2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Adaptation to Robot Failures and Shape Change in Decentralized Construction

Seung-kook Yun and Daniela Rus

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, Massachusetts, USA
yunsk@mit.edu, rus@csail.mit.edu

***Abstract*— Our prior work [1] presented a decentralized algorithm for coordinating the construction of a truss structure out of multiple components. In this paper, we discuss adaptation in decentralized construction. We partition construction in two tasks, tool delivery and assembly. Each task is performed by a networked team of specialized robots. We analyze the performance of the algorithms using the balls into bins problem, and show their adaptation to failure of robots, dynamic constraints, multiple types of elements and reconfiguration. The algorithms can be used for general types of source elements.**

I. Introduction

This paper discusses an adaptive decentralized approach to construction with robot teams. We wish to develop cooperative robot systems for complex assembly tasks in which parts of different types are delivered at the location where they are needed for the current assembly operation and assembled. We abstract this process in the form of tool delivery robots and assembly robots. In our previous work [1], we presented a decentralized control algorithm for coordinating part delivery and assembly for truss-like structures that consist of links and connectors, and proved that the decentralized controller is stable. In this paper we extend the work to construction beyond trusses where the target consists of any given number of parts from a set of part types. We give an analysis of this algorithm and prove that the algorithm is adaptive to (1) the failure of changing numbers of assembly robots and delivery robots, (2) dynamic constraints such as order of construction, and (3) changes in the geometry of the target structure during assembly. Finally, we demonstrate the performance of the algorithm in simulation.

A. Related work

This work is based on distributed coverage and robotic construction. Distributed coverage using Voronoi tessellation was proposed in [2] for multi-robot system. The same optimization criteria were used in a distributed coverage controller for real-time tracking by Pimenta et al. [3]. Schwager [4] used adaptive coverage control in which networked robots learn a sensory function. Pavone et al. [5] have been working on equitable partitioning by the power diagram.

SM^2 is a truss-walking inspection robot developed for space station trusses [6]. Skyworker performed truss-like assembly tasks [7]. Werfel et al. [8] introduced a 3D construction algorithm for modular blocks. Our previous work on robotic construction includes Shady3D [9], [10], [11] utilizing a passive bar. We also proposed a optimal algorithm for reconfiguration of a given truss structure to a target structure [12].

II. Coordinated Robotic Construction

We are given a team of robots, n of which are specialized as assembling robots and the rest are specialized as part delivering robots in Euclidean space $Q \subset \mathbb{R}^N (N = 2, 3)$. Let N_d be the number of delivery robots and N_a be the number of assembly robots. The robots can communicate locally with other robots within their communication range. The robots are given a target shape represented as a target density function $\phi_t : Q \to \mathbb{R}$. ϕ_t represents the goal shape geometry by specifying the intended density of construction material in space. For example, in Figure 1 the yellow region has high density (many materials) while the white region has low density. If the components can be built *independently* and an assembling robot is capable of assembling all of them, ϕ_t is linearly superposed as

$$\phi_t = \sum_{u=1}^{z} \beta_u \psi_u, \tag{1}$$

where z is the number of the component that can be assembled by an assembling robot, and β_u is a constant representing importance of the u^{th} component. Importance can measure time required to assemble the piece, time till the piece is needed in the assemble, etc.

Without loss of generality, we will focus the examples on truss structures built with two types of components: connectors and links in order to simplify exposition and figures. To represent truss structures, ϕ_t is defined point-wise on the grid that corresponds to the truss. The point density is proportional to the number of possible truss connection at the point. We wish to develop a decentralized algorithm that coordinates the robot team to deliver parts so that the goal assembly can be completed with maximum parallelism. We assume that the robots move *freely* in an Euclidean space (2D and 3D).

Algorithm 1 shows the main flow of construction in a *centralized* view. In the first phase, assembling robots spread in a convex and bounded target area Q which includes the target structure. They find placements using a distributed coverage controller which assigns to each robot areas of the target structure that have approximately the same assembly

complexity. In the second phase the delivering robots move back and forth to carry source components to the assembling robots. They deliver their components to the assembling robot with maximum *demanding mass*. The demanding mass is defined as the amount of a source component required for an assembling robot to complete its substructure. In this paper, we restrict the source components include two types: unit-length truss elements and connectors. However, the algorithm is general and can support any number of different assembly components. After an assembling robot obtains a component from a delivering robot, it determines the optimal placement for this component in the overall assembly and moves there to assemble the component. The assembly phase continues until there is no source component left or the assembly structure is complete.

Algorithm 1 Construction Algorithm

1: Deploy the assembling robots in Q
2: Place the assembling robots at optimal task locations in Q (Section II-A)
3: **repeat**
4: **delivering robots:** carry source components to the assembling robots
5: **assembling robots:** assemble the delivered components
6: **until** task completed *or* out of parts

A. Task Allocation by Coverage with Equal-mass Partitions

Suppose n assembling robots cover region Q with the configuration $\{\mathbf{p}_1, ..., \mathbf{p}_n\}$, where $\mathbf{p}_i$ is the position vector of the i^{th} robot. Given a point $\mathbf{q}$ in Q, the nearest robot to $\mathbf{q}$ will execute the assembly task at $\mathbf{q}$. Each robot is allocated the assembly task that includes its Voronoi partition V_i in Q.

$$V_i = \{\mathbf{q} \in Q | \, \|\mathbf{q} - \mathbf{p}_i\| \leq \|\mathbf{q} - \mathbf{p}_j\|, \forall j \neq i\} \tag{2}$$

The target density function ϕ_t is the density of truss elements, and it is fixed during the construction phase. Given V_i, we define its mass property as the integral of the target density function in the area.

$$M_{V_i} = \int_{V_i} \phi_t(\mathbf{q}) d\mathbf{q} \tag{3}$$

Each robot follows its own local controller, designed to achieve a global distribution of robots so that each robot to have the same amount of assembly work. We call this *equal-mass partitioning*. Note that Voronoi tessellation evolves as robots are controlled. The cost function can be modeled as the product of all the masses.

$$\mathcal{H} = \mathcal{H}_0 - \prod_{i=1}^{n} M_{V_i}, \tag{4}$$

where $\mathcal{H}_0$ is a constant and the bound of the product term as:

$$\mathcal{H}_0 = \left(\frac{1}{n} \sum_{i=1}^{n} M_{V_i} \right)^n = \left(\frac{1}{n} \int_Q \phi_t(\mathbf{q}) d\mathbf{q} \right)^n. \tag{5}$$

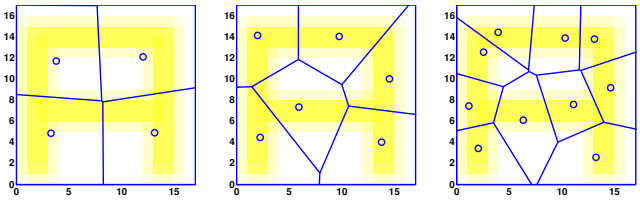


Fig. 1. Density function for an A-shaped bridge and coverage by the equal-mass partitioning. The blue circles are assembling robots. Yellow regions have dense ϕ_t.

$\mathcal{N}_i$ is a set of neighbor robots of the robot i. Minimizing this cost function leads to equal-mass partitioning, because the product of the masses is bounded by the sum which is constant. Therefore the prefect equal-mass partitioning makes the cost function zero. Using the cost function in (4), we have developed a decentralized controller that guarantees $\mathcal{H}$ converges to a local minimum [1].

1) Controller with Guaranteed Convergence: We wish for the controller to continuously decrease the cost function: $\dot{\mathcal{H}} \leq 0, t > 0$. Let $\boldsymbol{J}_i$ denote the part of the partial derivative term $\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i}$ which is related with the set $\{i, \mathcal{N}_i\}$.

$$\boldsymbol{J}_i = \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k \in \{i,\mathcal{N}_i\}, k \neq j}^{n} M_{V_k} \tag{6}$$

Given a velocity control for each robot, the decentralized controller that achieves task allocation is given by the control law

$$\dot{\mathbf{p}}_i = k \frac{\boldsymbol{J}_i}{\|\boldsymbol{J}_i\|^2 + \lambda^2}, \tag{7}$$

where k is a positive control gain and λ is a constant to stabilize the controller even around singularities where $\|\boldsymbol{J}_i\|^2 = 0$.

Figure 1 shows the resultant Voronoi regions of an A-shaped bridge, obtained from implementing the controller with 4,6 and 10 robots. You can see each robot has approximately the same area of the yellow region.

B. Delivery and Assembly Algorithms

Once the assembling robots are in place according to the equal-mass partitioning controller, construction may begin. State machines drive the delivering robots and the assembling robots. During construction we wish to distribute the source components (truss elements and connectors) to the assembling robots in a balanced way. Global balance defined as balance of delivery to all the assembling robots is asymptotically achieved by a probabilistic target selection of delivering robots that uses ϕ_t as a probability density function. For local balance defined for only neighboring robots, the delivering robots are driven by the gradient of demanding mass defined as the remaining structure to be assembled by the robot. Robots with more work left to do get parts before robots with less work left. Each assembling robot waits for a new truss element or connector and assembles it to the most demanding location in *its Voronoi region*. Therefore, construction is purely driven by the density function regardless

of the amount of the source components. We ensure that all the processes of the controllers work in a distributed way and each robot needs to communicate only with neighbors. Details of the control algorithms are explained in [1].

III. Analysis of the Algorithms

We now build on the algorithms and analyze the performance of the algorithms with respect to balance among the substructures and completion time. Simulation data is obtained from building the A-shaped bridge in Figure 1.

A. Balance of the sub-structures

Our goal is an algorithm that ensures the subassembly tasks proceed and get completed at the same time. This ensures that the overall construction is well-parallelized and there is no unnecessary waiting for subassembly completion. Let us assume the equal-mass partitioning is successful so that each assembling robot has the same amount of the target structure. The probabilistic deployment of the delivery algorithm leads to the traditional problem *ball-into-bins* where we throw m balls into n bins one by one with uniformly distributed probability of placing a ball at a bin. This problem is also known as *online load balancing* for distributed computation, where n servers are supposed to match m requests. In both cases, the question is what is the maximum number of balls (requests) in any bin (server).

Theorem 1: With only probabilistic deployment, the maximum deviation of delivery from the mean ($\frac{m}{n}$) is bounded by $\sqrt{2\frac{m}{n}\log n}$ with high probability.

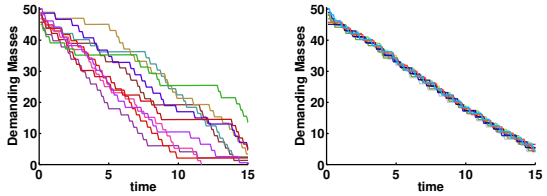
Proof: In case $m \gg n$ as ours, with high probability (normally $\gg 1 - \frac{1}{n}$), the maximum number of balls [13] is smaller than

$$\frac{m}{n} + \sqrt{2\frac{m}{n}\log n}. \tag{8}$$

Since the mean number of balls is $\frac{m}{n}$, The maximum deviation from the mean is bounded by $\sqrt{2\frac{m}{n}\log n}$. ■

Figure 2(a) shows the demanding masses simulated from an example where 10 assembling robots and 10 delivery robots are used and only the probabilistic deployment is implemented. We can see the demanding masses spread out as construction goes on. Figure 3 shows maximum deviation of the demanding mass from the mean and the theoretical bound. The mean of the maximum deviation and the error bars are obtained from 10 simulations.

Algorithm 1 allows a delivering robot to find the assembling robot with the maximum demanding mass after the probabilistic deployment, and that dramatically improves balance as shown in Figure 2(b) and Figure 3. During construction, all the demanding masses are within a range of a single truss element, which implies perfect balance. This local search can be understood as picking multiple bins first and putting a ball at the bin with the minimum number of balls. It is well known in the balls into bins problem that the maximum load can be greatly reduced if we can choose two bins at random rather than just one bin [14]. In the proposed algorithm, a delivering robot chooses where to place a source component among neighboring robots of the robot that is



(a) (b)

Fig. 2. Demanding masses resulted from (a) probabilistic deployment only (b) proposed algorithm. 10 assembling robots and 10 delivering robots are used.

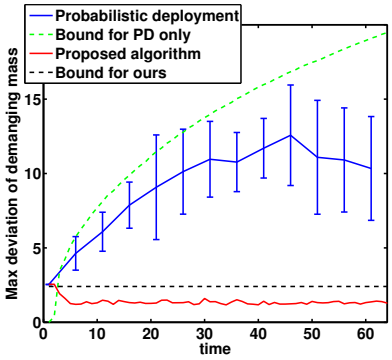


Fig. 3. Average demanding mass and error bars from the probabilistic deployment and Algorithm 1. Green dotted line is the theoretical bound when only probabilistic deployment is used for delivery. Red solid line is the simulation result from the proposed algorithm, and black dotted line is the theoretical bound for the algorithm. The bound makes more sense when enough time has passed, since the bound is valid for $m \gg n$.

picked by the probabilistic deployment. This is equivalent to having the robot choose multiple assembling robots on a graph.

Theorem 2: Algorithm 1 yields the maximum deviation bounded by $\frac{\log\log n}{\log 2}$ with high probability.

Proof: The maximum load decreases into [15]

$$\frac{\log\log n}{\log d} + \frac{m}{n}, \tag{9}$$

where d is a number of bins we can choose. [1] Since we do not know how many neighbor robots there are, we use a conservative bound with $d = 2$. ■

The black dotted line is the bound with $\frac{\log\log n}{\log 2}$. Note that the maximum deviation is not dependent on m.

B. Construction time and Travel distance

We conduct an empirical analysis of the construction algorithms, by testing several combinations of parameters. There are two major parameters that affect the total construction time: velocity of the robot and assembly time required for an assembling robot to assembling a part. If the assembly time is much larger than the reciprocal of the velocity, construction time will be dominated by the assembly time. If the assembly

[1] To qualify the equation, the graph should be regular with degree n^ϵ where ϵ is not too small [15]. In our case, we can not guarantee a degree of the graph that equal-mass partitioning would build. However, if the target structure is fully connected, ϵ should be at least greater than 2.

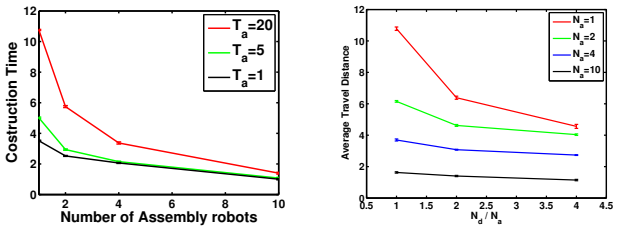


Fig. 4. (a)Total construction time. (b)Average travel distance of the delivery robots. Error bars are plotted together. The construction time is down-scaled by 1000 for a better view.

time is very short, the total time will be a function of the traveling distances of the robots. We evaluate the algorithms with the following sets of parameters: $N_a \in \{1, 2, 4, 10\}$, $N_d/N_a \in \{1, 2, 4\}$, $T_a \in \{1, 5, 20\}$. T_a is the assembly time.

When the assembly time is large, the construction time decreases proportional to the number of the assembly robots, as shown in Figure 4(a). Therefore the control algorithms yield good parallelism when a robot has a large assembly time. If the assembly time is small, we may modify the criteria for a delivery robot to select an assembly robot by incorporating expected traveling distance. This will be considered in our future work.

The average travel distance of the delivery robots is examined in Figure 4(b). Increasing the number of delivery robots is more effective when the number of assembly robots is small. However, too many delivery robots do not reduce the average distance and the construction time much (the slopes become flat as the number increases.) Careful choice of the robot numbers will yield the an appropriate tradeoff between robot numbers and construction time. This will be investigated in the future.

IV. Adaptation

The construction algorithms in Section III are adaptive to several cases such as failure of robots, construction with dynamic constraints, multiple types of source elements and reconfiguration between two structures. We next discuss each case.

A. Robustness to failure of robots

Assembling robots are critical since each assembling robot covers a unique region. Failures of the assembling robots can be tolerated by executing the subassembly equal-mass partitioning continuously as a background process. When a robot fails, its remaining subassembly task will get re-assigned and all the other assembly loads re-balanced. We assume a failed robot disappears with an element if it is carrying any. Algorithm 2 shows the main control loop for assembling robots with continuous equal-mass partitioning. ρ describes a density function for currently built structure, and Φ^c is a set of required connectors for the current structure. The assembling robots reconstruct the Voronoi regions when the surrounding network of the robots has changed. Since assembling robots move during construction, we introduce

Algorithm 2 Assembly with Equal-mass Partitioning

1: **repeat**
2: assemble the delivered components
3: move to $\hat{\mathbf{p}}_i$ by Equation 7
4: update V_i, G, ρ, Φ^c
5: **until** task completed

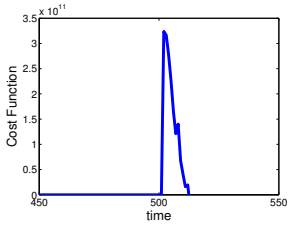


Fig. 6. Cost function of the simulation in Figure 5. At time 500, the cost rises up because of failure, however settles down by the equal-mass partitioning controller.

the virtual center of the Voronoi region $\hat{\mathbf{p}}_i$ and move it instead of a robot position, and reconstruct V_i around $\hat{\mathbf{p}}_i$. The assembling robots also need to update the parameters such as the graph of the built structure and demanding mass for truss and connectors. We assume that a robot can detect failure of its neighbor.

Theorem 3: Continuous coverage during construction compensates for the failure of the assembling robots

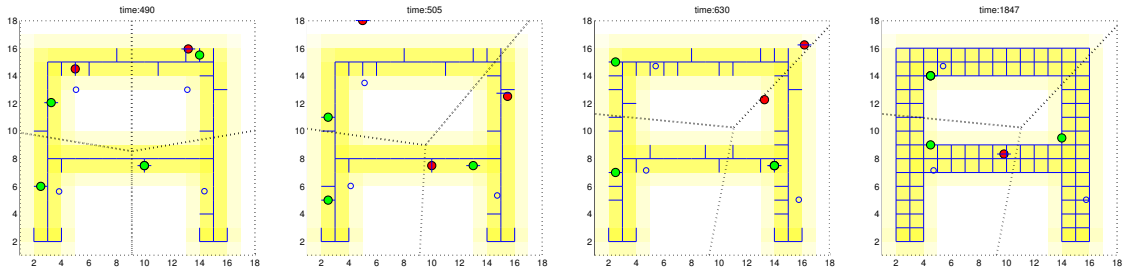
Proof: The coverage controller guarantees decay of the cost function $\mathcal{H}$ regardless of the number of neighbors. Therefore, if a robot fails, $\mathcal{H}$ will decrease to a local optimum with the changed configuration, as long as there are the remaining assembling robots. ■

Figure 5 shows a snapshot from a simulation with a failed robot. The robot in the upper right Voronoi region fails during construction as Figure 5(b), and the neighboring robots adapt their Voronoi regions to fill the region of the failed robot while continuing construction. Since the coverage control requires a significant amount of computation, the robots end it when the cost function settles down as shown in Figure 6.

Failure of delivering robots is not critical in our approach, because the system is transparent to that. Only the completion time would increase, since we have less number of delivering robots after the failure.

B. Dynamic constraint: construction in order

Territorial construction is subject to gravity constraints which in turn imposes ordering constructions on assembly job. For example, a 3D structure should be built from the ground up. We extend our algorithm to incorporate this type of constraint in terms of connectivity. Given ϕ_t, we ensure connectivity by revealing only the part of ϕ_t that is connected to the current structure. Equal-mass partitioning and the computation of the demanding mass are done with the revealed part of ϕ_t, which is now a time-varying function. We model this revealed part of ϕ_t as a time-varying target density function φ_t. The assembling robots perform equal-mass partitioning based on φ_t.



(a) (b) (c) (d)

Fig. 5. 4 assembling robots are constructing the bridge and one of them fails at time 500. Green circles are assembling robots, and red ones are delivering robots. Blue hollow circles are the virtual center of V_i. After failure, the remaining robots reconfigure their Voronoi regions adaptively and finish the construction.

Algorithm 3 Update the density function φ_t during building a single truss element

$\mathbf{q}_1 \leftarrow$ a set of nodes incident to e_{opt} and $\notin R_c$
$\mathbf{q}_2 \leftarrow$ two nodes of e_{opt}
set $t = 0$
repeat
  $\dot{\varphi}_t(\mathbf{q}_1) = \frac{\phi_t(\mathbf{q}_1)}{T_a}$
  $\dot{\rho}(\mathbf{q}_2) = \frac{\Phi_0(\mathbf{q}_1)}{T_a}$
  update φ_t and ρ
until $t > T_a$
$R_c \leftarrow \mathbf{q}_1$

We update φ_t by Algorithm 3. Given the grid map Q, R_c is a set of nodes that are reachable, Φ_0 is a unit density for each node of a truss element, and T_a is an assembly time to finish assembling a truss element. When an assembling robot starts to build a truss element at an edge e_{opt}, it checks whether the adjacent nodes of e_{opt} are in R_c or not. For the nodes to be revealed $\mathbf{q}_1$, the density function increases by the rate $\frac{\phi_t}{T_a}$ till time T_a. Therefore, only the nodes connected to the current structure (R_c) are used in the current target density function φ_t. The next chosen edge e_{opt} must be connected to the current structure.

The coverage control follows Algorithm 2. We modify it to incorporate the time varying density function. Note that φ_t varies smoothly since $\dot{\varphi}_t$ is a constant.

Given the cost function $\mathcal{H}$ that is now a function of φ_t replacing ϕ_t is Equation 3, differentiating $\mathcal{H}$ yields

$$\dot{\mathcal{H}} = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} \dot{\mathbf{p}}_i + F_i \prod_{k \notin \{i, \mathcal{N}_i\}} M_{V_k} \right). \tag{10}$$

The new term F_i comes from the time varying density function, and can be computed as

$$F_i = - \prod_{j=\mathcal{N}_i} M_{V_j} \int_{V_i} \dot{\varphi}_t(\mathbf{q}, t) d\mathbf{q}, \tag{11}$$

where $\dot{\varphi}_t$ is given by Algorithm 3. If we set the velocity input as

$$\dot{\mathbf{p}}_i = \frac{\boldsymbol{J}_i}{\|\boldsymbol{J}_i\|^2 + \lambda^2} (k - F_i) \tag{12}$$

where

$$\boldsymbol{J}_i = \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k \in \{i,\mathcal{N}_i\}, k \neq j}^{n} M_{V_k}, \tag{13}$$

$\dot{\mathcal{H}}$ becomes

$$\dot{\mathcal{H}} = - \sum_{i=1}^{n} \frac{1}{\|\boldsymbol{J}_i\|^2 + \lambda^2} \left(k \|\boldsymbol{J}_i\|^2 + \lambda^2 F_i \right) \prod_{l \notin \{i,\mathcal{N}_i\}} M_{V_l}. \tag{14}$$

Theoretically, setting the gain k to a large value ensures $\dot{\mathcal{H}} \leq 0$ unless all $\boldsymbol{J}_i$ are zero. We conjecture that F_i also becomes zero if all $\boldsymbol{J}_i$ are zero, however, we have not proven this yet. In practice, a robot sets the gain k_i that guarantees a local derivative of the cost function $\dot{\mathcal{H}}_i \leq 0$, which is defined as

$$\dot{\mathcal{H}}_i = \sum_{j=i, j \in \mathcal{N}_i} \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_j} \dot{\mathbf{p}}_j + F_j \prod_{k \notin \{i,\mathcal{N}_i\}} M_{V_k} \right). \tag{15}$$

Theorem 4: F_i is bounded.

Proof: M_V and $\dot{\varphi}_t$ are bounded. Therefore, by Equation (11), F_i is bounded. ■

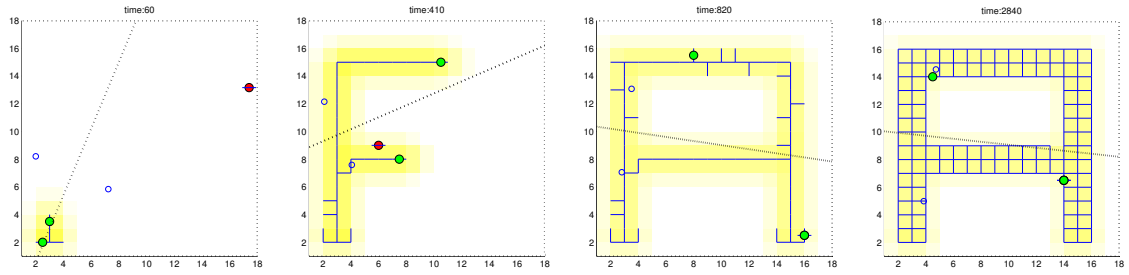
Theorem 5: The control input $\dot{\mathbf{p}}_i$ is bounded.

Proof: Because F_i and M_V are bounded, $\dot{\mathbf{p}}_i$ is bounded by Equation (12) ■

Figure 7 shows results from our implementation of the control algorithms with 2 assembling robots. The bridge is to be built from the lower left corner. Only the lower left part of the target density function is revealed as in Figure 7(a). The more the robots build, the more of ϕ_t is used until the entire target density function ϕ_t is revealed. As shown in Figure 8(a), the cost function is almost flat even though φ_t changes during construction, since the controller incorporate the time varying density function. We can see the masses of two robots are almost identical at all time during construction as in Figure 8(b).

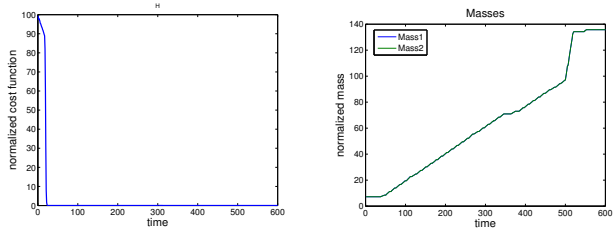
C. Reconfiguration

The goal structure might change after or during construction. We extend the construction algorithm to support adaptation to changing structure geometry during construction, in order to build a new goal structure from the current structure. Suppose a target structure ϕ_{t_1} has been built and a new target structure ϕ_{t_2} is given. Assuming the assembling robot is



(a) (b) (c) (d)

Fig. 7. Construction in order. 2 assembling robots and 2 delivering robots are building the bridge from the lower left corner.



(a) (b)

Fig. 8. Construction in order

capable of disassembly, Algorithm 4 shows how the original structure is reconfigured to the new structure. Here we set the target density function as difference between two structures $|\phi_{t_2} - \phi_{t_1}|$ for equal-mass partitioning, since disassembly also requires work of assembling robots. We assume cost for disassembly is the same as assembly. If they are different, we can generalize the target density function as:

$$\phi_t = (\phi_{t_2} - \phi_{t_1})_+ + \alpha(\phi_{t_1} - \phi_{t_2})_+, \tag{16}$$

where α is a workload ratio of disassembly to assembly and $(\cdot)_+$ represents positive only. From now on, we set $\alpha = 1$. The demanding mass is extended to two types: for assembly ($\Delta M^a_{V_i}$) and disassembly ($\Delta M^d_{V_i}$) , which are defined as

$$\Delta M^a_{V_i} = \int_{V_i} (\phi_{t_2}(\mathbf{q}) - \phi_{t_1}(\mathbf{q}))_+ d\mathbf{q} - \int_{V_i} \rho_a(\mathbf{q}) d\mathbf{q}, \tag{17}$$

$$\Delta M^d_{V_i} = \int_{V_i} (\phi_{t_1}(\mathbf{q}) - \phi_{t_2}(\mathbf{q}))_+ d\mathbf{q} - \int_{V_i} \rho_d(\mathbf{q}) d\mathbf{q}, \tag{18}$$

where ρ_a is the density function of the built structure and ρ_d is of the disassembled structure.

Algorithm 4 Reconfiguration Algorithm

1: Place the assembling robots by equal-mass partitioning with the density function $|\phi_{t_2} - \phi_{t_1}|$ in Q
2: **repeat**
3: **delivering robots:** carry source components from $(\phi_{t_1} - \phi_{t_2})_+$ to the assembling robots
4: **assembling robots:** assemble the delivered components in $(\phi_{t_2} - \phi_{t_1})_+$
5: **until** task completed *or* out of parts

1) Assembly Algorithm: The state machine used for the assembling robot in [1] is adjusted for reconfiguration. The robot has the following states:

- IDLE
- WAITING: waiting for a new component or request for a part
- MOVING_ASSEMBLY: moving to the optimal location to add the part
- ASSEMBLING: adding the component to the assembly
- MOVING_DISASSEMBLY: moving to the optimal location to detach the part
- DISASSEMBLING: removing the component and hand over it to a delivering robot

The last two states are added to the state machine in [1] for disassembly.

Algorithm 5 shows the details of the state machine for disassembly. The state machine for assembly is in [1]. When reconfiguration starts, an assembling robot initializes the parameters R, E, ρ_a, ρ_d and changes its state to WAITING. Recall that each robot has a local graph representation $G = (R, E)$ of the already built local substructure by itself and neighbors. If it receives a request for disassembly from a delivery robot, it finds the optimal location to remove a truss element in $(\phi_{t_1} - \phi_{t_2})_+$. The optimal location is chosen as an edge with the maximum demanding mass for disassembly. The robot moves to the location by setting the state to MOVING_DISASSEMBLY. In the MOVING_DISASSEMBLY state, an assembling robot moves to the target location $\mathbf{t}$ and changes the state to DISASSEMBLING when it arrives. Then it detaches the truss element and hand it over to the delivery robot. After disassembly, it updates the parameters such as R, E, ρ_d. The state goes back to WAITING.

2) Delivery Algorithm: Delivering robots also operate by an adjusted state machine from [1]. Each robot has the following states:

- IDLE
- ToSOURCE: moving to a picked point in $(\phi_{t_1} - \phi_{t_2})_+$
- ToTARGET: moving to a picked point in $(\phi_{t_2} - \phi_{t_1})_+$
- ToASSEMBLY: delivering the element to an assembling robot
- ToPICKUP: moving to get a new element from an assembling robot

Algorithm 5 Control Algorithm of assembling robots

STATE: IDLE
1: $R \leftarrow nodes \in \phi_{t_1}(V_i)$, $E \leftarrow edges \in \phi_{t_1}(V_i)$
2: $\rho_a(\mathbf{q}) = 0, \rho_d(\mathbf{q}) = 0$
3: *state*=WAITING
STATE: WAITING
4: **if** receive a request for disassembly **then**
5: $e =$findOptimalEdge$(R, E, (\phi_{t_1} - \phi_{t_2})_+, \rho_d)$
6: $\mathbf{t} = \mathbf{q}_{(node_1(e)+node_2(e))/2}$
7: *state*=MOVING_DISASSEMBLY
8: **end if**
STATE: MOVING_DISASSEMBLY
9: **if** reached $\mathbf{t}$ **then**
10: *state*=DISASSEMBLING
11: **else**
12: move to $\mathbf{t}$
13: **end if**
STATE: DISASSEMBLING
14: disassemble the material
15: update $\rho_d(e)$
16: $E \leftarrow E - e$
17: $R \leftarrow R - \{node_1(e), node_2(e)\}$
18: hand over the material to the delivery robot
19: *state*=WAITING

- PICKING: getting the element from the assembling robot

The last two states are for disassembly.

Algorithm 6 describes the details of the state machine. Instead of obtaining a source component from a source cache as in [1], a delivering robot gets it from the redundant structure $(\phi_{t_1} - \phi_{t_2})_+$ and carries it to the unfilled structure $(\phi_{t_2} - \phi_{t_1})_+$. Given an initially empty state, a delivering robot changes its state to ToSOURCE and picks a possible source location with respect to the probability density function $(\phi_{t_1} - \phi_{t_2})_+$. This probabilistic choice has already been used for finding an assembly location in [1], and we use the same method to pick a source component here. The state ToSOURCE ends when the robot reaches the chosen location and switches to ToPICKUP. In the state ToPICKUP, the robot figures out an assembly robot with the maximum demanding mass that is a sum of $\Delta M^a_{V_k} + \Delta M^d_{V_k}$. To ensure there is a source component to be disassembled, the assembly robot should have positive demanding mass for disassembly ($\Delta M^d_{V_k}$.) If the assembling robot has the state WAITING, then it requests disassembly and moves to the robot, switching the state to PICKING. The delivery robots waits for the assembling robot to finish disassembly and receives the new truss element, changing the state to ToTARGET. The assembly procedure for the state ToTARGET and ToASSEMBLY has been explained in [1].

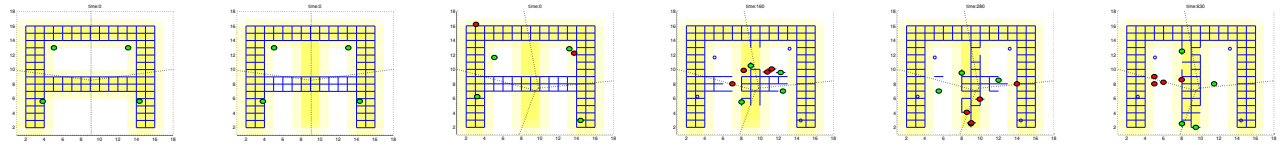
3) Implementation: Figure 9 shows snapshots of reconfiguration from an A-shaped bridge (Figure 9(a)) to an M-shape (Figure 9(b)). 4 assembling robots and 4 delivery robots are deployed. We can see the density function $|\phi_{t_2} - \phi_{t_1}|$ for equal-mass partitioning has cross-like shape (the yellow region without the truss and the truss outside the yellow region in Figure 9(b).) The partitioning results in new Voronoi regions as in Figure 9(c), and the delivering robots carry a truss element from redundant truss to the yellow region that is not filled by the truss yet.

Algorithm 6 Control Algorithm of delivering robots

STATE: IDLE
1: *state* = ToSOURCE
2: $\mathbf{t} \sim (\phi_{t_1} - \phi_{t_2})_+$
STATE: ToSOURCE
3: **if** reached $\mathbf{t}$ **then**
4: *state* = ToPICKUP
5: **else**
6: move to $\mathbf{t}$
7: **end if**
STATE: ToPICKUP
8: communicate with robot r_i s.t. $\mathbf{q} \in V_i$
9: $deliveryID = \text{argmax}_{(k=i,j\in\mathcal{N}_i),\Delta M^d_{V_k}>0} \Delta M^a_{V_k} + \Delta M^d_{V_k}$
10: **if** r_i = WAITING **then**
11: send a disassembly request to r_i
12: *state* = PICKING
13: $\mathbf{t} = \mathbf{p}_{deliveryID}$
14: **end if**
STATE: PICKING
15: **if** reached $\mathbf{t}$ and get a truss element **then**
16: *state* = ToTARGET
17: $\mathbf{t} \sim (\phi_{t_2} - \phi_{t_1})_+$
18: **else**
19: move to $\mathbf{t}$
20: **end if**
STATE: ToTARGET
21: **if** reached $\mathbf{t}$ **then**
22: *state*=ToASSEMBLY
23: **else**
24: move to $\mathbf{t}$
25: **end if**
STATE: ToASSEMBLY
26: communicate with robot r_i s.t. $\mathbf{q} \in V_i$
27: $deliveryID = \text{argmax}_{(k=i,j\in\mathcal{N}_i),\Delta M^a_{V_k}>0} \Delta M^a_{V_k} + \Delta M^d_{V_k}$
28: $\mathbf{t} = p_{deliveryID}$
29: **if** reached $\mathbf{t}$ & *state* of r_i = WAITING **then**
30: pass the material
31: *state* = ToSOURCE
32: **else**
33: move to $\mathbf{t}$
34: **end if**

D. Multiple types of source components

Figure 10 shows snapshots of the simulation of building the A-shaped bridge with two types of truss elements: *side* and *diagonal*. The density function is a simple sum of that for side and that for diagonal, since we assume assembling



(a) (b) (c) (d) (e) (f)

Fig. 9. Reconfiguration from the A-shaped bridge to the M-shaped bridge. 4 assembling robots and 4 delivering robots are used. (a) Completion of building the A-shaped bridge (b) New density function for the M-shaped bridge (c) Equal-mass partitioning for difference between the density functions (d-f) Reconfiguration

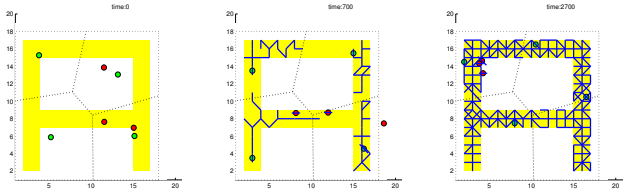


Fig. 10. A-shaped bridge with two types of truss elements. 4 assembling robots and 4 delivery robots are building the structure.

times for them are the same.

In the future, we will consider the case source components have dependency on each other so that they have to be built in some order.

V. Conclusion

In this paper, we review the decentralized control algorithms for coordinated construction of a truss structure, and extend them to be adaptive for various situations.

We show the probabilistic delivery algorithm is an instance of a classic problem: balls into bins. Analysis leads to theoretical bounds for unbalance among the sub-structures that are empirically proven in simulation. Given the assumption that equal-mass partitioning has found the global optimum, the local search algorithm reduces the bound from $\sqrt{2\frac{m}{n}\log n}$ to $\frac{\log\log n}{\log d}$.

Based on the proposed approach, the algorithms are adaptive for several cases. For failure of robots, the convergence property is not affected by failure of delivering robots, and keeping equal-mass partitioning makes the system robust to failure of assembling robots. Construction with dynamic constraints is possible by incorporating the time-varying density function and corresponding controllers which is slightly modified from the original controller. Non-dependent source elements can be used by superposing density functions for the elements. Reconfiguration between two structures are implemented by substituting the target density function for difference between target density functions of two structures.

Our current work is focused on the development of hardware implementation and a decentralized construction algorithm with an abstract goal.

VI. Acknowledgements

This project has been supported in part by The Boeing Company, the U.S. National Science Foundation, NSF grant numbers IIS-0426838, Emerging Frontiers in Research and Innovation (EFRI) grant #0735953, MURI SMARTS grant #N0014-09-1051, MURI ANTIDOTE grant #138802, and MURI SWARMS grant #544252. Seung-kook Yun is supported in part by Samsung Scholarship. We are grateful for this support.

References

[1] S. kook Yun, M. Schwager, and D. Rus, "Coordinating construction of truss structures using distributed equal-mass partitioning," in *Proc. of the 14th International Symposium on Robotics Research*, Lucern, Switzerland, August 2009.

[2] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," vol. 20, no. 2, pp. 243–255, 2004.

[3] L. C. A. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. C. Mesquita, and G. A. S. Pereira, "Simultaneous coverage and tracking (scat) of moving targets with robot networks," in *Proceedings of the Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, Mexico, December 2008.

[4] M. Schwager, D. Rus, and J.-J. E. Slotine, "Decentralized, adaptive control for coverage with networked robots," *International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, March 2009.

[5] M. Pavone, E. Frazzoli, and F. Bullo, "Distributed algorithms for equitable partitioning policies: Theory and applications," in *IEEE Conference on Decision and Control*, Cancun, Mexico, Dec 2008.

[6] M. Nechyba and Y. Xu, "Human-robot cooperation in space: SM^2 for new spacestation structure," *Robotics & Automation Magazine, IEEE*, vol. 2, no. 4, pp. 4–11, 1995.

[7] S. Skaff, P. Staritz, and W. Whittaker, "Skyworker: Robotics for space assembly, inspection and maintenance," *Space Studies Institute Conference*, 2001.

[8] J. Werfel and R. Nagpal, "International journal of robotics research," *Three-dimensional construction with mobile robots and modular blocks*, vol. 3-4, no. 27, pp. 463–479, 2008.

[9] S. kook Yun and D. Rus, "Optimal distributed planning for self assembly of modular manipulators," in *Proc. of IEEE/RSJ IEEE International Conference on Intelligent Robots and Systems*, Nice, France, Sep 2008, pp. 1346–1352.

[10] S. kook Yun and D. Rus, "Self assembly of modular manipulators with active and passive modules," in *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation*, May 2008, pp. 1477–1482.

[11] Carrick Detweiler, Marsette Vona, Yeoreum Yoon, Seung-kook Yun, and Daniela Rus, "Self-assembling mobile linkages," *IEEE Robotics and Automation Magazine*, vol. 14(4), pp. 45–55, 2007.

[12] S. kook Yun, D. A. Hjelle, H. Lipson, and D. Rus, "Planning the reconfiguration of grounded truss structures with truss climbing robots that carry truss elements," in *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009.

[13] M. Raab and A. Steger, "Balls into bins - a simple and tight analysis," 1998.

[14] M. D. Mitzenmacher, M. D. Mitzenmacher, and M. D. Mitzenmacher, "The power of two choices in randomized load balancing," IEEE Transactions on Parallel and Distributed Systems, Tech. Rep., 1996.

[15] K. Kenthapadi and R. Panigrahy, "Balanced allocation on graphs," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. New York, NY, USA: ACM, 2006, pp. 434–443.