Decentralized Method for Sub-swarm Deployment and Rejoining*

Meghan Chandarana[1], Wenhao Luo[2], Michael Lewis[3], Katia Sycara[2], and Sebastian Scherer[2]

***Abstract*—As part of swarm search and service (SSS) missions, robots are tasked with servicing jobs as they are sensed. This requires small sub-swarm teams to leave the swarm for a specified amount of time to service the jobs. In doing so, fewer robots are required to change motion than if the whole swarm were diverted, thereby minimizing the job's overall effect on the swarm's main goal. We explore the problem of removing the required number of robots from the swarm, while maintaining overall swarm connectivity. By preserving connectivity, robots are able to successfully rejoin the swarm upon completion of their assigned job. These robots are then made available for reallocation. We propose a decentralized and asynchronous method for breaking off sub-swarm groups and rejoining them with the main swarm using the swarm's communication graph topology. Both single and multiple job site cases are explored. The results are compared against a full swarm movement method. Simulation results show that the proposed method outperforms a full swarm method in the average number of messages sent per robot in each step, as well as, the distance traveled by the swarm.**

I. Introduction

In swarm search and service missions (SSS) robot swarms are tasked with searching a predefined area while immediately servicing jobs as they are sensed. Swarms use decentralized control laws to maintain robustness to individual robot failures, as well as, the addition of new robots. Applications range from surveillance of suspicious sites/targets to wildfire applications where swarms are tasked with putting out brush fires that have sparked from embers. New jobs "arrive" as they come within sensing range of a swarm robot. Each job requires a robot or subset of robots from the swarm to break off, travel to the job site and remain there until the job is successfully serviced. The robots that break off from the swarm must also maintain connectivity in order to rejoin the swarm upon completion of their job. Once robots rejoin the swarm, they are made available for reallocation elsewhere. Several jobs may be sensed simultaneously or while robots are already in the process of being deployed to another job site (Figure 1).

The work presented in this paper explores the problem of breaking off and deploying a required number of robots from the swarm to the job site(s) for servicing, as well as, rejoining those robots with the swarm after the job is serviced. We consider the additional constraint of maintaining connectivity. This requires that 1) robots are broken off from the swarm without graph disconnection and 2) connectivity is maintained between sub-swarm robots and the original swarm as they move towards the job site. Conventional swarm connectivity control laws consider the effect of connectivity constraints on the motion of the full swarm [1]. This can lead to unnecessary motion changes for some robots. To reduce each job site's effect on the whole swarm, a framework requiring only a subset of robots to switch their motions is needed for sub-swarm assignment and navigation. The framework must be flexible enough to manage multiple job sites with overlapping service times.

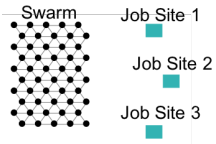


Fig. 1: Example scenario where multiple jobs need servicing.

Past research has focused on moving an entire swarm from one location to another. Methods include both multi-robot path planning approaches where each robot's path is explicitly defined [2] [3] and swarm control approaches where local control laws such as flocking [4] [5] and formation control [6] lead the group towards a goal location. In [7], Chen et al. present a control law for splitting a swarm into multiple sub-swarm teams. Although the number of sub-swarm teams can be controlled by parameter selection, the explicit number of vehicles that end up in each sub-swarm team is not defined. In addition, the overall connectivity of the swarm is not maintained, thereby eliminating the swarm's ability to reallocate robots to future jobs that arrive.

In [8] and [9] a simplical complex from algebraic topology [10] is utilized to incrementally move robots through space, while maintaining swarm connectivity. Ramaithitima et al. use the fence simplex to "push" robots into positions, which results in triangular lattice packing positions so that complete sensor coverage of an area is achieved [8]. The decentralized method presented by Li et al. in [9] preserves connectivity by incrementally "pushing" robots forward by defining frontier nodes based on the expansion of fence simplices in the direction of a goal to navigate an entire swarm through a cluttered environment.

The contributions of this work are as follows. First, we present a decentralized method for selecting and breaking off robots to form a sub-swarm team at a given job site without breaking the swarm's connectivity. Second, we leverage the topology of the communication graph to incrementally move the broken off robots towards the job site while maintaining connectivity with the original swarm. Lastly, we present a way to rejoin the sub-swarm with the swarm. The method is applied to both single job site and multiple job site cases.

[1]The author is with Mechanical Engineering at Carnegie Mellon University, Pittsburgh, PA, USA {mchandar@cmu.edu}

[2]The authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA, USA {luo@cmu.edu, katia@cs.cmu.edu, basti@andrew.cmu.edu}

[3]The author is with Information Sciences and Intelligent Systems at the University of Pittsburgh, Pittsburgh, PA, USA {ml@sis.pitt.edu}

*Sponsored by NASA LaRC (NIA Activity 201020) and AFOSR grant FA9550-15-1-0442.

II. Preliminaries

Consider a robot swarm of N vehicles whose positions, $p_i \in \mathcal{R}^m$ with $m \in \{2,3\}$, form a triangular lattice with interagent distance defined as $\|p_i - p_j\| = R_c, \forall j \in \mathcal{N}_i$, where R_c is the communication range of every robot and $\mathcal{N}_i$ is the set of all neighbors of robot i (Figure 2). Triangular lattices are used to conveniently move swarms through an environment [4] [11]. Due to the limited number of neighboring robots, the triangular lattice formation results in reduced computation and high scalability [12]. Each robot has 6 neighbors unless they are located on the boundary of the swarm, which results in fewer neighbors. All robots maintain an equilateral triangle with their neighbors. A controller similar to the one described in [11] can be used to form the swarm's initial triangular lattice.

Assume that every robot knows their position p_i within a common reference frame. Each robot in the swarm is assigned a unique identifier (UID). We assume the UIDs are $i \in \{1, 2, ..., N\}$. The swarm's communication graph is given by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Every node $v \in \mathcal{V}$ represents a robot. Each robot i communicates only with its direct neighbors (i.e., $\{j : \|p_i - p_j\| \leq R_c\}$). If robot j is a neighbor of robot i, then edge $(v_i, v_j) \in \mathcal{E}$. We assume the connectivity graph is undirected (i.e., $(v_i, v_j) \in \mathcal{E} \Rightarrow (v_j, v_i) \in \mathcal{E}$) and connected.

III. Method Overview

Each job site, k, is defined by a tuple: $< p_k, n_k, t_k >$, where p_k is the position of the site, n_k is the number of robots required to service it (i.e., size of the sub-swarm team) and t_k is the amount of time necessary to complete the service. The robot that sensed the job site, referred to as the *sensing robot*, is assumed to be closest to the site. For each job site k, the task is to break off robots and form a sub-swarm team of n_k robots at the job site's location p_k. After the job has been serviced, all robots must be rejoined with the original swarm. Although intended as an element of SSS missions, for clarity, the methods below are illustrated using a stationary swarm.

In each time step for each job site, k, the swarm uses decentralized algorithms to break off a new robot and push it towards the job site location, p_k. Each robot broken off moves to fill the previous position of its predecessor, with the first removed robot – or frontier robot – moving to fill a new unoccupied position (frontier node), which lies between it and the job site (Section III-A1). Thus, a chain is formed between the frontier robot and the swarm. As a robot is pulled out of the swarm, a chain of robots behind it move forward to maintain connectivity. The number of robots in the chain is given by the predefined hop radius, H. The last robot to move forward, known as the tail robot, is the furthest robot from the frontier node (Section III-A2). A path from the tail robot to the frontier node is then planned (Section III-A3).

As the frontier robot moves closer to the job site, the number of steps (and robots) that lay between it and the swarm increases. Every robot keeps track of how many steps, s, have been taken since the job site was sensed. A robot who is exactly s hops away from the frontier node considers itself the anchor node. By limiting the number of hops between the tail robot and the anchor node, the probability of overlap between chains for different job sites is decreased. This limits the possibility of robots being needed for multiple chains resulting in them choosing one and finding a replacement for the others. However, an empty position in the swarm known as a hole is left in the tail robot's original position (Figure 2). Continuing to leave the hole behind the tail robot may disconnect the graph in the future. Therefore, the hole is filled by moving a robot behind the tail robot forward to fill the open position. In doing so, the hole is pushed back one spot. Thus a robot behind that one needs to move forward. This continues until the hole is removed from the swarm's graph (Section III-A4).

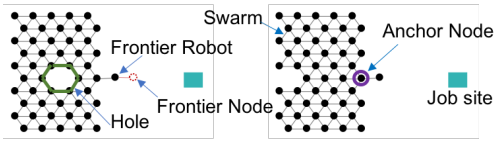


Fig. 2: Hole left in the swarm lattice (left) before the Algorithm 2 is used (right). Lines depict connected robots.

After a sub-swarm team (i.e., size equals n_k) has formed at job site, k, and the required t_k service time has expired, a rejoin action is initiated. To rejoin, boundary robots in the swarm are used to determine the frontier node (Section III-B). The furthest robot from the frontier node is deemed the tail robot. Like before, chain robots then move forward.

A. Sub-swarm Break Off

The anchor node (Figure 2) is initialized as the sensing robot. At the end of each time step, a robot who is s steps away from the frontier node assigns itself as the new anchor node. Each robot only maintains a belief of whether they themselves are the anchor node, and not the status of the other robots in the swarm.

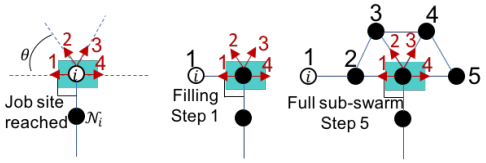


Fig. 3: Sample sub-swarm robot distribution.

1) Frontier Node: When forming the sub-swarm team, the frontier node is chosen as the point $F \in \mathcal{R}^m$ ($m \in \{2,3\}$) that is R_c away from the frontier robot in the direction of the job site, where R_c is the communication distance. Until the sensing robot is within range (distance less than R_c) of the job site, it is always chosen as the frontier robot.

As an example of a symmetric deployment, once a robot, i, reaches the job site, it defines 4 directions equally spaced around the job site (Figure 3 red). Direction 1 is defined as being 90° counterclockwise from the line between robot i's neighbor and robot i. The angle between directions is defined as $\theta = \pi/(m-1)$ where m is equal to the number of directions (i.e., 4). To build a sub-swarm team, the frontier node is chosen

Algorithm 1 Tail Robot Selection

1: **procedure** TAILSELECTION($i, p_i, \mathcal{N}_i, A_i, S_i, G, H$)
2: **if** $A_i = 0 \wedge S_i = 0$ **then**
3: $h \leftarrow \infty$, $m \leftarrow i, m_p \leftarrow p_i$
4: **end if**
5: **for all** $j \in \mathcal{N}_i$ **do**
6: SENDMSG(i, h, p_i)
7: **end for**
8: **while** $\{i', h', p_{i'}\} \leftarrow$ RECEIVEMSG() **do**
9: **if** $h > h' + 1$ **then**
10: $h \leftarrow h' + 1, m \leftarrow i', m_p \leftarrow p_{i'}$
11: **for all** $j \in \mathcal{N}_i$ **do**
12: SENDMSG(i, h, p_i)
13: **end for**
14: **end if**
15: **end while**
16: $t_r \leftarrow i, t_h \leftarrow h, t_d \leftarrow \|p_k - p_i\|$
17: **for all** $j \in \mathcal{N}_i$ **do**
18: SENDMSG(t_r, t_h, t_d)
19: **end for**
20: **while** $\{t'_r, t'_h, t'_d\}$RECEIVEMSG() **do**
21: **if** $((t_h < t'_h) \wedge (t'_h \leq H)) \vee ((t_h > H) \wedge (t'_h \leq H))$ **then**
22: $t_r \leftarrow t'_r, t_h \leftarrow t'_h, t_d \leftarrow t'_d$
23: **for all** $j \in \mathcal{N}_i$ **do**
24: SENDMSG(t_r, t_h, t_d)
25: **end for**
26: **else if** $(t_h = t'_h) \wedge (t_h \leq H) \wedge (t_r \neq t'_r)$ **then**
27: **if** $t_d < t'_d$ **then**
28: $t_r \leftarrow t'_r, t_d \leftarrow t'_d$
29: **for all** $j \in \mathcal{N}_i$ **do**
30: SENDMSG(t_r, t_h, t_d)
31: **end for**
32: **else if** $(t_d = t'_d) \wedge (t_r > t'_r)$ **then**
33: $t_r \leftarrow t'_r, t_d \leftarrow t'_d$
34: **for all** $j \in \mathcal{N}_i$ **do**
35: SENDMSG(t_r, t_h, t_d)
36: **end for**
37: **end if**
38: **end if**
39: **end while**
40: **end procedure**

Algorithm 2 Fill Hole Left in Swarm

1: **procedure** FILLHOLE($i, m, \mathcal{N}_i$)
2: $r_{move} \leftarrow 0$
3: **while** RECEIVETOKEN() **do**
4: **if** $m = m'$ **then**
5: $r_{move} \leftarrow i$
6: **for all** $j \in \mathcal{N}_i$ **do**
7: SENDMSG(m, i)
8: **end for**
9: **end if**
10: **end while**
11: **while** $\{m', i'\}$RECEIVEMSG() **do**
12: **if** $(m = m') \wedge (r_{move} > i')$ **then**
13: $r_{move} = i'$
14: **for all** $j \in \mathcal{N}_i$ **do**
15: SENDMSG(m, i)
16: **end for**
17: **end if**
18: **end while**
19: **end procedure**

as an unoccupied position along 1 of the 4 directions. Figure 3 shows an example with $n_k = 5$. Robots are distributed fully along a given direction in clockwise order. This distribution method is illustrative and can be replaced with a job-type specific distribution.

2) Tail Robot Selection: After finding the frontier node, Algorithm 1 is used to select the tail robot in a decentralized manner. The swarm first constructs a spanning tree rooted at the anchor node such that every robot in the tree is the fewest number of hops away from the root (lines 2-15). This is known as a hop-optimal tree [2]. Only the hop values for robots that are not the anchor node ($A_i = 0$) and are still left in the main swarm ($S_i = 0$) are updated. This forms a tree rooted at the frontier robot without needing to update the hop values of robots in the chain or sub-swarm. Each robot is aware of only its hop value (h), its master's UID (m) and its master's position (m_p) and not the full structure of the spanning tree. "Master" refers to a robot's parent node.

Once the spanning tree is created, the algorithm then finds the furthest robot (tail robot) in the tree within the predefined hop radius, H. Each robot initially believes the tail robot is itself and sets $t_r = i$, $t_h = h$ and $t_d = \|p_k - p_i\|$. It then sends a message to all its neighbors (lines 16-19). When a robot receives a message that indicates another robot is deeper in the tree (i.e., has a higher hop number) than its current belief of the tail robot and is less than the hop limit H, or if its current belief about the tail robot's hop value is greater than H and the other robot's hop value is less than or equal to H (lines 20-25), the robot updates its belief and sends a message to its neighbors (i.e., no longer considers itself the tail robot). Additionally, if a robot receives a message where the other robot's hop value and its current belief of the tail robot's hop value is the same, but the other robot is further away or is equally far away and has a lower UID, then the robot also updates its belief values (lines 26-37). Messages are sent until a consensus is reached and no new messages are created. Since the spanning tree is constructed such that every robot is the fewest number of hops away from the root node, the shortest path from the tail robot to the frontier node is the exact path in the tree from the tail robot to the frontier node.

3) Movement Action: Once the frontier node and tail robot have been selected, if the tail robot is farther away from the goal (job site) than the frontier node it initiates its new position as the position of its master in the spanning tree. Before moving, it sends a message to its master. When a robot receives a message, it sets its new position as its own master's position and sends a message to its own master. This continues until a robot is the frontier robot and its "master" is the frontier node location. Through this chain of movements, the full chain from tail robot to frontier robot moves forward. The remaining robots in the swarm do not move, thereby preserving a majority of the original communication graph. When the tail robot is closer to the goal than the frontier node, the robots cease to move.

4) Fill Hole Left in Swarm: Algorithm 2 mitigates this issue and fills in the hole by moving the robots behind the tail robot in the spanning tree forward one by one. The algorithm is initiated when the tail robot sends a token to all of its neighbors before moving forward, notifying them that its position will be vacated. Its children must then come to a consensus on who should take their master's position. That chosen robot then sends a token indicating its soon to be vacated position. The process repeats until the robot sending the token has no

Algorithm 3 Fill Multiple Holes in Swarm

```
1: procedure FillMultipleHoles(i, m, m_p, N_i, S_i)
2:     r_move ← 0
3:     while {S_i'}ReceiveToken() do
4:         if S_i = 0 then
5:             if m = m' then
6:                 r_move ← i, S_i ← S_i'
7:                 for all j ∈ N_i do
8:                     SendMsg(m, i, S_i)
9:                 end for
10:            end if
11:        else
12:            FindReplacement(m, S_i, m_p)
13:        end if
14:    end while
15:    while {m', i', S_i'}ReceiveMsg() do
16:        if (S_i = 0) ∨ (S_i = S_i') then
17:            if (m = m') ∧ (r_move < i') then
18:                r_move = i'
19:                for all j ∈ N_i do
20:                    SendMsg(m, i)
21:                end for
22:            end if
23:        else
24:            FindReplacement(m, S_i, m_p)
25:        end if
26:    end while
27: end procedure
```

children (i.e., when it is on the opposite edge of the swarm from the anchor node) and thus no messages are sent.

Lines 3-10 show that when a robot in the swarm receives a token, it checks if the sender is its master in the spanning tree. If so, the robot sets its belief of who should fill its master's place as itself and sends a message to its neighbors. When a robot receives a message from its neighbor it checks if it has the same master node (line 12). If so, and its current belief has a higher UID, the robot updates its belief and sends a message to its neighbors (lines 13-17). When no new messages have been sent, the robot whose UID matches its own belief sends a token to its neighbors.

B. Subswarm Rejoin

To pull robots from the sub-swarm back in to the main swarm, each boundary robot nominates a candidate frontier node. Boundary robots have fewer than 6 neighbors, resulting in only a partial triangular lattice surrounding themselves. If a robot has less than 6 neighbors it considers itself a boundary robot and relays that information to its neighbors. Messages are passed between robots until a consensus is reached on the set of all boundary robots. A candidate frontier node is one that lays on a lattice point equidistant from a given boundary robot and its neighboring boundary robot, but is not located in the same place as a current robot (Figure 4). Since every robot knows the boundary robot set and its neighbor's positions, this is found without passing messages. Boundary robots begin the final frontier node selection process by sending their neighbors a message containing their candidate frontier node. Messages are sent until the candidate node closest to the goal (swarm's original centroid location) and furthest away from the anchor node is chosen as the frontier node.

A virtual communication graph, $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ is then defined, where $\mathcal{V}_v$ is the set of all nodes in the initial communication graph plus the frontier node and $\mathcal{E}_v$ is the set of all edges in the initial communication graph plus those between any robots within R_c distance of the frontier node. The virtual communication graph is used to construct a hop-optimal spanning tree rooted at the frontier node. Robots connected to the frontier node initialize $h = 1$, $m = F_{UID}$, $m_p = F$. Every other robot initializes its hop value to infinity and its master node to be itself. A hop-optimal spanning tree rooted at the frontier node is constructed using the same method shown in Algorithm 1 lines 8-15.

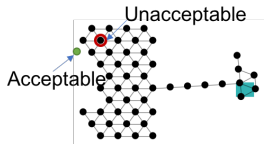


Fig. 4: Acceptable and unacceptable frontier node examples.

As opposed to limiting the depth of the tree search for selecting the tail robot when moving robots in to the sub-swarm (Algorithm 1, lines 16-39), the full tree is used to determine the tail robot. This results in the tail robot being the deepest leaf node. A movement action is then used to push robots forward in a chain from the tail robot position to the frontier node (Section III-A3). This process is repeated until the tail robot is closer to the goal location than the frontier node. This results in a similar rendezvous behavior to that seen at a goal location in [9]. This same procedure is used to move the entire swarm to the final goal location once the sub-swarm robots have rejoined the swarm.

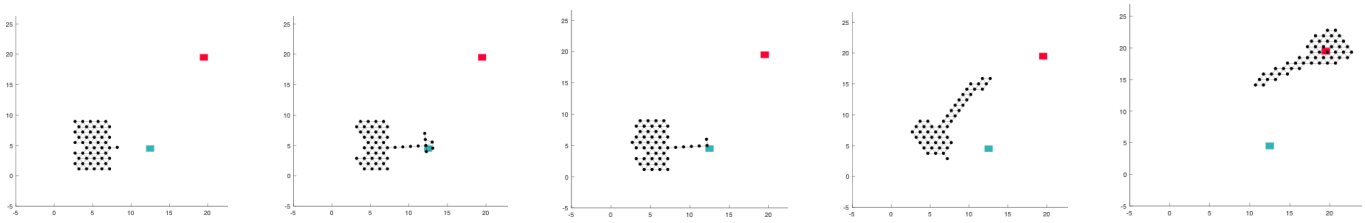
C. Multiple Job Sites

In the multiple site case, one hole remains for each tail robot that moves. Algorithm 3 is used to determine which robots will move forward behind a given tail robot when multiple holes exist in the swarm. Similar to the single hole case, a tail robot sends a token to its neighbors. When a robot receives a token, it has not already been chosen ($S_i = 0$) and its master node is the same as the one in the message, then it sets its belief of who should fill the vacant spot as itself and sends a message to its neighbors (lines 4-10). If it has already been assigned to fill another hole in the swarm, it finds an available replacement neighbor that is also connected to the robot who sent the token (lines 11-13).

When an available robot receives a message, its master node is the vacant node in the message and its UID is lower than the UID specified in the message, it updates its belief to be itself and sends a message to its neighbors (lines 15-22). If it is not available, it finds a replacement robot (lines 23-25). As in Algorithm 2, when no more messages have been sent, the algorithm terminates implicitly. The robot whose belief is itself then sends a token to its neighbors to notify them that its current position will be vacated. The process repeats until the robot who sends a token has no neighbor that is its child.

IV. Simulation

A 2D MATLAB simulation was used to compare the performance of our method to that of a full swarm method



(a) 1 step (b) 10 steps (c) 14 steps (d) 49 steps (e) 92 steps

Fig. 5: Simulation screenshots of 50 robots servicing one job site. The job site is shown in blue and the final goal in red.

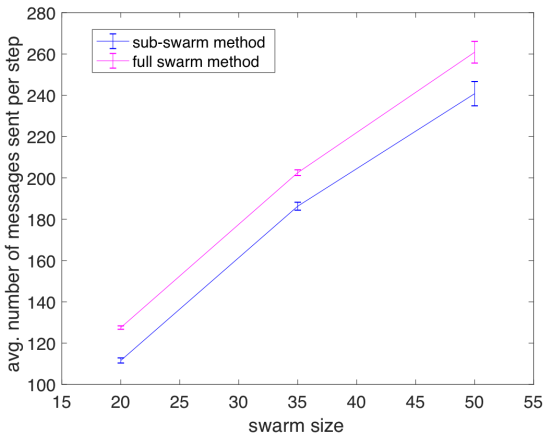


Fig. 6: Average number of messages sent in each step versus the total swarm size in the trial for the one job site condition.

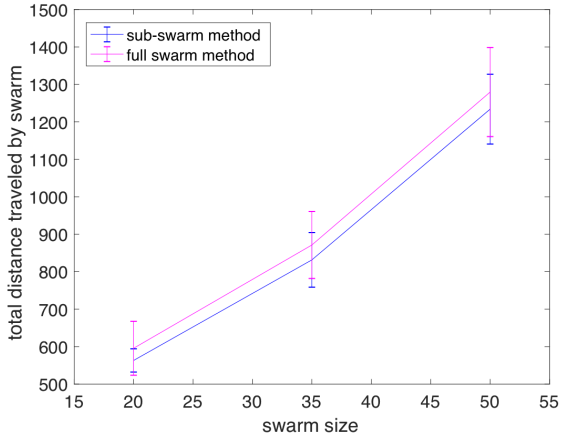


Fig. 7: The total distance traveled by the swarm in the trial versus the total swarm size for the one job site condition.

TABLE I: Avg. Number of Messages vs. Sub-swarm Size

% of Robots Sent to Sub-swarm	Avg. Number of Messages Sent
10%	228.3424
20%	218.5217
30%	204.1177

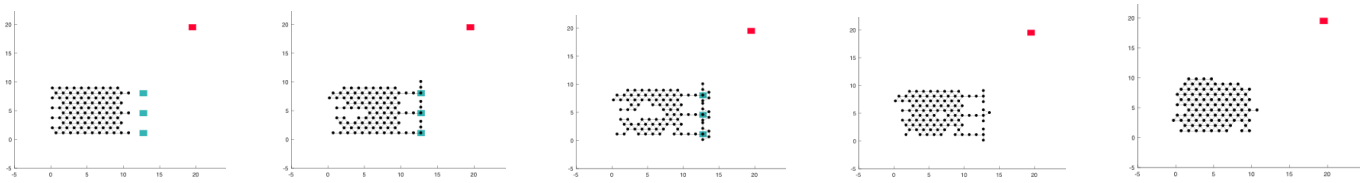
where robots move sequentially to job sites using the rejoin movement in Section III-B (comparable to [9]). For each trial in the single job site condition, the swarm was tasked with moving the required n_k robots to the job site, rejoining with the swarm and then moving to a final goal location. The job site was one of 12 positions equally spaced on a circle centered at the swarm's starting centroid at (5,5) with a radius of 8 units. Swarm sizes of 20, 35, and 50 were tested at each job site. Results were averaged over all locations. The average number of messages sent in each step and the total distance traveled by the swarm are shown. An analysis of variance (ANOVA) was conducted on the data using statistical analysis software IBM SPSS v. 25. In the multiple job site conditions, both two and three simultaneous sites were tested with 100 robots. A single set of job site locations (for both the two and three site conditions) was tested. All job sites were simulated excluding service times. Therefore, as soon as the sub-swarm team is formed the robots rejoin the swarm.

A. Single Job Site

An example single job site trial is shown in Figure 5. Robots begin to form the chain between the job site and swarm in Figure 5a. When a sub-swarm team is formed (Figure 5b) the rejoin behavior is triggered (Figure 5c). Once all robots have rejoined the swarm it moves toward to final goal location at (20,20) (Figure 5d) resulting in the swarm rendezvousing around the final goal (Figure 5e). For all single job site trials, the sub-swarm size, n_k, was 5. The results are compared to a full swarm where the swarm moves toward the job site until n_k robots are within range, moves back to the initial centroid, and finally moves to the goal. All values are averaged over the 12 job site locations. Error bars are shown for the standard deviation in each graph.

Our method outperforms the full swarm method in both average number of messages sent in each time step and total distance traveled by the swarm. Figure 6 shows the average number of messages sent versus the swarm size. The full swarm method consistently sends $\sim$20 more messages. The total distance traveled by the swarm versus the swarm size is shown in Figure 7. Our method travels slightly less distance (blue) than that of the full swarm method (magenta).

The difference between the methods and swarm size is significant for the average number of messages sent ($p < 0.0001$, $\eta^2 = 0.877$ and $p < 0.0001$, $\eta^2 = 0.996$ respectively). Method and swarm size are also a significant factor in the differences seen in the distance traveled ($p < 0.0001$, $\eta^2 = 0.923$ and $p = 0.05$, $\eta^2 = 0.056$ respectively). No interaction is seen between the method and swarm size for the number of messages and distance ($\eta^2 = 0.077$, $\eta^2 = 0.001$ respectively). The linear relationship between swarm size and both the



(a) 1 step (b) 5 steps (c) 8 steps (d) 11 steps (e) 17 steps

Fig. 8: Simulation screenshots of 100 robots servicing 3 job sites. For clarity, the job sites are removed after they are serviced.

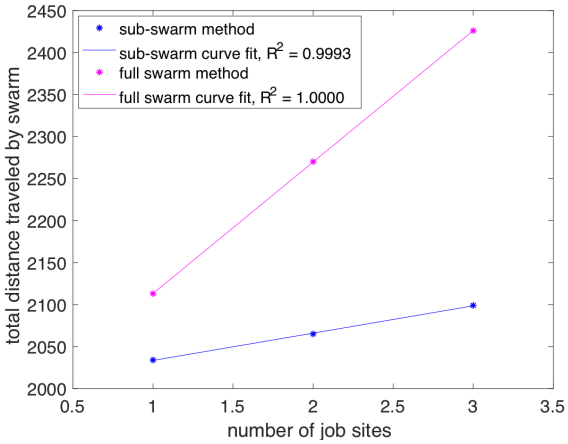


Fig. 9: Total distance traveled versus number of jobs sites.

average number of messages per robot and total distance traveled by the swarm shows the scalability of our method. In addition, the results in Table I show that for a swarm size of 50 as the percentage of robots sent to the sub-swarm increases, the average number of messages sent per robot decreases, reiterating the scalability of our method.

B. Multiple Job Sites

The main difference between the proposed sub-swarm break off method and the full swarm method is their ability to handle multiple job sites. Figure 8 shows an example where 3 job sites must be serviced. Only the sub-swarm break off and rejoining portions of the trial are shown. Figure 8a depicts the chains between job sites and the main swarm. Once robots are within range of the job sites sub-swarm teams are formed (Figure 8b). As soon as the required number of robots is present in the sub-swarm teams (Figure 8c) the rejoin behavior is triggered and robots begin to move back in to the swarm (Figure 8d). As in the single job case, robots rendezvous around the swarm's original centroid before moving towards the final goal.

As opposed to the full swarm method that requires the swarm to move robots to the job site locations individually, our method can send robots to the multiple job sites with overlapping service windows. To demonstrate the advantage of our method, a swarm of 100 robots was given 1, 2 and 3 job sites to service. Figure 9 shows that the overall distance traveled by our method (blue) is significantly less than that of the full swarm method (magenta) for each of the job site cases. The distance increases linearly with the number of job sites, with the full swarm method increasing more rapidly ($R^2_{sub} = 0.9993$ and $R^2_{full} = 1.000$, $p < 0.05$). The difference in distance traveled in the 3 job site case versus the 1 job site case is 4.13x higher for the full swarm method.

V. Conclusion

This paper presents a decentralized method for breaking off robots to reach multiple job sites and rejoining them with the swarm once service is completed. The performance is compared against a full swarm method. Result show that our method: (1) requires less messages and travels less distance in the single job case, (2) scales linearly with the size of the swarm in terms of messages sent per robot and distance traveled by the swarm, (3) results in less messages sent per robot as the ratio of sub-swarm robots to swarm robots increases and (4) allows robots to travel 4.13x less distance than the full swarm method as the number of job sites increases.

References

[1] L. Sabattini, N. Chopra, and C. Secchi, "Decentralized connectivity maintenance for cooperative control of mobile robotic systems," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1411–1423, 2013.

[2] W. Luo, S. S. Khatib, S. Nagavalli, N. Chakraborty, and K. Sycara, "Distributed knowledge leader selection for multi-robot environmental sampling under bandwidth constraints," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 5751–5757.

[3] S. Swaminathan, M. Phillips, and M. Likhachev, "Planning for multi-agent teams with leader switching," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5403–5410.

[4] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on automatic control*, vol. 51, no. 3, pp. 401–420, 2006.

[5] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic control*, vol. 52, no. 5, pp. 863–868, 2007.

[6] X. Li, D. Sun, and J. Yang, "A bounded controller for multirobot navigation while maintaining network connectivity in the presence of obstacles," *Automatica*, vol. 49, no. 1, pp. 285–292, 2013.

[7] Z. Chen, T. Chu, and J. Zhang, "Swarm splitting and multiple targets seeking in multi-agent dynamic systems," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 4577–4582.

[8] R. Ramaithitima, M. Whitzer, S. Bhattacharya, and V. Kumar, "Sensor coverage robot swarms using local sensing without metric information," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3408–3415.

[9] A. Li, W. Luo, S. Nagavalli, and K. Sycara, "Decentralized coordinated motion for a large team of robots preserving connectivity and avoiding collisions," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1505–1511.

[10] H. Allen, "Algebraic topology," 2001.

[11] G. Lee and N. Y. Chong, "A geometric approach to deploying robot swarms," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 257–280, 2008.

[12] G. Lee, N. Y. Chong, and H. Christensen, "Adaptive triangular mesh generation of self-configuring robot swarms," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2737–2742.