Collective transport via sequential caging Supplementary Material

Vivek Shankar Vardharajan[1], Karthik Soma[2], and Gioavnni Beltrame[1]

[1] Polytechnique Montreal, Montreal QC H3T 1J4, Canada,
[2] National Institute of Technology, Tiruchirapalli, India

This supplementary material presents additional details as well as experimental results that provide more insights on the proposed method. In particular, Section 1 presents the Gradient algorithm for task allocation, task update algorithm and Cage stopping algorithm. Section 2 provides additional insights into the simulation experiments.

1 Detailed Methods

In this section, few important algorithms which aids the understanding of the paper are outlined. In short, the robots in the deployment bid for tasks assigned using gradient algorithm. As detailed in the paper, we consider a (SA) single allocation where only one task is assigned to a robot. A seed robot takes up the first task using this gradient algorithm, this robot creates two branches which are grown by the robots until the termination condition is meet. The seed robot spawns two tasks that is assigned to the robots using the gradient algorithm. On reaching an assigned target, the robots apply a correction control to maintain the inter-robot distance. Following, the inter-robot correction the robots check for the termination condition, if it is not satisfied they create a new task using the relative position of the previous robot in the current branch. The remaining robot use the gradient algorithm to determine the appropriate robot for the new task. This process continues until both the branches meet the termination condition. As shown in the paper, if the perimeter of the object satisfies a certain sufficient condition, the two chains will deterministically terminate. The following subsections provides a detailed procedures on the gradient algorithm.

1.1 Gradient Algorithm

The Gradient algorithm allocates tasks to the robots in the deployment cluster. These robots then go on to become the cagers. The first task of the gradient algorithm is fed in as the rough estimate of the the centroid the object to be pushed. A virtual stigmergy is used for sharing the values and determining the winner(Task_stigmergy). All the robots other than the cagers bid a value of $||x_i - tau_i||$ for all the tasks currently present in the stigmergy. The gradient algorithms allows the smallest bidder to take up the task and excludes that robot form bidding further. Note that the number of tasks keep changing as the cagers keep updating the tasks untill the stop cage criteria is reached. In the process, every robot keeps a local copy of all the robots positions in each branch by extracting the values from the Task_stigmergy. This are noted down in the LB and RB lists, which are useful for various purposes which will be clear in later sections. The parameters K and the list Bidding_timer allow the robots wait before making a decision. This is because of the communication delays possible while the messages propagate through the topology. The robots bid their value for every Kth instance from the starting untill the bidding time is reahed. It is to be noted that the robots keep note of the bidding_timer for every task individually and a new timer for every task is opened as when it detects a new task in the Task_stigmergy. The robot with the lowest bid at the end of the bidding timer is alocated the task, from where it retireves the information about it's task and the branch. It uses this information to edge follow to the partiular task. Open succesful allocation of the task to the robot the robot closes the task, so that no other bot can bid for that task there after. It is possible that the gradient algorithm allocates a task to a robot which has already been awarded a task. To avaoid such assignments the gradient algorithm reopens the tasks for bidding again. The conflict manager algorithm also solves a few conflicts while the gradient algorithm is running in the bots.

1.2 Task update algorithm

Task update algorithm 4 is used by the robots on reaching a target to determine the new task location. Using the relative positional information of the previous neighbor in the current branch, the robot builds a vector to this neighbor. This vector is rotated 180° for the branch to grow in the opposite direction of the neighbor and this would be the next task location. The robot corrects this next task location based on the desired inter-robot distance and updates it in the virtual stigmergy. The other free robots on the knowledge of the availability of the new task bids for this task and the winner takes the task and grows the branch even further.

1.3 Cage Stopping algorithm

The cage stopping algorithm 3 is used by all the robots once they reach the their target and before spawning a new task for the current branch. Before implementing this algorithm they calculate the next task as explained in the Next task update algorithm. The cagers then retrieve the current opposite point from their opposite branch. They then check if the distance between these two points is greater than the stopping tolerance(d_T). If more than two robots, typically one from each brach come to a conclusion that the tolerance has reached they update a virtual stigmergy, which notifies all the other robots to switch to the path consensus state. For information about the mathematical proof about the stopping criteria the reader can refer the main paper.

Algorithm 1 Modified Gradient Algorithm to assign tasks sequentially during caging

```
procedure GRADIENT ALGO($\mathbb{T}$, $x_i$)
    for each $\tau_i \in \mathbb{T}$ do
        if Branch == Left then
            add(LB,$\tau_i$)
        else
            add(RB,$\tau_i$)
        end if
        if (ROLE==Cager) & (Task_Stigmergy.Get_id($\tau_i$)==$r_i$) & (Task_Stigmergy.Get_bid($\tau_j$)!=nil) then
            Task_Stigmergy.Put_bid($\tau_i$,"Reopen")
        end if
        if not(bidding_timer[$\tau_i$ ] % k) then
            $c_{r_i \tau_i} = ||x_i - \tau_j||$
            Current_bid= Task_Stigmergy.Get($\tau_i$)
            if Current_bid > $c_{r_i \tau_i}$ then
                Task_Stigmergy.Put($\tau_i$, $c_{r_i \tau_i}$)
            end if
        else
            bidding_timer[$\tau_i$] = bidding_timer[$\tau_i$] + 1
        end if
        if bidding_timer[$\tau_i$] == bidding_time then
            if (Task_Stigmergy.Get_Id($\tau_i$)==$r_i$)&(ROLE!=Cager) then
                $ROLE = Cager$
                if Branch == Left then
                    TASK = LB[1]
                else
                    TASK = RB[1]
                end if
                Task_Stigmergy.Put($\tau_i$,NIL)
            else
                Task_Stigmergy.Put_bid($\tau_i$,"Reopen")
            end if
        end if
    end for
end procedure
```

Algorithm 2 Conflict Manager for the Modified Gradient Algorithm

```
procedure CONFLICT MANGER($\tau_i$,local_copy,remote_copy)
    if remote_copy.task == local_copy.task then
        if remote_copy.data $\leq$ local_copy.data then
            return remote_copy
        else
            return local_copy
        end if
    else
        if local_copy.robot $<$ remote_copy.robot then
            if ($r_i$ != remote_copy.robot) then
                return local_copy
            else
                Task_Stigmergy.Put($\tau_i$.size+1,remote_copy.data)
                return local_copy
            end if
        else
            if ($r_i$ != local_copy.robot) then
                return remote_copy
            else
                Task_Stigmergy.Put($\tau_i$.size+1,local_copy.data)
                return remote_copy
            end if
        end if
    end if
end procedure
```

Algorithm 3 Stopping Criteria for caging

```
procedure CAGE STOP($\mathbb{T}_{next}$, $x_i$)
    if Branch == Left then
        oppbranch_prevpoint = RB[size_RB]
    else
        oppbranch_prevpoint = LB[size_LB]
    end if
    if $||\mathbb{T}_{next} - oppbranch\_prevpoint|| \leq d_T$ then
        Cagingstop_Stigmergy.Put($r_i$,"BARRIER")
    end if
end procedure
```

Algorithm 4 Task update Algorithm

```
1: procedure TASK UPDATE($x_i$)
2:     if Branch == Left then
3:         prevpoint = LB[size_LB - 1]
4:     else
5:         prevpoint = RB[size_RB - 1]
6:     end if
7:     prevpoint_vector = vec_sub(prevpoint,$x_i$)
8:     nexttask_vector = vec_rot(prevpoint_vector,180)
9:     nexttask = nexttask_vector + $I_d$
10: end procedure
```

Algorithm 5 Inter-cage distance correction

```
1: procedure INTER_CAGE_CORRECTION
2:     taxicab_dist=$||prevpoint\_vec||_1$
3:     if fabs(taxicab_dist - $I_d$) > $d_{tol}$ then
4:         $u_i = \perp x_o$
5:     else
6:         Target_reached=1
7:     end if
8: end procedure
```

Algorithm 6 Edge following procedure

```
1: procedure EDGE_FOLLOW
2:     closest_nei = get_closest_nei_pos()
3:     $u_i$ = ($\perp$ closest_nei) + $(I_d - ||closest\_nei||)\frac{closest\_nei}{||closest\_nei||}$
4: end procedure
```

2 Experiments

In this section, visualizations for the various cases are discussed. One random experiment from each of the 30 experiments of the irregular case for 3 shapes and one each from 25 robots(straight,zigzac and straight_rot), 50 robots(straight,zigzac and straight_rot) and 100 robots(straight and zigzac) are shown in detail. The starting positions of all the robots in the deployment cluster, the final caging positions and the final positions are shown for better understading of the system. The individual trajectories, shows the rotation detail and the movement history of the robot from the deployment cluster to the end point.

2.1 Irregular Shapes

Our algorithm is demonstrated to work for irregular objects. The algorithm was tested on 3 types of Irregular object in the shape of Clover, Box rotation and cloud as shown in figure 1. There was consitency in the maintainance of the inter robot distance becuase of the implementation of the inter robot correction distance. The consistency of the cage stop is also seen well with the irregular shaped objects. For details about the design parameter and the other performance factors, one can refer the main paper.

2.2 Regular Shapes

For the regular shapes, movement and caging both the algorithms were tested for 3 sizes and 3 benchmarking paths(figures 2,3 and 4) . It can be seen that the formation is quite well maintained for the straight and zigzac cases for all three sizes. Rest assured convergence of caging and convergence of the object to the end point is shown for a example form all the cases. For details about the design parameter and the other performance factors, one can refer the main paper.

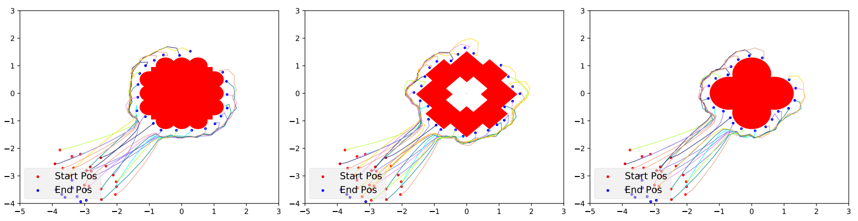

Fig. 1: Irregular shapes caging visualizations cloud shape(left), Box rotation shape(middle) and Clover shape(right)

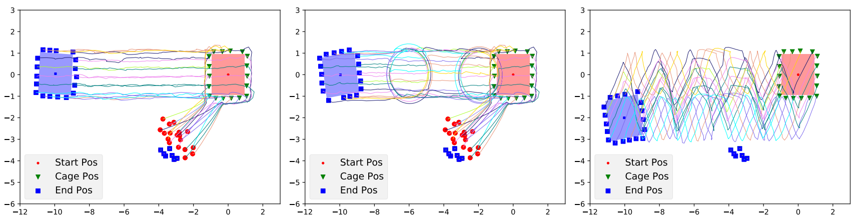

Fig. 2: caging and movement visualizations for straight(left), straight_rot(middle) and zigzac(right) for 25 robots case

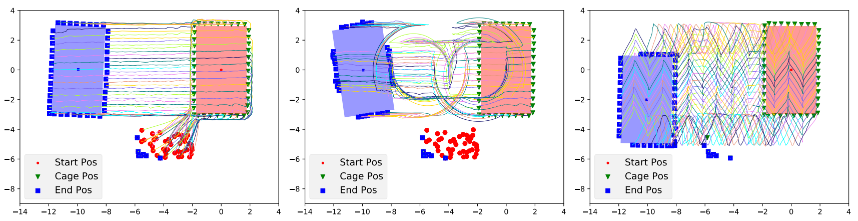

Fig. 3: caging and movement visualizations for straight(left), straight_rot(middle) and zigzac(right) for 50 robots case

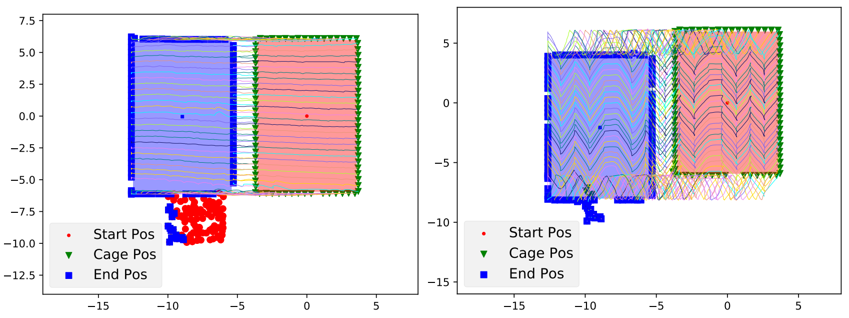

Fig. 4: caging and movement visualizations for straight(left) and zigzac(right) for 100 robots case