Constraint-Aware Coordinated Construction of Generic Structures

David Stein, T. Ryan Schoen, Daniela Rus

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, Massachusetts, USA
{stein,rschoen,rus}@csail.mit.edu

***Abstract*— This paper presents a constraint-aware decentralized approach to construction with teams of robots. We present an extension to existing work on a distributed controller for robotic construction of simple structures. Our previous work described a set of adaptive algorithms for constructing truss structures given a target geometry using continuous and graph-based equal-mass partitioning [1], [2]. Using this work as a foundation, we present an algorithm which performs construction tasks and conforms to physical constraints while considering those constraints to parallelize tasks. This is accomplished by defining a mass function which reflects the priority of part placement and prevents physically impossible states. This mass function generates a set of pointmasses in $\mathbb{R}^n$, and we present a novel algorithm for finding a locally optimal, equal-mass, convex tessellation of such a set.**

I. Introduction

This paper presents a constraint-aware decentralized approach to construction with teams of robots. We aim to create complex structures using heterogeneous sets of robots and parts. We abstract this process using tool delivery and assembly robots; delivery robots pick up parts and carry them from some part source into the construction site where assembly robots perform actuation tasks to add the parts to the structure. In our previous work we presented a set of adaptive algorithms for constructing truss structures given a target geometry, and demonstrated their feasibility experimentally [3], [4]. This paper extends that work in two ways: first, by presenting a novel partitioning algorithm that uses a detailed description of the target structure rather than a target geometry to distribute tasks among robots (section III); second, by introducing a distributed algorithm for delivering parts in order to conform to physical constraints such as the stability of the structure and the ability of robots to reach areas in the structure (section IV).

If we imagine the construction of a building, there is a strong natural set of constraints on the build order. These fall into two major categories: (1) physical constraints require supporting structures be set up first: one must lay a foundation before a wall, and the first floor before the second; (2) reachability constraints require that construction not block pending work: plumbing and electrical is installed between studs before drywall encloses the space, though neither rely on each other for support. The work we present in this paper addresses the problem of representing and conforming to both of these classes of constraints, and determining ways to maximize and automate parallelism.

A. Related Work

Our work builds on prior research on robotic construction and distributed coverage. A simple distributed 3D construction algorithm is described in [16], while [6] describes a 3D construction algorithm for modular blocks in a distributed setting. Stochastic algorithms for robotic construction with dependency on raw materials are analyzed in [7]. Our previous work on robotic construction includes Shady3D [9] utilizing a passive bar and an optimal algorithm for reconfiguration of a given truss structure to a target structure [8], and experiments in building truss structures [4]. Using Voronoi partitions to deploy robots for coverage was originally proposed in [12] and has been extended several times since then for tasks such as adaptive coverage [15] and equitable partitioning [14]. Our most recent work extends the idea of equitable partitioning and combines it with coordinated construction of truss structures [2], locational optimization [1], and adaptation to failure and shape change [3].

To perform equal-mass partitioning, our approach utilizes previous work on computation using barycentric coordinates [17] and convex hulls [13].

II. Overview of Construction Algorithm

We are given a team of robots, a blueprint of a desired structure, and a construction region Q. A subset of n of the robots are specialized as assembly robots and the rest are specialized as delivery robots. The blueprint describes the location, type, and physical requirements for stability of each object ("part") in the structure. The robots are given a local section of a blueprint, and have full knowledge of the progress of the construction of the target structure in the area surrounding them and their neighbors. We describe an algorithm that coordinates the construction of a given structure while maximizing parallelism across assembly robots and conforming to the physical constraints of the structure. The algorithm is guaranteed to construct the structure in an order that is feasible in that it does not prevent any sub-assembly from being completed.

This problem formation differs from previous work in coordinated construction ([3], [2]) in that we introduce knowledge of physical constraints on assembly and delivery. We consider two new constraints: the requirement that each part must be capable of staying in place once set (physical dependency), and the requirement that a delivery robot must be able to reach both the source and assembly robots at all times (reachability).

We represent these constraints in the blueprint by defining parts as vertices connected on two graphs: a physical dependency graph $G_p(V, E_p)$, and a reachability graph $G_r(V, E_r)$.

We define G_p by placing a directed edge $(v_i, v_j) \in E_p$ between any v_i and v_j where v_j cannot be stably placed unless v_i already has been, regardless of the state of any other part. We assume that any part is placeable iff all of its parents in G_p have been placed.

We define G_r by placing a directed edge $(v_i, v_j) \in E_p$ between any v_i and v_j where a robot at the location of v_j could access v_i, regardless of the state of other parts in the systems.

Without loss of generality and for ease of exposition, we assume that there is some constant upper bound c on the maximal degree of any vertex in V.

Algorithm 1 outlines the construction algorithm from a global perspective. The three primary functions performed by the robots are partitioning, delivery, and assembly. Assembly robots are deployed into Q, and the partitioning controller (Section III) ensures that each assembly robot i has some partition of $\mathcal{P}_i \subset V$ that is spatially compact and that each partition has roughly the same amount of work assigned to it at all times. Delivery robots constantly deliver parts from some source to the assembly robots, using a priority function to determine the best place to deliver each part within a local region (Section IV).

Algorithm 1 Construction Algorithm

```
1: Randomly deploy assembly robots in $Q$
2: **repeat**
3:     **delivering robots:** deliver source components to assembling robots
4:     **assembling robots:** assemble the delivered components while constantly updating partitions
5: **until** task completed *or* out of parts
```

III. Equal Mass Partitioning using a Discrete Approach

In our problem formulation we represent each part in the target structure as a point, which is reasonable given the discrete nature of parts. We define the *demanding mass* of a part as a measure of its priority in placement order, where the mass of a part is 0 if a part is unplaceable or already placed and positive otherwise (this is discussed further in section IV). By partitioning based on this mass function, we can allocate roughly the same amount of reachable, actionable work to each robot. We repeat this algorithm continuously during runtime to maintain an equitable partitioning of Q as masses change dynamically while the structure is built.

A trade-off of the significant increase in fidelity we get by updating our model from a geometry to a blueprint is a change in the nature of the density of the Q. The density of Q is used by most coverage algorithms, including canonical Lloyd algorithms for equipartitioning, to perform gradient descent to converge to equal-mass partitions. Our blueprint forces the density of Q to be a dynamic summation of scaled Dirac delta functions, which has a gradient of either zero or infinity at all points, meaning we can not use the class of deployment algorithms that depend on Voronoi partitioning.

Vertex swap, which we present a potential solution to this problem in [1], works on a graph rather than in $\mathbb{R}^n$, and requires multi-hop communication. In order to use this algorithm, we need to define a graph that connects the set of positive mass points. If we create a relatively sparse graph we introduce unnecessary assumptions which limit which points can be in the same partition. If we create a well-connected graph we introduce the assumption of excessively large communication radii as neighbors are defined by edges in the graph rather than L^p distance. We have developed a equipartitioning algorithm that does not require a graph connecting points, uses only local communication, and has lower complexity than vertex swap.

We identify partitions that are spatially compact and approximately equal mass, but as stated above Voronoi partitioning and vertex swap are not viable options. The problem of partitioning a set of point masses in $\mathbb{R}^n$ into non-intersecting, convex, equal-mass partitions is NP-hard, even in $\mathbb{R}^2$. We present the *hull vertex swap* algorithm (Algorithm 2), an efficient distributed method for approximating equal-mass partitioning using only single hop communication.

Hull vertex swap converges to a convex partitioning of the points $v \in V$ distributed across the space Q into a set of partitions. We allow each partition $\mathcal{P}_i, i \in [1, n]$ to "steal" points from its set of neighbors $\mathcal{N}_{\mathcal{P}_i}$. The focus of the algorithm is to determine which vertices can be transferred from one partition to another without creating an intersection between the convex partitions, and which vertices can be stolen to effectively converge to a solution that locally maximizes our measure of equality.

We now discuss how to determine which vertices can be stolen without introducing intersections between partitions. We then present how to compute which vertex is best to steal, if any, and finally present a proof of convergence and data from simulation. To compute which vertex to steal, each robot first computes the convex hull of its partition $\mathcal{P}$; then for each vertex v_i in the hulls of its neighbors, it considers the region that would be added to the polygon defined by the convex hull of $\mathcal{P}$ if v_i were moved into $\mathcal{P}$. Any v_i that would not create an intersection between two polygons if added to $\mathcal{P}$ is considered a *stealable* vertex. The area added to the region can be quickly tested for intersection by finding the triangle formed by the tangent rays between $\mathcal{P}$ and v_i and testing the edges of each of the hulls in $\mathcal{N}_{\mathcal{P}}$ for intersection with that triangle (see Figure 1). In higher dimensional cases this extends to the pyramid formed by tangent planes.

We measure equality using a cost function $\mathcal{H}$ from [12] with a constant distance function. Given that each vertex v has a mass $\phi(v)$:

$$M_{\mathcal{P}} \triangleq \sum_{v \in \mathcal{P}} \phi(v) \tag{1}$$

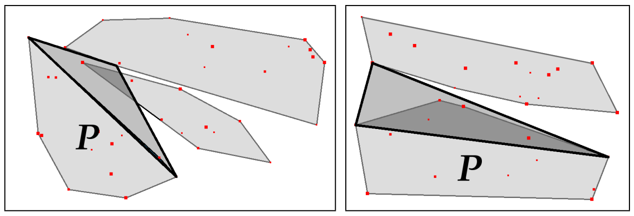

Fig. 1. Test to identify stealable vertices. The triangles with bold outlines mark the region that would be added to $\mathcal{P}$ due to a trade of a vertex. (left) Adding the vertex would cause a collision between two polygons. (right) This vertex would be considered a valid candidate to trade.

Algorithm 2 Partitioning Algorithm

```
1: Deploy into $Q$ at random pose $\mathbf{p}_i$
2: $\mathcal{P} \leftarrow \{v | (||pose(v) - \mathbf{p}_i|| < ||pose(v) - \mathbf{p}_j||) \forall j \neq i\}$
3: loop
4:     compute convex hull of $\mathcal{P}$
5:     update $\mathcal{N}_P$
6:     $X \leftarrow \{v | v \in \mathcal{N}_{\mathcal{P}}, v$ is stealable$\}$
7:     $i \leftarrow \underset{v_i \in X}{\text{argmax}}(\Delta \mathcal{H}_{\mathcal{P}}(v_i))$
8:     if $\Delta \mathcal{H}_{\mathcal{P}}(v_i) > 0$ then
9:         communicate to $\mathcal{N}_j : v_i \in \mathcal{P}_{\mathcal{N}_j}$ to remove $v_i$
10:        $\mathcal{P} \leftarrow \mathcal{P} \cup v_i$
11:    end if
12: end loop
```

$$\mathcal{H}_Q = \prod_{i \in [1,n]} M_{\mathcal{P}_i} \tag{2}$$

Without loss of generality, if we consider moving a vertex v from $\mathcal{P}_1$ to $\mathcal{P}_2$, we can compute the change in mass:

$$\Delta \mathcal{H}_Q = \left(\prod_{i=3}^{n} M_{\mathcal{P}_i} \right) \left(M_{\mathcal{P}_2} + \phi(v) \right) \left(M_{\mathcal{P}_1} - \phi(v) \right) - \mathcal{H}_Q \tag{3}$$

When comparing two potential exchanges of vertices, we only need knowledge of the partitions that will change in order to compute both the sign and relative magnitude of our deltas. We therefore need only local knowledge to determine which vertex, if any, is best to trade. We can therefore compute a scaled local $\Delta \mathcal{H}_{\mathcal{N}}$ of moving some v from some neighbor's partition $\mathcal{P}_i$ to $\mathcal{P}_{self}$ with:

$$\Delta \mathcal{H}_{\mathcal{N}} = \left(\prod_{\mathcal{P}_k \in \mathcal{N} \wedge \mathcal{P}_k \neq \mathcal{P}_i} M_{\mathcal{P}_k} \right) \left(\phi(v) (M_{\mathcal{P}_i} - M_{\mathcal{P}_{self}} - \phi(v)) \right) \tag{4}$$

$$\Delta \mathcal{H}_{\mathcal{N}} = \frac{\Delta \mathcal{H}_Q}{\prod_{\mathcal{P} \notin \mathcal{N}_{\mathcal{P}_{self}}} M_{\mathcal{P}}} \tag{5}$$

A. Convergence

Theorem 1: Algorithm 2 will converge to a local maximum.

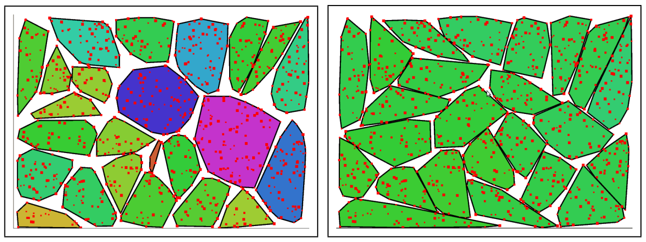

Fig. 2. Data from running partitioning algorithm. (left) Initial configuration and (right) after 26 time-steps on a set of point-masses with random location and mass. Shade is a function of total mass of a partition.

Proof: We know that the denominator in equation 5 will be unchanged by a vertex being stolen and that therefore

$$\underset{v_i \in X}{\text{argmax}}(\Delta \mathcal{H}_{\mathcal{N}}(\mathcal{P} \leftarrow v_i)) = \underset{v_i \in X}{\text{argmax}}(\Delta \mathcal{H}_Q(\mathcal{P} \leftarrow v_i)) \tag{6}$$

so each stolen vertex will result in an increase in $\mathcal{H}_Q$. The value of $\mathcal{H}$ is bounded from above and all $|\Delta \mathcal{H}|$ is bounded from below, so by induction the algorithm must converge to a local maximum. ■

B. Runtime

Theorem 2: Update at each step of algorithm 2 runs in $\mathcal{O}(||\mathcal{N}||^d + ||\mathcal{N}|| ||\mathcal{P}||)$ time.

Proof: Consider a single step of algorithm 2 running on a robot in $\mathbb{R}^d$. Finding a triangle or cone takes $\mathcal{O}(||\mathcal{P}||)$ time. Checking for intersections takes $\mathcal{O}(||\mathcal{N}||^{d-1})$. This check needs to be run on $\mathcal{O}(||\mathcal{N}||)$ candidate points [13]. Computation of each $\Delta \mathcal{H}$ takes constant time, so the computation of candidate points dominates this function. The runtime per step is therefore $\mathcal{O}(||\mathcal{N}||(||\mathcal{N}||^{d-1} + ||\mathcal{P}||)) = \mathcal{O}(||\mathcal{N}||^d + ||\mathcal{N}|| ||\mathcal{P}||)$. ■

Because only the hull is considered, this is often much faster in practice.

C. Experiments

We ran the partition algorithm on several hundred randomly generated sets of pointmasses with random mass. Point location was sampled from either a uniform distribution or 2D Gaussian. The partition masses converged on all pointsets such that their standard deviation was less than twice the average mass of a point. No partitionings contained outliers after convergence, which suggests that most local maxima are good approximations of equal-mass partitioning (see Figures 2 and 3). The simulations took 15.5 minutes in an environment with 500 point masses with 12 robot state machines each running in a separate thread on a single 1.2 GHz core. Running the same environment with 5 robots converged in 2.5 minutes, and with 5 robots and 250 points the system converged consistently in under 45 seconds.

IV. Delivery and Assembly with Part Ordering

Delivery robots repeatedly choose random assembly robots and deliver the part with the highest demanding mass inside the chosen assembly robot's partition. The assembly robot waits for a delivery and then performs whatever actions are necessary to attach the part to the main structure.

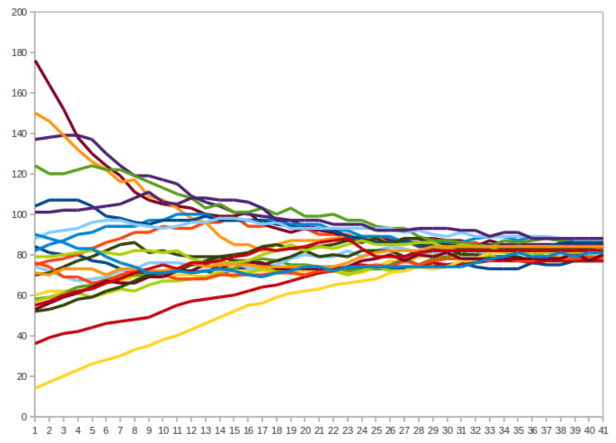

Fig. 3. Total mass of each partition over time during a typical run of the partitioning simulator.

Algorithm 3 Delivery Algorithm

```
1: loop
2:     Move within communication range of random assembly robot $r$
3:     Receive highest priority vertex in $\mathcal{P}_r$ from $r$
4:     Bring corresponding part from part source to $r$
5: end loop
```

Algorithm 4 Assembly Algorithm

```
1: Start partition algorithm (Alg. 2)
2: loop
3:     for $v \in \mathcal{P}_{self}$ do
4:         if $v$ reachable from outside construction site then
5:             $dist(v) \leftarrow 1$
6:         else
7:             $dist(v) \leftarrow 1 + min(\{dist(u)|(u,v) \in E_r\})$
8:         end if
9:     end for
10:    yield until delivery
11:    receive delivery of part $v$
12:    place $v$ and signal neighbors
13:    for $u \in$ all children and parents of $v$ do
14:        update $\phi(u)$ (Equation 23)
15:        for $w \in$ all children and parents of $u$ do
16:            update $\phi(w)$
17:        end for
18:    end for
19: end loop
```

In our definition, parts with 0 mass violate either physical or reachability constraints. Between any two parts with non-zero mass, the part with higher mass is given priority in placement. Given this planning algorithm, the mass function $\phi(\cdot)$ dictates the order in which parts are placed. We need a mass function with the following properties:

- no part placement violates global constraints
- after a part is placed the number of placeable parts tends to increase or remain constant
- the creation of bottlenecks and hallways is avoided if possible
- changes to the local density function can be efficiently calculated and updated using only local information

The precise order in which parts are placed is partially a function of the assignment of partitions and availability of parts, which are respectively non-deterministic and outside of our control. The ordering should optimize over some set of local metrics. To build this function, we present mass functions that each satisfy one of our goals and then describe a combined definition. In each definition we represent the placement of a part by removing the vertex v_i corresponding to the part placed and also removing any edge going into or out of v_i from both graphs.

Before defining our mass function we need to make a modification to the reachability graph. We need local information about the global property of reachability, and one way to do this is to modify reachability into a DAG. We do this by defining $G'_r(V, E'_r)$ such that:

$$E'_r \triangleq \{(u,v)|(u,v) \in E_r \wedge dist(u) > dist(v)\} \tag{7}$$

We are now ready to begin defining the mass function ϕ. First we define the global constraints formally: any v_i is placeable iff it will be physically supported and not render any unplaced parts unreachable. We define two boolean variables $\xi_p(v)$ and $\xi_r(v)$ to represent this criteria.

$$\xi_p(v) = (deg^-_{G_p}(v) \neq 0) \tag{8}$$

$$\xi_r(v) = (\exists j : ((v_j, v) \in E'_r) \;\wedge\; (deg^+_{G'_r}(v_j) = 1)) \tag{9}$$

ξ_p indicates that a part will not be physically supported if its indegree is anything but 0; all the parts it depends on for support must already be placed. ξ_r indicates that the part should not be placed if doing so would prevent delivery robots from reaching another part; that is, if placing a part blocks a unique exit it cannot be placed.

$$\phi_c(v) = \begin{cases} 0 & \xi_p(v) \vee \xi_r(v) \\ 1 & otherwise \end{cases} \tag{10}$$

Because the ordering of parts is defined by a set of DAGs, any mass function that obeys the constraints above and sets all other $\phi(v_i)$ to a positive value will terminate if the problem is solvable. This is sufficient to have a system that will build a structure without violating any physical constraints, however with binary mass placement order will be essentially random. The remaining mass functions allow behavior to be tuned to tend towards placement that allows for better parallelism of assembly tasks and access to the structure by delivery robots.

Before presenting these functions, we introduce the following scoring function and briefly discuss its properties. Given some function $f : x \to \mathbb{Z}^+$, and some candidate sets X_i with the property $||X_i|| \leq c_X \forall i$:

$$score(f(\cdot), X) = \sum_{x \in X} \left(2^{f(x)}\right)^{-c_X} \tag{11}$$

This function takes advantage of the properties of geometric series to provide a function that will, given two candidate sets X_1 and X_2, give a higher score to the set with the most values generating the lowest value of f. That is, if we identify the lowest value of $f(x)$, y which is generated by a different number of members of the two sets:

$$y = \min_i \{i : ||\{x \in X_1 | f(x) = i\}|| \neq ||\{x \in X_2 | f(x) = i\}||\} \tag{12}$$

then $score$ has the property that

$$score(f, X_i) > score(f, X_j) \tag{13}$$

implies

$$||\{x \in X_i : f(x) = y\}|| > ||\{x \in X_j : f(x) = y\}|| \tag{14}$$

For example: consider two nodes on a directed graph with sets of children X_1 and X_2, and a function $f(v)$ which returns the outdegree of a node. The node with more children that have outdegree 0 will have a higher score ($score(f, X_i)$). In the case of a tie, the node with more children with outdegree 1 will have a higher score. After that ties are broken by the number of children with outdegree 2, and so on. We use this function extensively in our definitions.

First we define a function that will help to place parts such that we first maximize the number of parts still available to be placed (i.e., reveal as many new parts as possible). A reasonable function could rank parts first by the number of physical dependencies they satisfy. We can represent this ranking with the $score$ function:

$$\phi_p(v_i) = score(deg^-_{G_p}, \{v_j | (v_i, v_j) \in E_p\}) \tag{15}$$

Similarly, we would like to place blocks that are least likely to cause a bottleneck first. By rating blocks by the number of different ways to reach their children we can place preference against restricting high-traffic paths. We also would like to tend toward placing parts in harder-to-reach locations first, so we need to define a slightly more complex test function $g(v_i) = \max_j(deg^+_{G'_r}(v_j)) - deg^+_{G'_r}(v_i)$.

$$\phi_r(v_i) \sim score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{16}$$

We also would like to tend toward working in areas far from the easily reachable edge of the system first (i.e., at the end of a hallway). We can use the distance function from Algorithm 4 to measure this:

$$\phi_r(v_i) \sim dist(v_i) \tag{17}$$

To combine these two statements we normalize the distance function to between $\frac{1}{2}$ and 1. The score function behaves such that multiplying by a half is the equivalent of redefining the input function $f'(\cdot) = f(\cdot) + 1$. In this case doing so would effectively lower the outdegree of each of a node's children by 1, thus lowering the node's priority. This allows us to scale ϕ by distance without breaking the tiered behavior of the score function.

$$k_{dist}(v_i) = \frac{dist(v_i) - 1}{2(\max(dist(v) \forall v \in V, 2) - 1)} \tag{18}$$

$$\phi_r(v_i) = k_{dist}(v_i) score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{19}$$

Finally, in combining these three measures of mass, we need to rescale our masses to allow comparison between ϕ_r and ϕ_p. To achieve this we introduce two scaling factors: β which rescales the range of in-degrees of nodes in E'_r to match that of E_p, and γ which can prioritize reachability or physical dependency as required by the task. The exact tuning of these functions varies depending on the capability and number of each class of robot, and this relationship is left as future work.

$$\beta = \frac{\max_{v_i}(deg^-_{G_p}(v_i))}{\max_{v_i}(deg^+_{G'_r}(v_i))} \tag{20}$$

$$\gamma \in \{-1, 0, 1\} \tag{21}$$

If we define $g'(v_j) = \beta(g(v_j) + \gamma)$, we can introduce those scaling factors to the reachability function by substituting into equation 19, which will normalize it to resemble the physical dependency function:

$$\phi'_r(v_i) = k_{dist}(v_i) score(g', \{v_j | (v_j, v_i) \in E'_r\}) \tag{22}$$

We can now combine equations (22), (15), and (10) to define our combined mass function for use by the controller.

$$\phi(v_i) = \phi_c(v_i)(\phi'_r(v_i) + \phi_p(v_i)) \tag{23}$$

A. *Runtime*

Upon the placement of a part, at most c parts will have a change of degree, which in turn means only c^2 parts have a potential change in mass. This allows constant time for a robot to update all masses after a part has been placed.

B. *Convergence*

Theorem 3: The controller outlined in algorithms 3 and 4 will converge to a complete structure if possible.

Proof: Our constraints are described by two DAGs. The mass function we describe here gives positive mass to all vertices with no unplaced parents, which by definition describes and follows a valid topological ordering of both G_p and G'_r, and will therefore converge without violating either sets of constraints. ■

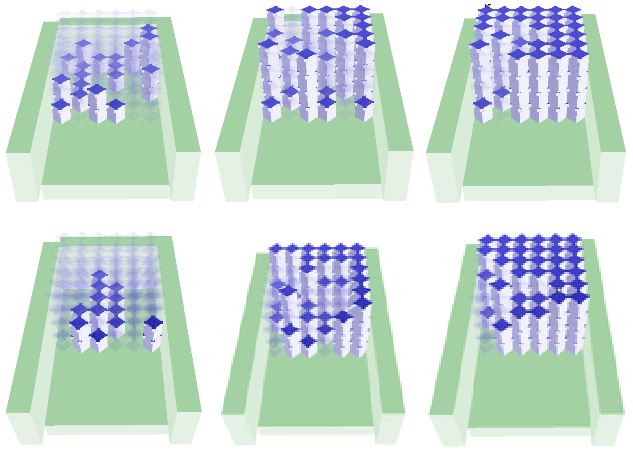
Fig. 4. Part placement while building a solid cube using (top) uniform mass and (bottom) ordering. Note that without the ordering algorithm, work in the front occurs first (top middle), making it harder for delivery robots to reach subassemblies in the back. Also note how more of the stacks of blocks in the top right have reached their maximum height, leaving less opportunities for parallelism. A more detailed discussion of a simulator trail is given in Section V-A.

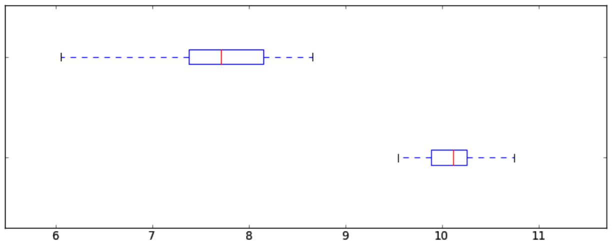

Fig. 5. The average number of parts with positive mass across time over 50 runs of building a solid cube at the end of a hallway with 5 assembly and 4 delivery robots. (top) With uniform mass on placeable parts and (bottom) using the proposed algorithm.

V. Simulation and Experiments

We have implemented Algorithm 3 and 4 in simulation and evaluated them on several construction test cases, and have performed some preliminary experiments on a physical platform. In simulation we used two structures: one which demonstrates the properties of the controller in a high-stress scenario, and one which demonstrates a structure that might be encountered in actual applications. On our platform we ran tests on structures to test each type of contraint.

Each simulated environment was tested at least 50 times for each possible permutation of between 1 and 5 delivery and assembly robots on three environments - an empty box and solid box at the end of a hallway (Figure 4: complex reachability, relatively simple physical dependencies) and a model airplane (Figure 6: complex physical dependencies, less complex reachability). All runs completed construction without violating constraints. We rated each run on availability - the number of parts ready to be placed - and throughput - the maximal number of delivery robots that could make a simultaneous delivery. In our practical experiments we built structures using two delivery and two assembly robots, and

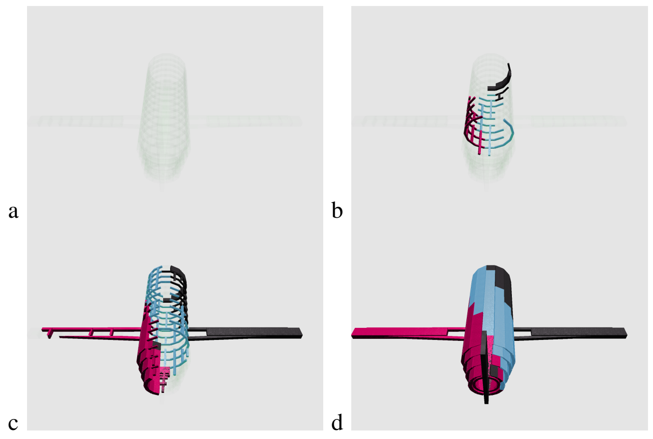

Fig. 6. The controller running on three assembly and two delivery robots building a model plane in simulation, which has complex physical dependencies and relatively simple reachability constraints. Parts are color-coded by the robot that placed them.

observed no failures in our ordering algorithm.

As a baseline, we compared the results of the mass function (equation 23) with the minimal constraint (equation 10). We saw a clear performance advantage to our algorithm in creating opportunities for parallelism (Figure 5). The throughput in our example situations did not exhibit statistically significant deviation from the uniform mass functions until late in each run.

A. Simulation Example: Model Plane

As an example of the behavior of the system we considered building a model plane with complex physical dependencies and a solid block which has complex reachability constraints. For illustration, we have colored parts differently if they were placed by different assembly robots. In this example, there are several constraints:

- The scaffolding must be completed before panels are placed over it
- Scaffolding must be built ground up, and have horizontal struts holding it up before it is more than 3 struts tall
- Wings require stable scaffolding before they are built, and the attachment point of main wing supports is unreachable after panels have been placed over the scaffolding around the connection point.

We initialize the system with three delivery and two assembly robots. The plane blueprint consists of 686 parts and 2402 edges which fully represent the dependencies described above.

In the screenshots from the simulator in Figure 6, these constraints are followed, and each assembly robot does roughly the same amount of work. In (a) we see an empty blueprint as the assembly robots are deployed. In (b) construction begins, with each of the three partitions filling up with roughly the same amount. In (c) the scaffolding has been mostly completed before panels are added, which maximizes the amount of potential work to be done and

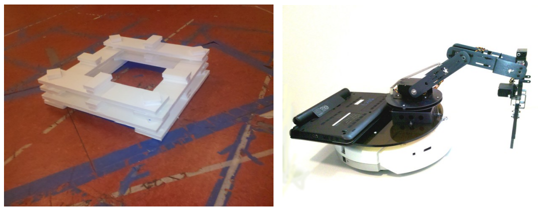
Fig. 7. The box built by the system and the modified mobile manipulator platform

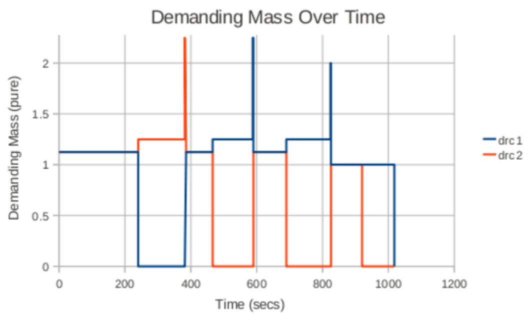

Fig. 8. Demanding mass of the two assembly robots over time during the experiment pictured in figure 9. After a part is placed we assigned work arbitrarily and allowed the algorithm in section III to handle rebalancing, causing a transient spike in demanding mass of a robot after each part placement.

would allow for efficient parallelism if more robots were added. (d) shows the finished product, a completed airplane. The three robots placed 220, 232, and 234 parts respectively over the course of this example. It is also notable that when the robot placing dark grey parts completed its section of the assembly task it relocated to the still incomplete tail and resumed construction there, which was a direct result of the active partitioning algorithm.

B. Preliminary Experiments

Using an experimental setup of 2 assembly and 2 delivery robots we ran our controller from section V-A on a physical platform. Our experimental setup used the robots to build multi-layer boxes to test physical contraints, (see figure 7) and rows of boards to test reachability contraints. The robots were built using the iCreate robot base and Crustcrawler arms with specialized grippers much like the system we presented in [4], but with the removal of specialize parts to improve the robustness and battery life of the robots and allow for longer experiments. We built the box 8 times with no error in part ordering. The demanding mass was normalized between the two robots within at most two messages sent per robot after each part placement. Each robot had non-zero demanding mass when more than one task was available, allowing for paralelization (figure 8). We are currently extending and improving this system.

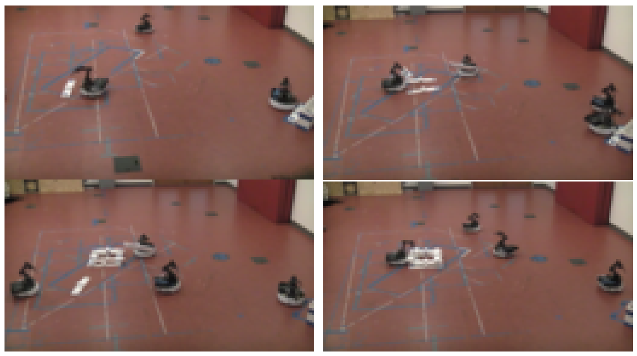
Fig. 9. The controller running on two assembly and two delivery robots building a box out of boards. There is a supply depot in the bottom right corner of each picture.

VI. Conclusion

We have extended our distributed controller for building structures to conform to physical dependencies. This is achieved by defining a mass function which reflects the priority of part placement and blocks the controller from reaching physically impossible states. The mass function tends towards efficient parallelization by projecting potential future states. This function introduces the need for equal-mass convex tessellation, for which we present a novel algorithm.

VII. Acknowledgements

This project has been supported in part by The Boeing Company, the U.S. National Science Foundation, Emerging Frontiers in Research and Innovation (EFRI) grant #0735953, MURI SMARTS grant #N0014-09-1051, and MURI SWARMS grant #544252. We are grateful for this support.

References

[1] S. Yun and D. Rus, “Distributed coverage with mobile robots on a graph Locational optimization and equal-mass partitioning,” in *Workshop on the Algorithmic Foundations of Robotics*, 2010.
[2] S. 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.
[3] S. Yun and D. Rus, “Adaptation to robot failures and shape change in decentralized construction,” in *Proceedings of IEEE International Conference on Robotics and Automation*, 2010.
[4] A. Bolger, M. Faulkner, D. Stein, L. White, S. Yun, and D. Rus, “Experiments in decentralized robot construction with tool delivery and assembly robots,” in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.
[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] 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.
[7] L. Matthey, S. Berman, and V. Kumar, “Stochastic strategies for a swarm robotic assembly system.” in *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 1953–1958.
[8] S. 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.
[9] S. 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] O. Baron, O. Berman, D. Krass, and Q. Wang, “The equitable location problem on the plane,” *European Journal of Operational Research*, vol. 183, no. 2, pp. 578 – 590, 2007.

[11] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009, electronically available at http://coordinationbook.info.

[12] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, “Coverage control for mobile sensing networks,” *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 2, pp. 243 – 255, 2004.

[13] F. Preparata and S. Hong, “Convex hulls of finite sets of points in two and three dimensions,” in *Communications of the ACM*, 1977.

[14] M. Pavone, E. Frazzoli, and F. Bullo, “Distributed policies for equitable partitioning: Theory and applications,” in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, dec. 2008, pp. 4191 – 4197.

[15] M. Schwager, J.-J. Slotine, and D. Rus, “Decentralized, adaptive control for coverage with networked robots,” in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 3289 –3294.

[16] G. Theraulaz and E. Bonabeau, “Coordination in distributed building,” *Science*, vol. 269, no. 5224, pp. 686–688, 1995. [Online]. Available: http://www.sciencemag.org/content/269/5224/686.abstract

[17] P. Yiu, “The uses of homogeneous barycentric coordinates in plane euclidean geometry,” *International Journal of Mathematical Education in Science and Technology*, vol. 31, pp. 569–578, 2000.