed to be point sized and hence there was no need of a collision avoidance scheme. Practically the robots have a size, which is added in the proposed method.

We have successfully implemented the algorithm presented in [6] and integrated it with the collision avoidance technique, Follow the Gap Method [7]. The paper aimed at making the approach of Zhou et al. [6] realistic in nature by the introduction of collision avoidance. The results show that there can be a significant difference in results with the introduction of collision avoidance, with the number of iterations increasing exponentially with the number of robots. Overall, the proposed work successfully carries the decentralized sorting of robots while also avoiding collisions. This paper only discusses about sorting the robots in a straight line. We would like to extend our work so that robots can sort themselves in some simple 2-D shapes, for example, square, circle, etc.

Acknowledgements Authors would like to acknowledge Akanshi Mittal for her valuable contribution in producing a flawless document.

References

1. Tan Y, Zheng Zy (2013) Research Advance in Swarm Robotics. Defence Technol 9(1):18–39
2. Navarro I, Matia F (2013) An Introduction to Swarm Robotics. ISRN Robot 2013:1–10
3. Brambilla M, Ferrante E, Birattari M, Dorigo M (2013) Swarm robotics: A review from the swarm engineering perspective. Swarm Intell 7(1):1–41
4. Lei X, Fang M, Fujita H (2019) Moth-flame optimization-based algorithm with synthetic dynamic ppi networks for discovering protein complexes. Knowl-Based Syst 172:76–85
5. Al-zoubi AM, Mirjalili S, Fujita H, Lei X, Fang M, Fujita H (2018) An efficient binary salp swarm algorithm with crossover scheme for feature selection problems. Knowl-Based Syst 154:43–67
6. Zhou Y, Goldman R, Mclurkin J (2016) An Asymmetric Distributed Method for Sorting a Robot Swarm. IEEE Robot Autom Lett 2(1):261–268
7. Sezer V, Gokasan M (2012) A novel obstacle avoidance algorithm: "Follow the gap method". Robot Auton Syst 60(9):1123–1134
8. Litus Y, Vaughan RT (2010) Fall in! Sorting a group of robots with a continuous controller. CRV 2010 - 7th Canadian Conference on Computer and Robot Vision, pp 269–276
9. Krupke D, Hemmer M, McLurkin J, Zhou Y, Fekete SP (2015) A parallel distributed strategy for arraying a scattered robot swarm. IEEE International Conference on Intelligent Robots and Systems 2015-December, pp 2795–2802
10. Chang EJ (1982) Echo algorithms: Depth parallel operations on general graphs. IEEE Trans Softw Eng SE-8(4):391–401
11. Kala R (2018) Routing-based navigation of dense mobile robots. Intel Serv Robot 11(1):25–39
12. Golas A, Narain R, Curtis S, Lin MC (2013) Hybrid Long-Range Collision Avoidance for Crowd Simulation. IEEE Trans Vis Comput Graph 20(7):1022–1034
13. Shang B, Crowder R, Zauner KP (2014) Swarm Behavioral Sorting based on Robotic Hardware Variation. 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH), pp 631–636
14. Kumar M, Garg DP, Kumar V (2008) Self-sorting in a swarm of heterogeneous agents. Proceedings of the American Control Conference, pp 117–122
15. Saad A, Khan SA, Mahmood A (2018) A multi-objective evolutionary artificial bee colony algorithm for optimizing network topology design. Swarm and Evolutionary Computation
16. Nitschke GS, Schut MC, Eiben AE (2012) Evolving behavioral specialization in robot teams to solve a collective construction task. Swarm and Evolutionary Computation
17. Ding H, Hamann H (2014) sorting in swarm robots using Communication-Based cluster size estimation. In: Dorigo M et al (eds) Swarm intelligence. ANTS 2014. Lecture notes in computer science, vol 8667. Springer, Cham, pp 262–269
18. Kalles D, Mperoukli V, Papandreadis A (2012) Emerge-Sort : Swarm intelligence sorting. In: Maglogiannis I, Plagianakos V, Vlahavas I (eds) Artificial intelligence: Theories and applications. SETN 2012. Lecture notes in computer science, vol 7297. Springer, Berlin, pp 98–105

19. Ding H, Hamann H (2014) Sorting in swarm robots using communication-based cluster size estimation. Swarm Intelligence, Springer International Publishing, pp 262–269
20. Degener B, Kempkes B, Kling P, Meyer auf der Heide, F (2010) A continuous, local strategy for constructing a short chain of mobile robots. Structural Information and Communication Complexity. Springer, Berlin, pp 168–182
21. Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. Autonomous Robot Vehicles, pp 396–404
22. Fiorini P, Shiller Z (1998) Motion planning in dynamic environments using velocity obstacles. Int J Robot Res 17(7):760–772
23. Ye C, Webb P (2009) A sub goal seeking approach for reactive navigation in complex unknown environments. Robot Auton Syst 57(9):877–888
24. Degener B, Kempkes B, Kling P, Meyer auf der Heide, F (2010) A continuous, local strategy for constructing a short chain of mobile robots. In: Patt-Shamir B, Ekim T (eds) Structural information and communication complexity. Springer, Berlin, pp 168–182