Constraint-Aware Distributed Robotic Assembly and Disassembly

by

Timothy Ryan Schoen

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author .....
Department of Electrical Engineering
and Computer Science
May 21, 2012

Certified by ....
.....
Daniela Rus
Professor
Thesis Supervisor

Accepted by.....
.....
Prof. Dennis M. Freeman, Chairman
Masters of Engineering Thesis Committee

Constraint-Aware Distributed

Robotic Assembly and Disassembly

by

Timothy Ryan Schoen

Submitted to the Department of Electrical Engineering
and Computer Science
on May 21, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this work, we present a distributed robotic system capable of the efficient assembly and disassembly of complex three-dimensional structures. We introduce algorithms for equitable partitioning of work across robots and for the efficient ordering of assembly or disassembly tasks while taking physical constraints into consideration. We then extend these algorithms to a variety of real-world situations, including when component parts are unavailable or when the time requirements of assembly tasks are non-uniform. We demonstrate the correctness and efficiency of these algorithms through a multitude of simulations. Finally, we introduce a mobile robotic platform and implement these algorithms on them. We present experimental data from this platform on the effectiveness and applicability of our algorithms.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgements

I would like to thank the Boeing Corporation for sponsoring this research, and Daniela Rus for her advice and guidance over the past two years. I am indebted to the members of the Distributed Robotics Lab, and especially to my fellow robot wranglers David Stein and Ross Knepper. Special thanks to Seungkook Yun, without whose work none of this could have been possible. And finally, I am incredibly grateful for my friends and family, who have given me so much love and support over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Five million individuals in the United States are employed by the construction and extraction services industry, making it an average-sized industry with about four percent of the US workforce[1]. However, individuals in this industry suffer a disproportionate amount of the work-related fatalities in the country; construction results in the largest number of work-related fatalities of any industry[2]. Among these fatalities, the largest portion of them are the construction laborers themselves, the individuals performing the lowest-level construction tasks[3]. We believe that these trends extend to the other countries as well, and as such make construction a dangerous job.

Most constructions tasks require the transportation, manipulation, and precise assembly of a variety of objects, many of which may be heavy or hazardous in a variety of ways. We believe that humans are not particularly well-suited for these tasks, and propose instead the introduction of robotics to accomplish these assembly tasks. Thus we remove humans from the situations most hazardous to their health and well-being.

We acknowledge that there are many components of assembly for which a human is much more able to complete than a robot. The solutions we propose are not intended to completely replace humans in the assembly progress, but instead to replace only the most basic and dangerous of these activities. As the field of robotics progresses, one can imagine more responsibility being allocated to the robotic platform to complete assembly activities,

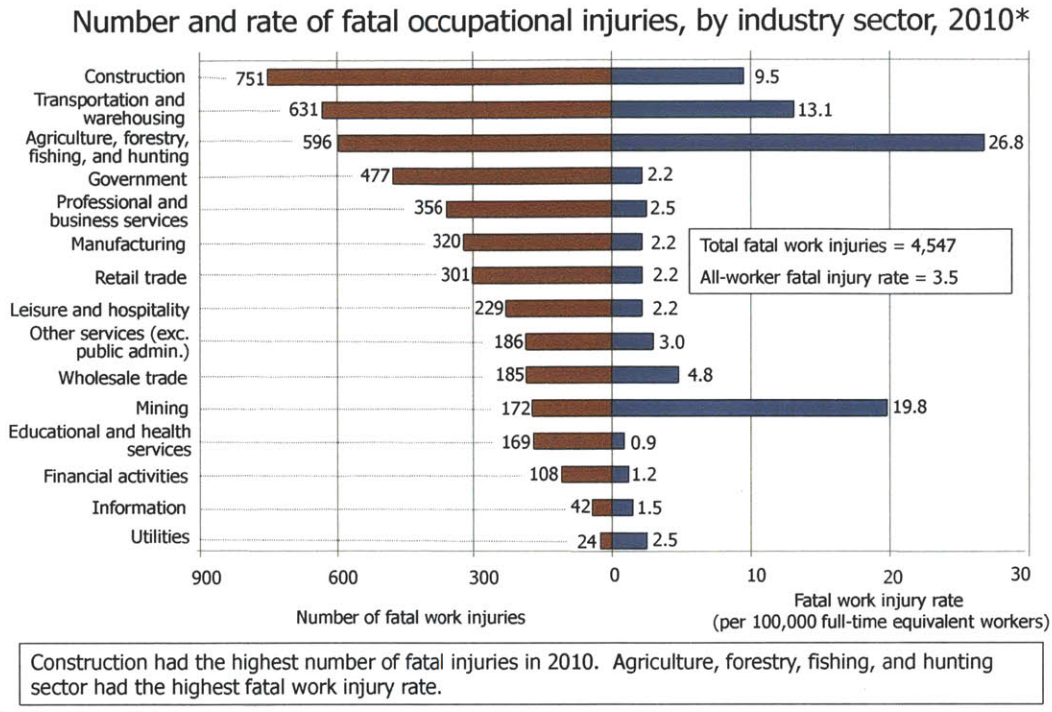


Figure 1-1: At 751 fatalities in 2010, construction results in the largest number of work-related fatalities of any industry. As reported by the U.S. Bureau of Labor Statistics.

but still being supervised by skilled human workers.

One approach to robotic construction is a centralized controller that completes the assembly task in a predefined order. This approach has a number of drawbacks. The efficiency and parallelism is limited, in the sense that the tasks are completed in a serial manner. In the case of multiple robots receiving commands from a centralized controller, parallelism increases but there is a lack of scalability. As the number of robots increases the complexity of processing and communication for the centralized controller grows quickly. A centralized approach is also not adaptable to many types of failure or disturbance. A fatal failure in the central controller would halt the assembly process regardless of the number of robots involved. A shortage of a particular part type could equally freeze the serial assembly process.

We believe that adaptive, decentralized algorithms that address these issues will be

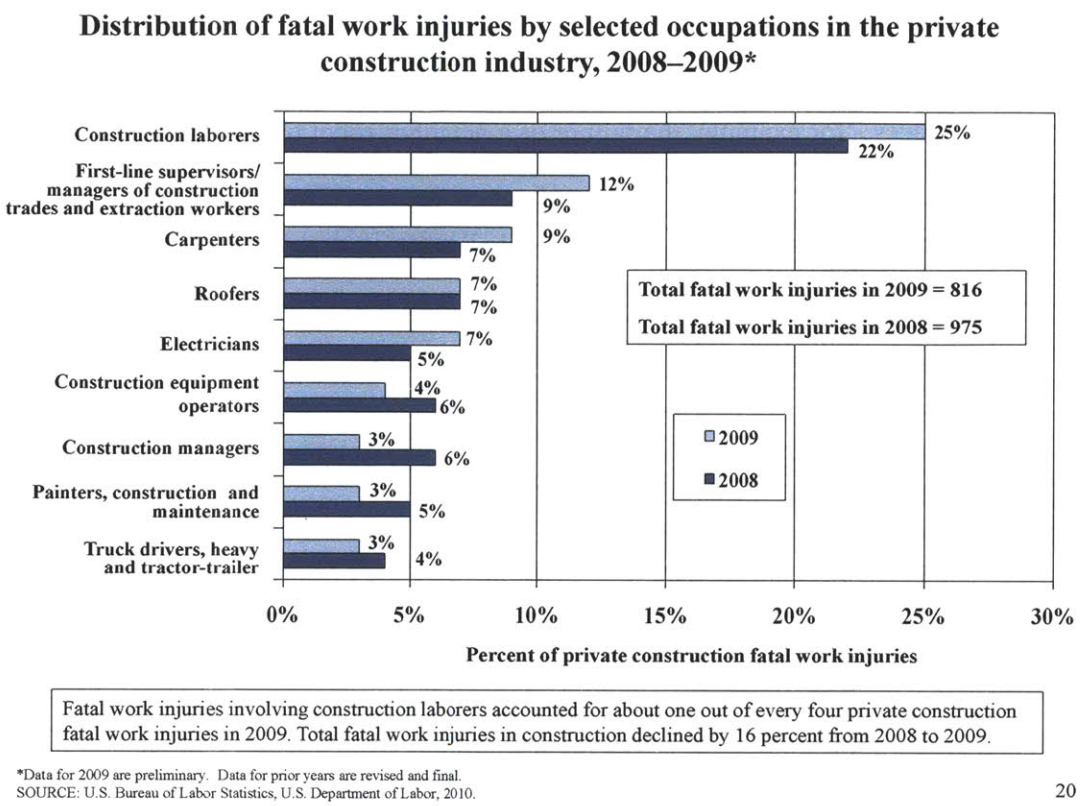


Figure 1-2: Among construction fatalities, about a quarter of the fatalities are suffered by the construction laborers themselves. As reported by the U.S. Bureau of Labor Statistics.

central to the future of manufacturing. To that end, our work has focused on developing and implementing these algorithms. We imagine a team of n robots working cooperatively to construct a given structure. We divide these robots into two classes. The first class, part delivery robots, are specialized for retrieving parts from a source location or repository and delivering them to the second class of robots. This second class, the assembly robots, are specialized for performing the assembly task given the parts delivered to them. These assembly tasks could be placing a part, bolting pieces together, applying adhesive, or any localized assembly task.

Each of the assembly robots is given a blueprint of the target structure, but beyond that the processing, control, and communication are completely decentralized. This presents several unique algorithmic challenges. The robots must decide amongst themselves how to partition the assembly work in a way that is equitable and maximizes parallelism. Given

these partitions, the robots must then decide how to sequence their order of operations, again to maximize efficiency and parallelism. Each robot must also be capable of responding to disturbances - for example, the failure of a neighbor robot or the shortage of a part supply.

We present algorithmic solutions to these challenges in a provably correct manner, while maintaining several desirable properties. We then implement our algorithms on a robotic platform to demonstrate their practicality in manufacturing tasks.

1.2 Algorithmic Contributions

The work presented in this paper builds on a body of knowledge developed at MIT and other institutions regarding the efficient construction of structures using teams of distributed robots. Specifically, it extends the work of Seungkook Yun and David Stein to make the set of assembly algorithms more robust. The main contributions of this work are as follows.

1.2.1 Discrete Partitioning

As many of the algorithms presented require the equal-weight partitioning of assembly tasks, we first present a novel algorithm to partition a set of discrete point-masses into equal partitions.[1] This allows the body of work available in a construction process to be equally partitioned across a team of robots, such that each can work efficiently and the overall goal can be completed in the least amount of time. The algorithm is extendable to any dimensionality, although we envision most applications in two or three dimensions.

1.2.2 Constraint-Aware Ordered Assembly

The work of Yun et al. addressed how to efficiently assemble arbitrary structures given a blueprint, but the resulting assembly order disregarded the physical constraints the structure. We present an algorithm that considers these physical constraints, and develops an

[1]This is taken from previous work, "Constraint-Aware Coordinated Construction of Generic Structures" by D. Stein, T. R. Schoen, and D. Rus.[4]

assembly order designed to maximize parallelism and reduce bottlenecks in the task.[2]

1.2.3 Ordered Assembly with Part Unavailability

The above assembly algorithm is then adapted to account for the fact that some parts required for assembly may not be always present. We present an adaptation to the algorithm that continues to maximize robot parallelism in the face of part shortages.

1.2.4 Ordered Assembly with Time Constraints

The above assembly algorithm is again adapted to consider the fact that all assembly operations are not equal. Some may be more complex than others, requiring more time to complete. Our algorithm takes these timing parameters into consideration when scheduling tasks, such that the structures are assembled in the most efficient manner.

1.2.5 Constraint-Aware Ordered Disassembly

Occasionally construction tasks are needed to build temporary structures, such as a scaffolding of trusses used to support another assembly. These structures must be disassembled after their purpose is complete, to clear the construction area and recycle the parts used in the structure. We present a novel algorithm for the ordering of tasks required to disassemble an arbitrary structure, again considering the physical constraints of such a disassembly process.

1.3 Organization of Thesis

The first three chapters describe the challenges this thesis attempts to solve, as well as their importance and context in the manufacturing industry. Chapter 4 describes our algorithm for discrete partitioning of work among robots, and then Chapters 5 and 6 describe the ordering algorithms used to maximize parallelism and efficiency in the assembly tasks.

[2]This is taken from previous work, "Constraint-Aware Coordinated Construction of Generic Structures" by D. Stein, T. R. Schoen, and D. Rus.[4]

Chapter 7 then uses the principles described so far to develop algorithms for efficient disassembly. A robotic platform is described in Chapter 8, and then the results of experiments with our algorithms on this platform are presented in Chapter 9. Finally, conclusions and possibilities for future work are described in Chapter 10.

Chapter 2

Related Work

Our work builds on prior research on robotic construction and distributed coverage. A simple distributed 3D construction algorithm is described by Theraulaz[5], while Werfel[6] describes a 3D construction algorithm for modular blocks in a distributed setting. Fahlman describes a system for planning how to build a structure using simple parts[7]. Stochastic algorithms for robotic construction with dependency on raw materials are analyzed by Matthey[8]. Three-dimensional construction with consideration to physical constraints such as gravity and stacking was achieved by [9].

Ayanian and Kumar developed a decentralized feedback controller for a team of robots to navigate around obstacles[10]. Stochastic policies for parallel task allocation in robotic swarms were investigated by [11]. [12] developed methods for evaluating the complexity of structures, as it applies their distributed robotic construction.

The U.S. Air Force detailed their early efforts of robotic construction in [13]. Parker et al. described a system for nest construction using a team of robots[14]. Human-robot cooperation for construction of heavy items was explored by Lee, et al.[15] A coordinated robotic lego construction experiment was described by Schuil[16]. Stroupe et al. presented a heterogeneous robotic assembly system designed to maximize a number of cost metrics[17]. Our previous work on robotic construction includes Shady3D [18] utilizing a passive bar and an optimal algorithm for reconfiguration of a given truss structure to a target structure[19], and experiments in building truss structures[20].

The graph Voronoi diagram is described by Erwig[21]. Using Voronoi partitions to

deploy robots for coverage was originally proposed by Cortez et al.[22] and has been extended several times since then for tasks such as adaptive coverage[23] and equitable partitioning[24]. Pavone et al. described a method of distributed equitable partitioning[25]. Maini et al. explored a genetic graph partitioning algorithm[26], and Leland and Hendrickson presented a study of several load balancing algorithms, in this case for parallel computing but as could be applied to other uses[27]. Durham et al. presented a decentralized algorithm for creating Voronoi partitions among robots with pairwise communication[28]. Our recent work extends the idea of equitable partitioning and combines it with coordinated construction of truss structures[29], locational optimization[30], and adaptation to failure and shape change[31].

Our approach utilizes previous work on computation using barycentric coordinates[32] and convex hulls[33]. Our early algorithms were implemented on a team of robots by Bolger, demonstrating the early practicality of our approaches[34].

Chapter 3

Problem Formulation

In this work, we address the challenge of utilizing a team of robotic mobile manipulators to construct a fixed assembly.

3.1 Assembly

We define an assembly to be a collection of parts connected to each other, creating a single structure. A blueprint defines the relative locations of each of these parts, as well as the nature of the parts, how they are connected to each other, and which parts have physical or reachability dependencies on each other. In this work, we assume that all structures are fixed; that is, after a part has been connected to an assembly, it will not be moved further.

Under this definition and assumptions a multitude of structures can be assembled, ranging from very simple two-party assemblies to complicated three-dimensional shapes such as arches, pyramids, or furniture.

3.2 Robots

We are given a team of robots, some of which specialize in the assembly of components parts into the more complicated structure - we call these the assembly robots. The rest of the robots specialize in retrieving parts from a part cache and delivering them to the assembly robots - these in turn are called delivery robots. These robots are mobile, relatively small,

and have communication capabilities with their closest neighbors. They can manipulate parts and their surroundings with manipulators of any type, including the possible use of external tools or the assistance of humans.

The team of robots is completely decentralized. There is no central robot or scheduler determining the assembly order or issuing commands to the robots. Each robot acts independently and determines on its own best course of action to take.

3.3 Demanding Mass

We define a function $\phi(v)$ over the assembly space, which we refer to as the *demanding mass*. For each part v, the demanding mass indicates the priority of that part. The robots utilize this mass function to determine the highest priority parts to place onto the assembly. In prior work the demanding mass function was smoothed so as to be continuous; in this work, the demanding mass function will be comprised of a delta function at the location of each part, and zero elsewhere.

3.4 Task Partitioning

In order to maximize parallelism, we follow the work of Yun et al. and divide the partition the demanding mass function into sections assigned to each robot[29]. There is exactly one partition per assembly robot, and each partition contains zero or more point masses representing parts that have not yet been assembled. We require that the partitions be convex and non-overlapping; each unassembled part is assigned to exactly one partition. Given this formulation, we would like partitions to have equal mass so that the work is equitable between robots and the overall structure is completed in the most efficient way possible.

The high-level description of our algorithm is as follows. Given the starting positions of the robots, we initially create a Voronoi partitioning of the parts. We calculate the convex hulls of each robot's partition. The robots then communicate with their neighbors to trade vertices on their hulls to create equal-work partitions. As we show, this is guaranteed to

converge to a local maximum.

3.5 Assembly Ordering

We assume that the blueprint provides information about the physical dependency and reachability constraints of the structure, as will be defined later. We represent this information in the form of two directed acyclic graphs (DAGs), $G_r = (V, E_r)$ and $G_p = (V, E_p)$. The nodes of the graphs represent the parts to be assembled, and the edges represent dependencies. The edges of the reachability graph point from parts that could be blocked by other parts to those blocking parts. The edges of the physical dependency graph point from parts that provide support to the parts they support. Under this formulation, edges point from parts that will be assembled earlier in the assembly process to parts that will be assembled later.

These requirements, while providing a few hard constraints about the order of assembly process, still leave a large amount of flexibility. Instead of assembling parts in a random order within these constraints, we would like to optimize our ordering to maximize parallelism and reduce bottlenecks. That is, at any point, we would like a robot to decide deterministically which part it will next assemble in order to maximize a parallelism metric.

Chapter 4

Discrete Equal Mass Partitioning

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 in more detail in Chapter 5). 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 the workspace 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 as a potential solution to this problem in [30], 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 1), 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 4-1). In higher dimensional cases this extends to the pyramid formed by tangent planes.

We measure equality using a cost function $\mathcal{H}$ from [22] with a constant distance func-

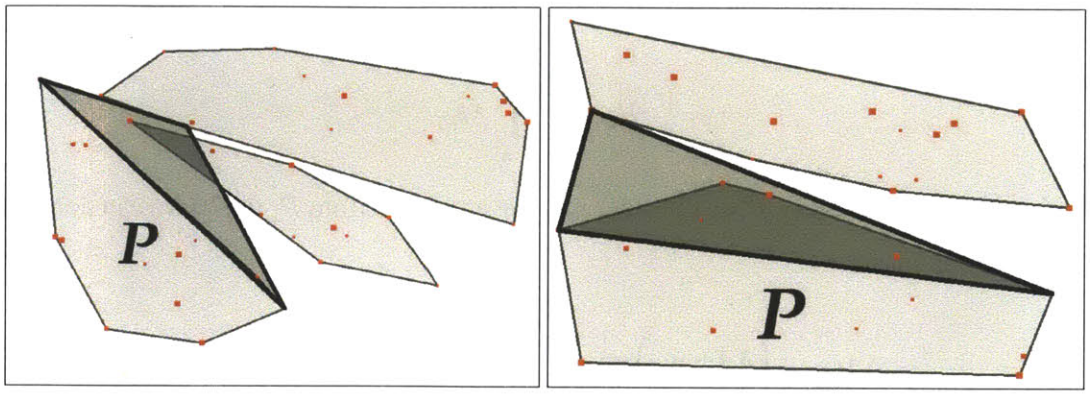


Figure 4-1: Test to identify stealable vertices. In the first image, the triangles with bold outlines mark the region that would be added to $\mathcal{P}$ due to a trade of a vertex. In the first case, adding the vertex would cause a collision between two polygons. In the second, the vertex under consideration would be a valid candidate to trade.

Algorithm 1 Partitioning Algorithm

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

tion. Given that each vertex v has a mass $\phi(v)$:

$$M_{\mathcal{P}} \triangleq \sum_{v \in \mathcal{P}} \phi(v) \tag{4.1}$$

$$\mathcal{H}_Q = \prod_{i \in [1,n]} M_{\mathcal{P}_i} \tag{4.2}$$

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:

$$\Delta\mathcal{H}_Q = \left(\prod_{i=3}^{n} M_{\mathcal{P}_i}\right)\left(M_{\mathcal{P}_2} + \phi(v)\right)\left(M_{\mathcal{P}_1} - \phi(v)\right) - \mathcal{H}_Q \tag{4.3}$$

$$\Delta\mathcal{H}_Q = \left(\prod_{i=3}^{n} M_{\mathcal{P}_i}\right)\left(M_{\mathcal{P}_1} M_{\mathcal{P}_2} + \phi(v)\left(M_{\mathcal{P}_1} - M_{\mathcal{P}_2} - \phi(v)\right)\right) - \mathcal{H}_Q \tag{4.4}$$

$$\Delta\mathcal{H}_Q = \left(\prod_{i=3}^{n} M_{\mathcal{P}_i}\right)\phi(v)\left(M_{\mathcal{P}_1} - M_{\mathcal{P}_2} - \phi(v)\right) \tag{4.5}$$

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:

$$\Delta\mathcal{H}_{\mathcal{N}} = \left(\prod_{\mathcal{P}_k \in \mathcal{N} \wedge \mathcal{P}_k \neq \mathcal{P}_i} M_{\mathcal{P}_k}\right)\phi(v)(M_{\mathcal{P}_i} - M_{\mathcal{P}_{self}} - \phi(v)) \tag{4.6}$$

$$\Delta\mathcal{H}_{\mathcal{N}} = \frac{\Delta\mathcal{H}_Q}{\prod_{\mathcal{P} \notin \mathcal{N}_{\mathcal{P}_{self}}, \mathcal{P} \neq \mathcal{P}_{self}} M_{\mathcal{P}}} \tag{4.7}$$

4.1 Convergence

Theorem 1 *Algorithm 1 will converge to a local maximum.*

Proof 1 *We know that the denominator in equation 4.7 will be unchanged by a vertex being stolen and that therefore*

$$\underset{v_i \in X}{\operatorname{argmax}}(\Delta\mathcal{H}_{\mathcal{N}}(\mathcal{P} \leftarrow v_i)) = \underset{v_i \in X}{\operatorname{argmax}}(\Delta\mathcal{H}_{Q}(\mathcal{P} \leftarrow v_i)) \tag{4.8}$$

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.

4.2 Runtime

Theorem 2 *The update at each step of Algorithm 1 runs in $\mathcal{O}(||\mathcal{N}||^d + ||\mathcal{N}||||\mathcal{P}||)$ time.*

Proof 2 *Consider a single step of Algorithm 1 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 [33]. 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.

4.3 Simulations

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 4-2 and 4-3). 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

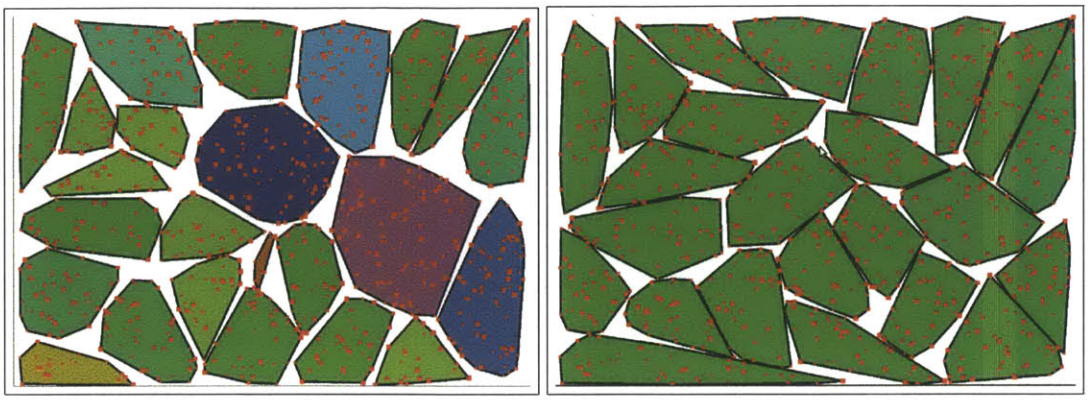


Figure 4-2: Data from running partitioning algorithm. The first image shows the initial configuration, and the second shows the partitions after 26 time-steps on a set of point-masses with random location and mass. Shade is a function of total mass of a partition.

2.5 minutes, and with 5 robots and 250 points the system converged consistently in under 45 seconds.

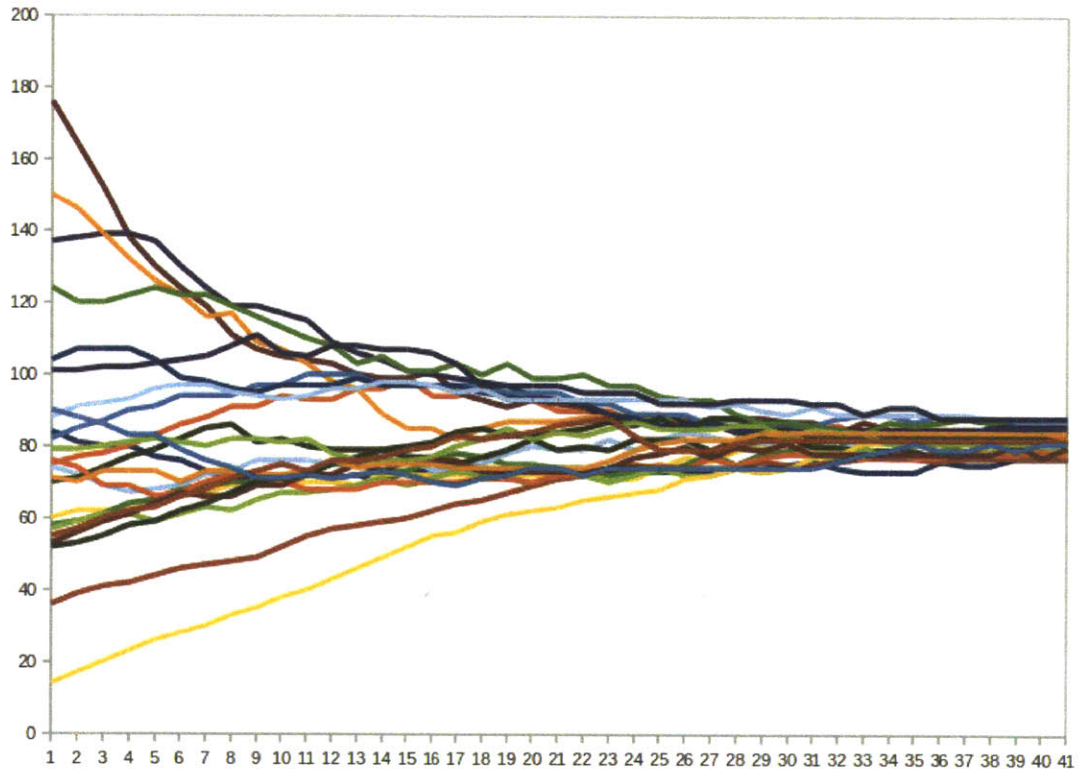


Figure 4-3: Total mass of each partition over time during a typical run of the partitioning simulator.

Chapter 5

Delivery & Assembly with 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.

Algorithm 2 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**

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

Algorithm 3 Assembly Algorithm

```
1: Start partition algorithm (Alg. 1)
2: loop
3:   for v ∈ P_self do
4:     if v reachable from outside construction site then
5:       dist(v) ← 1
6:     else
7:       dist(v) ← 1 + min({dist(u)|(u, v) ∈ 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 ∈ all children and parents of v do
14:    update φ(u) (Equation 5.24)
15:    for w ∈ all children and parents of u do
16:      update φ(w)
17:    end for
18:  end for
19: end loop
```

- 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:

$$E'_r \triangleq \{(u, v)|(u, v) \in E_r \wedge dist(u) > dist(v)\} \tag{5.1}$$

We are now ready to begin defining the mass function ϕ. First we define the global con-

straints 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.

$$\xi_p(v) = (deg^-_{G_p}(v) \neq 0) \tag{5.2}$$

$$\xi_r(v) = (\exists j : ((v_j, v) \in E'_r) \ \wedge \ (deg^+_{G'_r}(v_j) = 1)) \tag{5.3}$$

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.

$$\phi_c(v) = \begin{cases} 0 & \xi_p(v) \vee \xi_r(v) \\ 1 & otherwise \end{cases} \tag{5.4}$$

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 $f : x \rightarrow \mathbb{Z}^+$, and some candidate sets X_i with the property $||X_i|| \leq c_X \forall i$:

$$score(f(\cdot), X) = \sum_{x \in X} \left(2^{f(x)}\right)^{-c_X} \tag{5.5}$$

The correctness of ours algorithms depends on a property of the function, which we shall call the *ranking property*. The property is defined as follows. Assume we are given a

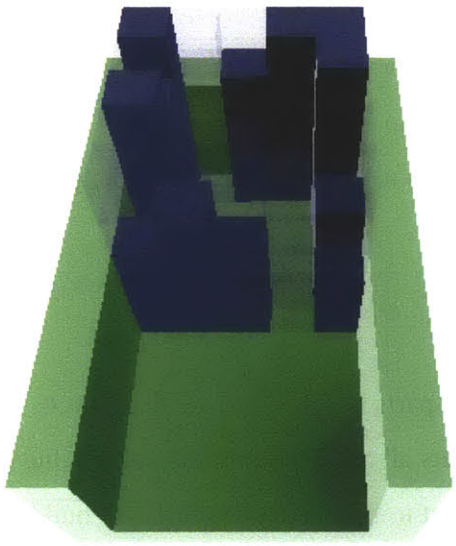


Figure 5-1: The state of the system mid-run building a hollow blue box at the end of a green hallway, with a uniform mass function. With nothing but basic knowledge of the DAG the system can complete the structure, but part placement is suboptimal. Note that the front of the structure is mostly built, creating a bottleneck which limits the rate at which delivery robots can deliver parts; and some parts of the structure are built to full height, limiting the number of assembly bots that can work simultaneously.

function $f : x \rightarrow \mathbb{Z}^+$ and two sets X_1 and X_2 which are bounded in size by some integer constant $k > 0$. The ranking property states that the lowest value of $f(x)$ produced by the members of the sets for which the two sets does not have an equal number of elements producing that value, the set with more elements producing that value will have a higher score. Formally, if

$$y = \min_i \{i : ||\{x \in X_1 | f(x) = i\}|| \neq ||\{x \in X_2 | f(x) = i\}||\} \tag{5.6}$$

$$||\{x \in X_1 | f(x) = y\}|| > ||\{x \in X_2 | f(x) = y\}|| \implies score(f, X_1) > score(f, X_2) \tag{5.7}$$

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.

Theorem 3 *The ranking property holds for the score function.*

Proof 3 *Assume we are given a function $f : x \rightarrow \mathbb{Z}^+$ and two different sets X_1 and X_2 which are bounded in size by some integer constant $k > 0$. Without loss of generality, we will say that the function $f(\cdot)$ produces the same number of results on X_1 and X_2 for all values lower than some on-negative integer y. Stated differently, y is the lowest value for which $f(\cdot)$ produces a different number of results on the two sets. Again without loss of generality, assume that $f(\cdot)$ produces more results on X_1 at y than on X_2 at y.*

$$k > \log k \tag{5.8}$$

Subtracting a term ky from both sides and rearranging,

$$-ky > \log k - k(y + 1) \tag{5.9}$$

$$2^{-ky} > k2^{-k(y+1)} \tag{5.10}$$

Given either set $X_i, i \in \{1, 2\}$, we can split the score function into parts.

$$score(f(\cdot), X_i) = \sum_{x \in X_i | f(x) < y} \left(2^{-kf(x)}\right) + \sum_{x \in X_i | f(x) = y} \left(2^{-ky}\right) + \sum_{x \in X_i | f(x) > y} \left(2^{-kf(x)}\right) \tag{5.11}$$

We observe that the first term in 5.11 is necessarily the same for both sets, and can therefore be treated as a constant. Furthermore, we see that the following bounds must exist:

$$\sum_{x \in X_1 | f(x) = y} \left(2^{-ky}\right) - \sum_{x \in X_2 | f(x) = y} \left(2^{-ky}\right) \geq 2^{-ky} \tag{5.12}$$

$$\sum_{x \in X_1 | f(x) > y} \left(2^{-kf(x)}\right) - \sum_{x \in X_2 | f(x) > y} \left(2^{-kf(x)}\right) \geq -k2^{-k(y+1)} \tag{5.13}$$

Using 5.11, 5.12, and 5.13, the difference between can be represented as

$$score(f(\cdot), X_1) - score(f(\cdot), X_2) \geq 2^{-ky} - k2^{-k(y+1)} \tag{5.14}$$

Per 5.10, this difference is strictly positive, and therefore

$$score(f(\cdot), X_1) > score(f(\cdot), X_2) \tag{5.15}$$

Figure 5-2 depicts this property of the score function.

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:

$$\phi_p(v_i) = score(deg^-_{G_p}, \{v_j | (v_i, v_j) \in E_p\}) \tag{5.16}$$

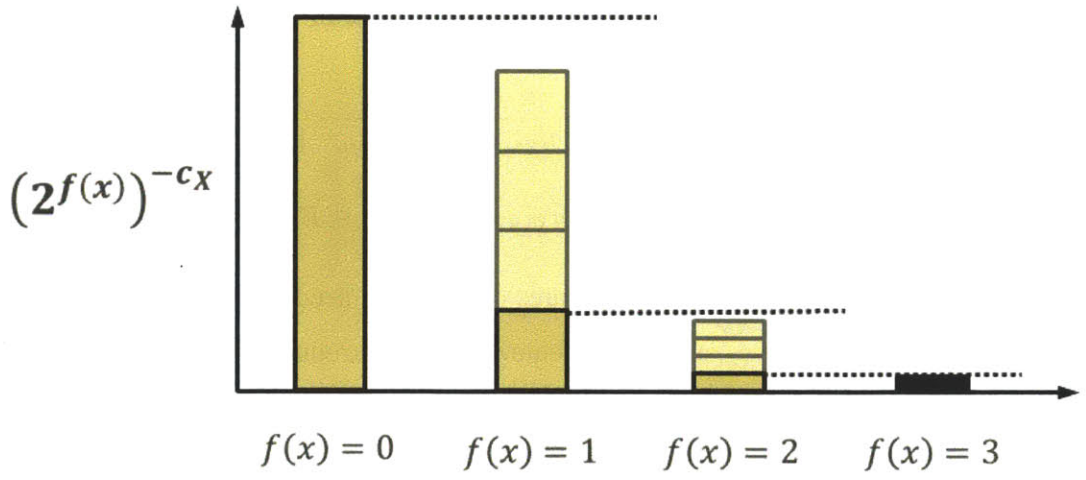


Figure 5-2: The score function has the property that given two sets, the function will give a higher score to the set with most values generating the lowest value of f.

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 $g(v_i) = \max_j(deg^+_{G'_r}(v_j)) - deg^+_{G'_r}(v_i)$.

$$\phi_r(v_i) \sim score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.17}$$

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 3 to measure this:

$$\phi_r(v_i) \sim dist(v_i) \tag{5.18}$$

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.

$$k_{dist}(v_i) = \frac{dist(v_i) - 1}{2(\max(dist(v) \forall v \in V, 2) - 1)} \tag{5.19}$$

$$\phi_r(v_i) = k_{dist}(v_i) score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.20}$$

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.

$$\beta = \frac{\max\limits_{v_i}(deg^-_{G_p}(v_i))}{\max\limits_{v_i}(deg^+_{G'_r}(v_i))} \tag{5.21}$$

$$\gamma \in \{-1, 0, 1\} \tag{5.22}$$

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 5.20, which will normalize it to resemble the physical dependency function:

$$\phi'_r(v_i) = k_{dist}(v_i) score(g', \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.23}$$

We can now combine equations (5.23), (5.16), and (5.4) to define our combined mass function for use by the controller.

$$\phi(v_i) = \phi_c(v_i)(\phi'_r(v_i) + \phi_p(v_i)) \tag{5.24}$$

5.1 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

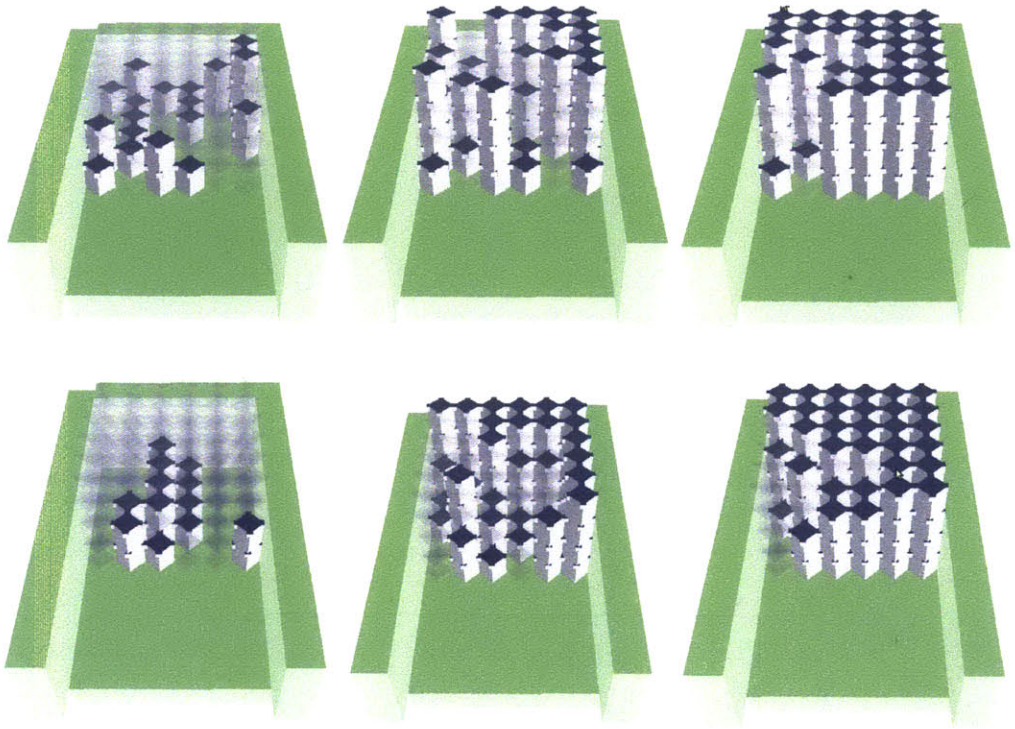


Figure 5-3: Part placement while building a solid cube using uniform mass (top) and ordering (bottom). Note that without the ordering algorithm, work in the front occurs first (top middle), making it harder for delivery robots to reach subassemblies in the back. Also note how more of the stacks of blocks in the top right have reached their maximum height, leaving less opportunities for parallelism.

to update all masses after a part has been placed.

5.2 Convergence

Theorem 4 *The controller outlined in algorithms 2 and 3 will converge to a complete structure if possible.*

Proof 4 *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.*

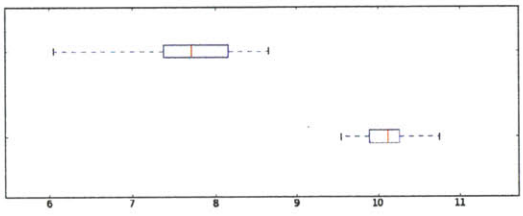


Figure 5-4: The average number of parts with positive mass across time over 50 runs of building a solid cube at the end of a hallway with 5 assembly and 4 delivery robots, with uniform mass on placeable parts (top) and masses calculated using the proposed algorithm (bottom).

Chapter 6

Adaptation in Decentralized Assembly

6.1 Decentralized Scheduling Algorithm in the Presence of Part Supply Uncertainty

Let us define λ_v as the part type of part v, and assume all types of the same part are identical and that there are a finite number of part types. Many parts may have the same type, and each assembly requires one or more types of part.

Let the function $n(\lambda)$ represent the number of parts available of type λ. We assume that all robots have information about the number and type of robots available, although for our purposes it is sufficient to represent $n(\lambda) \in \{0, 1\}$ as the presence or lack of parts of type λ.

The first modification to our algorithm ensures that a part which we lack is not considered "placeable". We introduce another boolean variable

$$\xi_s(v) = (n(\lambda_v) = 0) \tag{6.1}$$

and then redefine our constraint weight to include this variable.

$$\phi_c(v) = \begin{cases} 0 & \xi_p(v) \vee \xi_r(v) \vee \xi_s(c) \\ 1 & otherwise \end{cases} \tag{6.2}$$

We now introduce two algorithms: one for when a robot receives communication that a

supply of a certain part type has been extinguished, and another when it receives communication that a part type has been replenished. A diagram demonstrating how a subtree is pruned upon running out of a particular part type can be found in Figure 6-1.

Algorithm 4 Part has been extinguished

1: Receive communication that $n(\lambda) = 0$
2: $E_p^\lambda \leftarrow \emptyset$
3: $E_p' \leftarrow E_p$
4: **for** $(v_i, v_j) \in E_p : \lambda_{v_j} = \lambda$ **do**
5: $\quad E_p^\lambda \leftarrow E_p^\lambda \cup \{(v_i, v_j)\}$
6: $\quad E_p' \leftarrow E_p' \backslash \{(v_i, v_j)\}$
7: **end for**
8: $E_p \leftarrow E_p'$

Algorithm 5 Part has been replenished

1: Receive communication that $n(\lambda) > 0$
2: $E_p \leftarrow E_p \cup E_p^\lambda$

By making these modifications, we achieve several improvements. First, since a part is not considered placeable if its supply has run out, an assembly robot will not assign positive demanding mass to those parts. This prevents delivery robots from seeking that part.

Second, by removing the physical dependency of extinguished parts from the parts they depend on, Algorithm 3 will now weight those depended-on parts less. This is desired, because part types that are currently lacking do not provide the robot additional work to perform. Placing physical dependencies does not free up more parts for the assembly robots to place. Doing this ensures that we maintain efficient ordering construction, by pretending that the parts no longer exist in the assembly blueprint.

Note that in this algorithm we did not modify the reachability graph, G_r. This is done so that even though we are ignoring the extinguished parts in the dependency graph, we still do not place parts that would prevent placing the extinguished part once the part supply has been replenished.

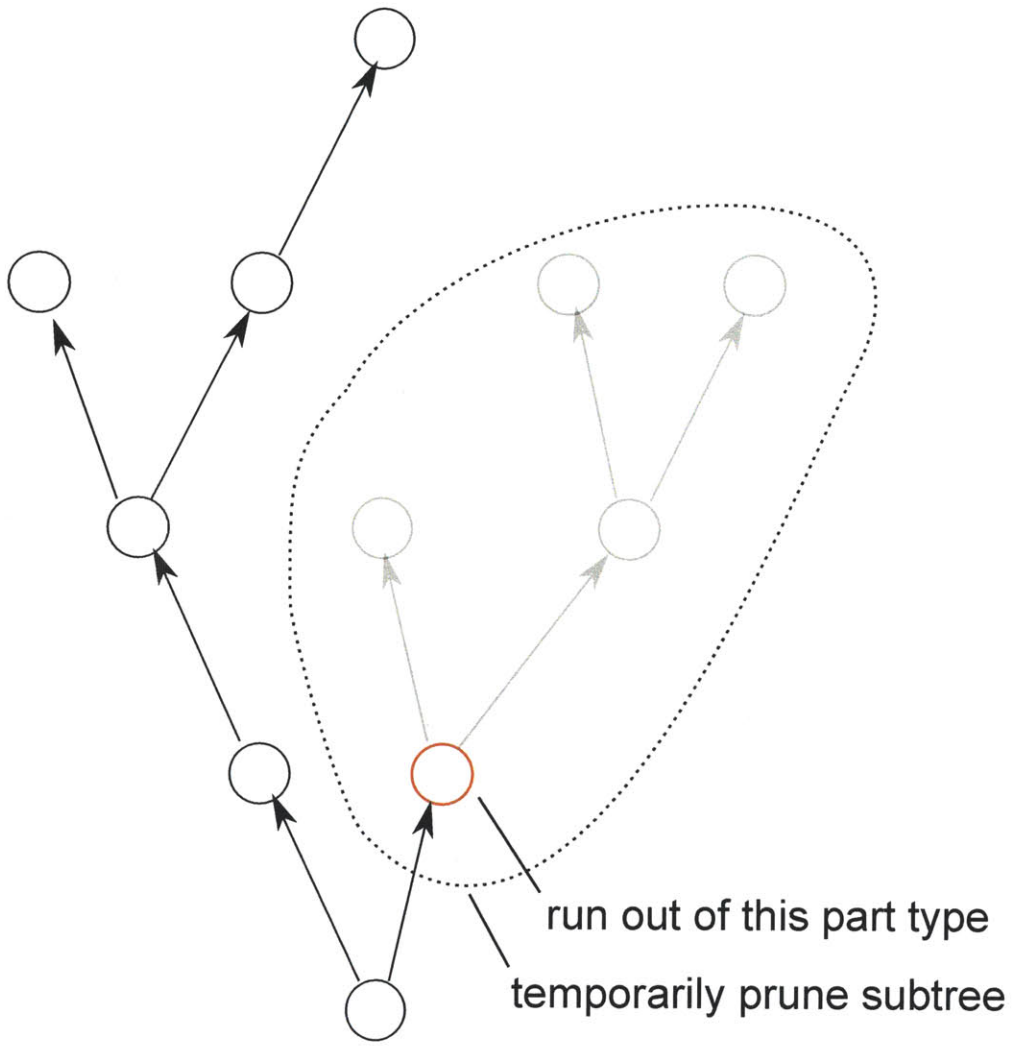


Figure 6-1: When the supply of a part type runs out, the assembly subtree with that part as the root is temporarily pruned. When the part is resupplied, the subtree is added back into the overall assembly tree.

6.1.1 Convergence

Theorem 5 *The controller, when modified by algorithms 4 and 5, will converge to a complete structure if possible.*

Proof 5 *Any time the supply of a part type runs out, the graph will be modified iff parts of that type remain to be assembled in the structure. Therefore, if the structure is possible to complete the parts will be resupplied. At that time, the controller takes its previous form.*

Therefore, if the structure is possible to complete, the modified algorithm will converge to a complete structure.

6.1.2 Runtime

Algorithm 4 requires $O(||V|| + ||E_p||)$ time to run, since it makes copies of the physical dependencies edges and loops through all nodes in the graph. Algorithm 5 simply makes a copy of the edges, so its runtime is $O(||E_p||)$.

These two algorithms are only run when a part supply is extinguished or resupplied. Since these two events rely on factors external to the algorithm, it is impossible to completely describe what effects they have on the runtime of the algorithm. However, the modifications that they make to the physical dependency graph do not change the asymptotics of the underlying algorithm.

6.1.3 Generalization

Note that in the presence of no part supply restrictions, neither algorithm will be utilized during assembly. Therefore this modification can be viewed as a generalized version of the original algorithm that takes part supply into account.

6.2 Decentralized Scheduling Algorithm with Non-Uniform Assembly Times

Our previous algorithms have assumed that all assembly operations take the same amount of time to accomplish, which is rarely true in practice. For example, one can imagine an assembly where one task is twice as valuable to complete in order to maximize parallelism and efficiency; our prior algorithms would choose the former task to complete first. However, if the first task takes three times as long to complete, then the second task is actually more desirable to complete first. It will make additional work available sooner.

Once we introduce the concept of assembly time, we are no longer interested in the ability of a task completion to make work available; instead, the quantity of interest is a task's ability to make work available divided by the amount of time it takes to complete the task. Mathematically, we introduce this into the algorithm by altering the scoring function. It now takes the form:

$$score(f(\cdot), X) = \sum_{x \in X} \left(2^{\frac{f(x)}{\tau_x}}\right)^{-c_X}. \tag{6.3}$$

where τ_x represents the amount of time required to complete task x. In the algorithm the function $f(x)$ represents the number of other nodes that will be affected by completing a task (for example, by fulfilling physical dependencies). By dividing this by the time required to complete that task, we now ensure that we weight assembly tasks according to their actual value to the assembly process.

6.2.1 Convergence

Theorem 6 *The controller, when modified by equation 6.3, will converge to a complete structure if possible.*

Proof 6 *We have already proved that the original controller will converge to a complete structure if possible. A part's mass has only a simple linear relationship on the score function. Assuming the time to complete an assembly task, τ_x, is a positive number, the*

sign of the score function is not affected. Therefore the sign of every part's mass is equally unaffected. The order of part placement may change, but a placeable part will not become unplaceable and vice versa. Therefore the structure will converge if originally possible.

6.2.2 Runtime

The addition of an assembly time term to the scoring function does not affect computational complexity or runtime in any significant way.

6.2.3 Generalization

In the case of uniform assembly times such that $\tau_x = \tau$ is a constant, the equation 6.3 can be reformatted:

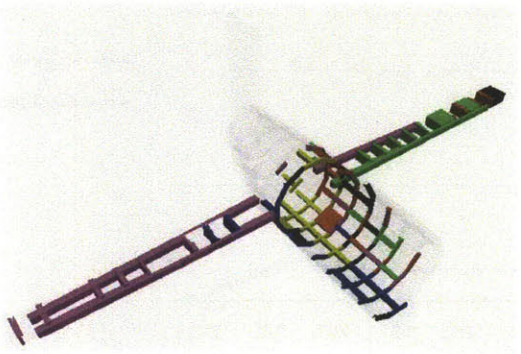
$$score(f(\cdot), X) = 2^{-\frac{cX}{\tau}} |X| \sum_{x \in X} \left(2^{f(x)}\right)^{-cX}. \tag{6.4}$$

As such, the scores and therefore the part weights would be linear scaled version of the part weights from the original algorithms. As overall scaling does not influence part order, the assembly order would remain the same. We can therefore view this as a generalized version of the original algorithm that takes assembly time into consideration.

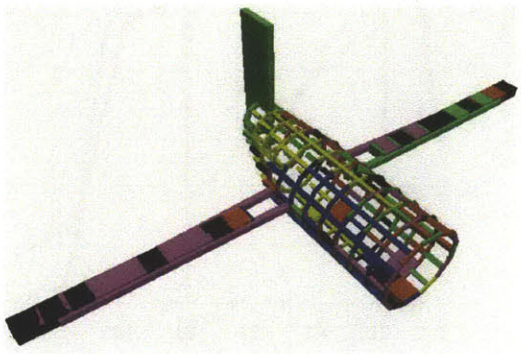
We can also combine this modification with the above part supply algorithm, to make a fully generalized version that takes both assembly time and part supply into consideration.

6.3 Simulations

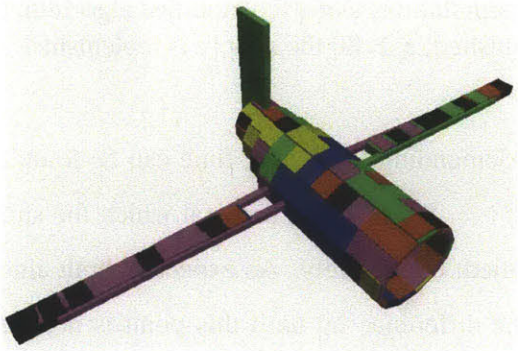
To test the effectiveness of the part supply modification to our algorithm, we ran it on our airplane simulation seen in Figure 6-2 using six assembly robots and six delivery robots. In order to evaluate its effectiveness with respect to part supply, at t=20 the supply of plane wall panels is extinguished. At t=80 the supply is replenished so that the assembly can be fully constructed. The simulation was run twenty times: ten times using the original algorithm, and ten times using the modified part supply algorithm. The results from each set of ten were averaged together.



(a) At $t = 20$ the fuselage panels have run out. Much of the structure is left to complete.



(b) At $t = 80$, the fuselage panels are resupplied. The assembly robots have constructed much of the framework of the fuselage, but were unable to place any fuselage panels in the past 60 timesteps.



(c) The plane has been fully assembled at $t = 102$ with the resupplied fuselage panels.

Figure 6-2: The plane assembly used in simulation contains a fuselage, two wings, and a tail section, each of which is composed of many individual parts. The parts are color-coded to indicate which robot of the six assembly robots placed each part (e.g., red parts were placed by robot 1, green were placed by robot 2). There are a variety of structural dependencies between parts, making the construction order complex. The plane is shown at (a) $t = 20$, (b) $t = 80$, and (c) completion at $t = 102$.

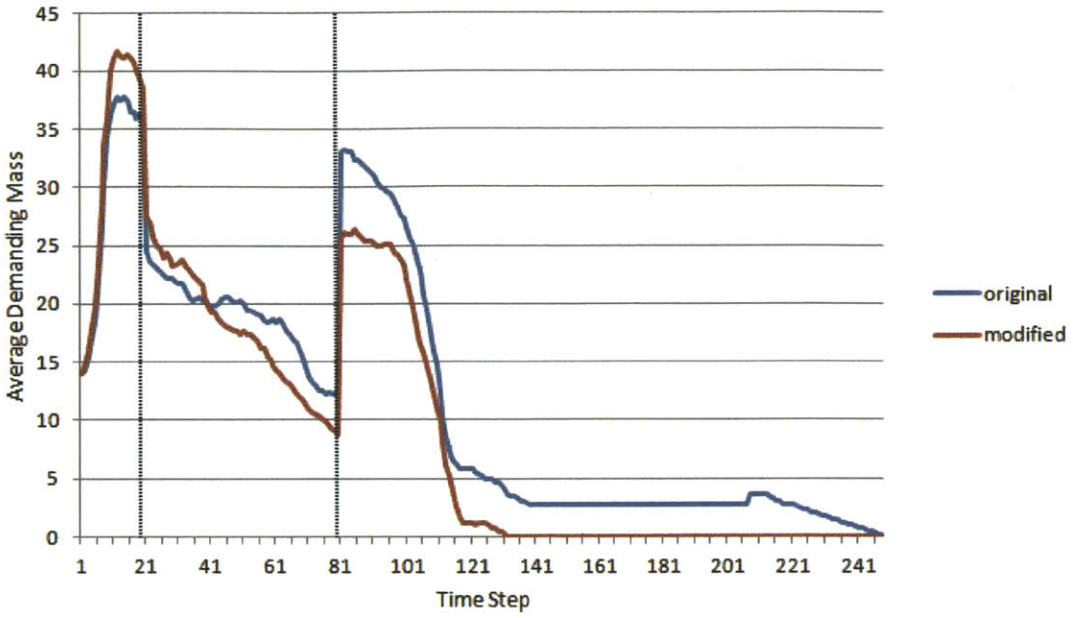


Figure 6-3: The average demanding mass over time of ten simulations using the original algorithm (blue) and ten simulations using the modified algorithm (red). At t=20 the supply of plane panels is extinguished; at t=80 the supply is replenished.

A graph of average demanding mass over time can be found in Figure 6-3. The vertical lines at $t = 20$ and $t = 80$ show the times at which the supply of plane panels was extinguished and resupplied, respectively. As expected, both algorithms perform roughly the same until t=80. The difference up until this point is not the amount of work being done, but that with the modified algorithm the robots are more intelligently choosing *which* work to do in order to efficiently parallelize the remaining work once the part supply is replenished.

It follows that the behavior diverges after $t = 80$. The original algorithm produces a slightly larger spike in available work - this is expected, since the unmodified weighting function would have caused the dependencies of plane panels to be assembled. Thus, when panels are resupplied there is a large and immediate need for them. However, under the original algorithm, there are undiscovered bottlenecks in the assembly process that have not been addressed. This produces a much longer overall completion time.

In contrast, the modified algorithm weighted the dependencies of the plane panels low

(since they did not free up additional work to complete), and instead focused on other sources of bottlenecks. This leaves a large amount of available, parallel work to be completed. Once the plane panels are resupplied, the robots can place the panels' dependencies and the panels themselves. The modified algorithm finishes the construction task much faster than the original algorithm.

This is an instance in which the differences between the two algorithms are especially highlighted, because the structure has areas of potential bottleneck and there are relatively many robots working simultaneously. We expect that this mimics the structures and working environments of real applications, and thus this simulation accurately represents the benefits of the modified algorithm.

Chapter 7

Disassembly with Part Ordering

Using the same structures and logic as the constraint-aware assembly process, we can provide a controller to disassemble a structure in a similar fashion. However, we have to redefine our paradigm of "assembly" and "delivery" robots. For this task, the assembly robots will be disassembling the structure, while the delivery robots will be picking up discarded parts and delivering them out of the construction site, perhaps back to the part cache. From here on these tasks will be referred to as disassembly and clearing, respectively. The redefined algorithms are designed to mimic the structure of the assembly algorithm in Chapter 5, while integrating the stochastic delivery properties of Bolger, et al[20].

Algorithm 6 Clearing Algorithm

1: **loop**
2: Move to random assembly robot r
3: Listen for demanding mass of all robots in communication range
4: Move to robot s with highest demanding mass
5: Clear part from s
6: **end loop**

Additionally, we need to redefine our concept of part's demanding mass. Now parts that have been placed have mass, whereas parts that have been disassembled and cleared have 0 mass. Between any two parts with non-zero mass, the part with higher mass is given priority in disassembly. Given this planning algorithm, the mass function $\phi(\cdot)$ still dictates the order in which parts are disassembled.

To begin adapting the mass function, we must first redefine our disassembly criteria.

Algorithm 7 Assembly Algorithm

```
1: Start partition algorithm (Alg. 1)
2: loop
3:    Calculate highest priority vertex v from φ(·)
4:    remove v and signal neighbors
5:    for u ∈ all children and parents of v do
6:       update φ(u) (Equation 7.7)
7:       for w ∈ all children and parents of u do
8:          update φ(w)
9:       end for
10:   end for
11:   yield until clearing
12: end loop
```

Since we no longer need to worry about reachability or part availability, we can capture this in a single variable.

$$\phi_c(v) = (deg^+_{G_p}(v) \neq 0) \tag{7.1}$$

This indicates that a part will not be removable if its outdegree is anything but 0. That is, we should not remove a part if there are other parts still depending on it for physical stability.

Because our mass function maintains the same form as the mass function for assembly, it retains the same desired properties. One of these is that it will terminate if the problem is solvable (which it is, assuming that the assembly was constructed using the same blueprint). This is sufficient to have a system that will disassemble a structure while maintaining stability, however with binary mass disassembly order will be essentially random.

The remaining mass functions allow behavior to be tuned to tend towards disassembly that allows for better parallelism.

To help to remove parts such that we first maximize the number of parts still available to be removed (i.e., remove parts that many other parts are supporting), we can rank parts first by the number of physical dependencies they have. We can represent this ranking with the *score* function from Chapter 5:

$$\phi_p(v_i) = score(deg^+_{G_p}, \{v_j | (v_i, v_j) \in E_p\}) \tag{7.2}$$

Similarly, we would like to remove parts that cause bottlenecks. By rating parts by the number of different ways to reach their children we can place preference toward removing parts around high-traffic paths. We also would like to tend toward disassembling parts in harder-to-reach locations last, so we need to define a slightly more complex test function $g(v_i) = \max_j(deg^-_{G'_r}(v_j)) - deg^-_{G'_r}(v_i)$.

$$\phi_r(v_i) = score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{7.3}$$

Finally, in combining these three measures of mass, 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.

$$\beta = \frac{\max_{v_i}(deg^-_{G_p}(v_i))}{\max_{v_i}(deg^+_{G'_r}(v_i))} \tag{7.4}$$

$$\gamma \in \{-1, 0, 1\} \tag{7.5}$$

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 7.3, which will normalize it to resemble the physical dependency function:

$$\phi'_r(v_i) = score(g', \{v_j | (v_j, v_i) \in E'_r\}) \tag{7.6}$$

We can now combine equations 7.6, 7.2, and 7.1 to define our combined mass function for use by the controller.

$$\phi(v_i) = \phi_c(v_i)(\phi'_r(v_i) + \phi_p(v_i)) \tag{7.7}$$

7.1 Runtime

Upon the removal 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 removed.

7.2 Convergence

Theorem 7 *The controller outlined in algorithms 6 and 7 will converge to a completely disassembled structure if possible.*

Proof 7 *The structure was assembled using the reverse algorithm, and relying on the same underlying mass function and DAGs. The mass function gives positive mass to all vertices with no remaining parents, which by definition describes and follows a valid topological ordering of G_p, and will therefore converge without violating physical constraints.*

Chapter 8

Implementation

8.1 Mobile Manipulators

The platform on which we have chosen to implement our algorithms is a team of KUKA youBots. The youBot, seen in Figure 8-1, consists of a holonomic base capable of omnidirectional movement and a five degree-of-freedom arm with two-finger gripper[35]. The robots are equipped with an onboard PC running Ubuntu Linux, giving flexibility of software choices. The mini ITX PC board also contains embedded Wifi to allow the robots to communicate with one another, although we have augmented them with Netgear WNCE2001 Wifi adapters for increased communication integrity.

8.2 Localization

Localization for the youBots is provided by a 12-camera Vicon motion capture system, which can track position and orientation to millimeter and milliradian precision respectively. Retroreflective markers in unique patterns allow the Vicon system to identify marked robots in the workspace.

In our experiments, the robotic base and manipulator were separately marked. The base was tracked for navigation, collision avoidance, and rough navigation toward a goal location. The arm was tracked for fine position adjustments to allow for precise manipulation.

These poses of both the bases and arms are broadcast wirelessly to the robots at 10Hz

Figure 8-1: Side view of the KUKA YouBot. The holonomic base allows for omnidirectional movement, while the five d.o.f. arm provides a usable workspace in front of, to the side of, and on top of the robot. The spherical reflective markers can be seen on both the base and manipulator for accurate localization.

using the tf interface for ROS, as described below.

8.3 Software and Communication

The software architecture runs within the Robot Operating System (ROS). There are several nodes, depicted in Figure 8-2, that run simultaneously. At the lowest level, there are hardware-specific nodes to control the robotic arm and base through ROS wrappers for the youBot driver. These in turn are given commands by the planner, which directs the overall flow of the assembly process. The planner is in constant communication with the blueprint node, which maintains the state of the assembly process and the goal structure. The blueprint node coordinates with the partitioner node, where the heart of the algorithm exists. The partitioner ensures that work is being evenly split among the robots, and ensures an efficient assembly order. Finally a Vicon node interacts with the Vicon motion capture system to provide position information to the partitioner and planner, which these respectively use to spatially partition the work and issue execution commands appropriately.

Our system takes advantage of the distribution and communication infrastructure in ROS. All nodes are run in a decentralized manner on the appropriate robot (with the exception of the Vicon system, which is necessarily centralized). Communication is performed through ROS channels or "topics". All nodes are designed to function successfully with an arbitrary number of robots, although the experiments described here will only focus on two in order to demonstrate the specifics of the system.

8.4 Blueprints and parts

Structures are specified using the YAML markup language. A blueprint file is a list of parts, each of which contains a unique identifier, a pose in the target structure, and any dependencies the part may have or provide. Using this simple but robust description, an arbitrarily complex structure can be specified.

As a simplified structure, we decided to build squares in an approach similar to how a log cabin is constructed. Two parallel parts are stacked on alternating sides. Two layers of

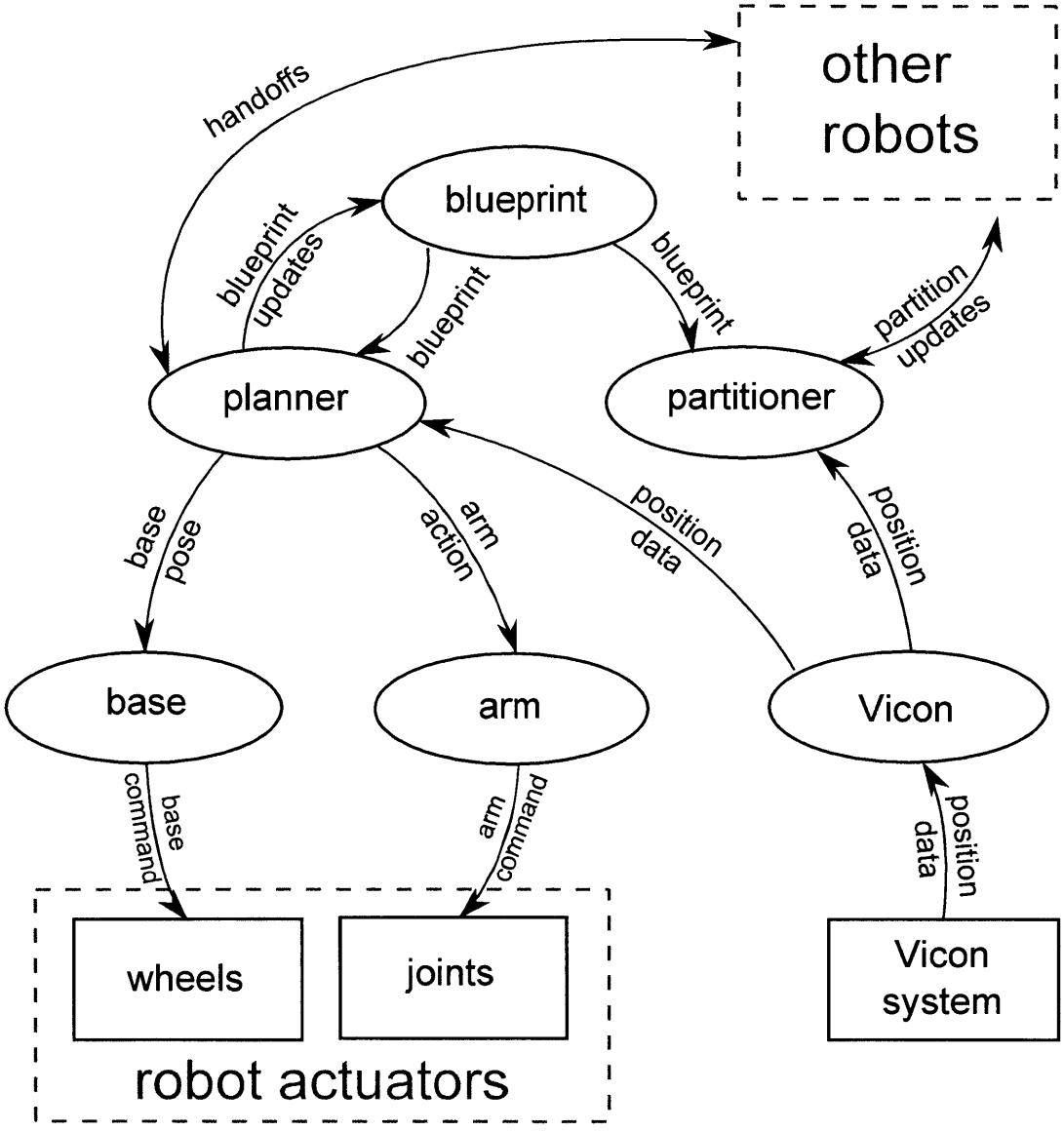


Figure 8-2: System architecture and information flow. Each oval represents a separate ROS node, and the arrows indicate messages being passed between nodes (or in some cases, between robots).

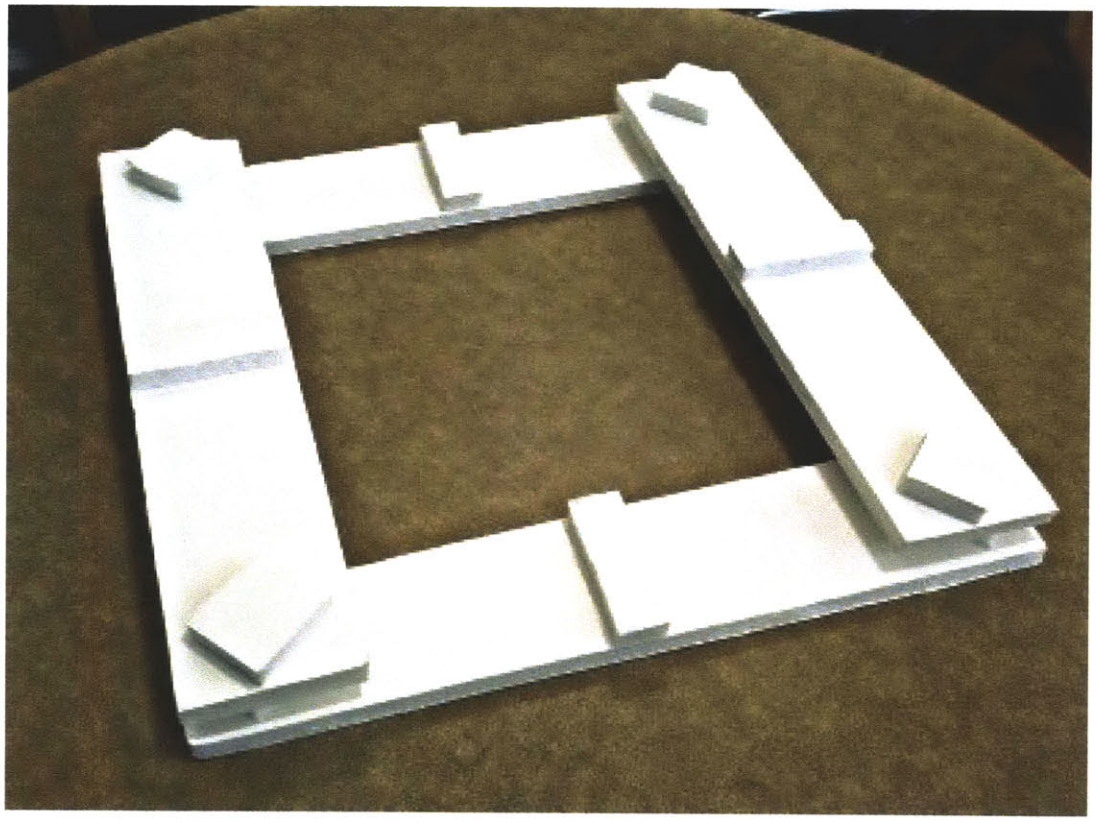


Figure 8-3: Our test parts for the main algorithm, arranged in a simple two-layer "log cabin" design. Each part has a gripping area and two diamond-shaped supports, one on each end.

this structure are shown in Figure 8-3. The parts are constructed out of lightweight foam, and consist of a long flat section with two diamond-shaped supports on either end and a raised gripping area in the center.

To demonstrate our part supply algorithm, we will make use of styrofoam cubes to build a staircase. For part heterogeneity, we have split the cubes into two groups and color-coded them accordingly. The red cubes represent the finished "tops" of the staircase, whereas the blue cubes represent the unfinished "foundation" of the staircase, perhaps made out of concrete in a real assembly process. The assembled staircase with both types of parts can be found in Figure 8-4.

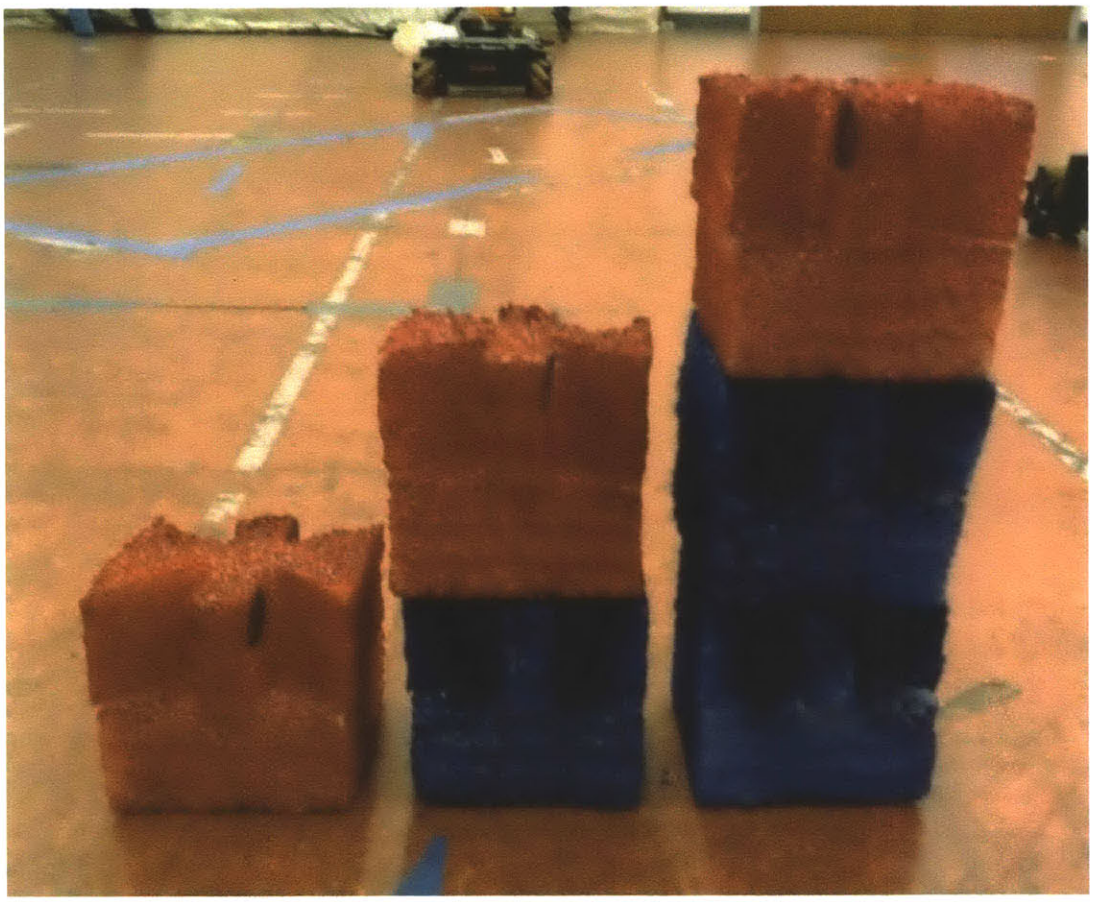


Figure 8-4: Our test parts for the part supply algorithm. Parts are divided into top parts (red) and foundation parts (blue). Each part is a styrofoam cube with slots cut into the top to allow the youBots to grip them.

8.5 Navigation

A motion planner described in [36] is employed for rough navigation. It uses a combination of a grid-based global planner and an equivalence class-based local planner to calculate a smooth and safe path to the goal. A static map is employed to assign high-cost areas to the part cache and target construction area, such that the robots only approach them as needed to retrieve or place parts.

For any navigation task, the robots switch to a second controller upon reaching the general vicinity of the goal. This controller uses simple proportional feedback control on the base velocity given the position of the arm in order to precisely position the robotic manipulator in the correct position for the next step in the task. This ensures maximum accuracy and minimal errors over the entire assembly process.

8.6 Manipulation

In these experiments neither the parts nor the manipulators were equipped with any sensing or vision. Additionally, the parts were not marked with the retroreflective markers. Instead, the robots relied on the parts being accurately and precisely placed at a predetermined source location. A precise handoff must also be executed between the delivery robot and assembly robot. If a delivery robot does not carefully deliver or report the coordinates of a component part, the assembly robot will be unable to retrieve it and place it on the assembly.

To assist in manipulation, we added small sandpaper discs on the inside of the youBot fingers. This prevented the parts from slipping or changing orientation. We also added these sandpaper discs onto the colored cubes used in the part supply algorithm, to further prevent slipping.

8.7 Ordering

As described in Chapter 5, our ordering relies on two DAGs G_p and G_r represented in the assembly blueprint. These respectively indicate the physical dependencies and reachability constraints of the target assembly. Given the structure of these graphs and a scoring function, we can weight individual parts by their contribution to the parallelism and efficiency of the overall task. Assembly robots therefore choose the parts with the largest weight to assemble next, ensuring that the robots are greedily opening up the most future work to be done.

To save complexity and time over trials, the ordering of parts in the assembly process is calculated prior to execution. In practice this could be performed either offline or online, depending on the requirements of the task and the complexity of the assembly blueprint. The algorithms and approaches are the same regardless of the choice.

8.8 Delivery

After retrieving a part from the source, a delivery robot begins listening to broadcasts coming from assembly robots. These broadcasts contain each assembly robot's demanding mass for each part type. The delivery robot chooses the assembly robot with the highest demanding mass for the part type it has retrieved, and begins moving toward that particular robot.

Once the delivery robot is within appropriate range of the assembly robot, it passes the part. An image of this handoff occurring can be found in Figure 8-5. Again, since the parts and grippers are not equipped with any sensors, the robot must be precise in the handoff.

8.9 Assembly

Finally, assembly is performed using the same precision techniques in order to produce a stable, accurate placement of the individual parts. Assembly repeats until the structure is complete or parts are no longer delivered via the delivery robots.

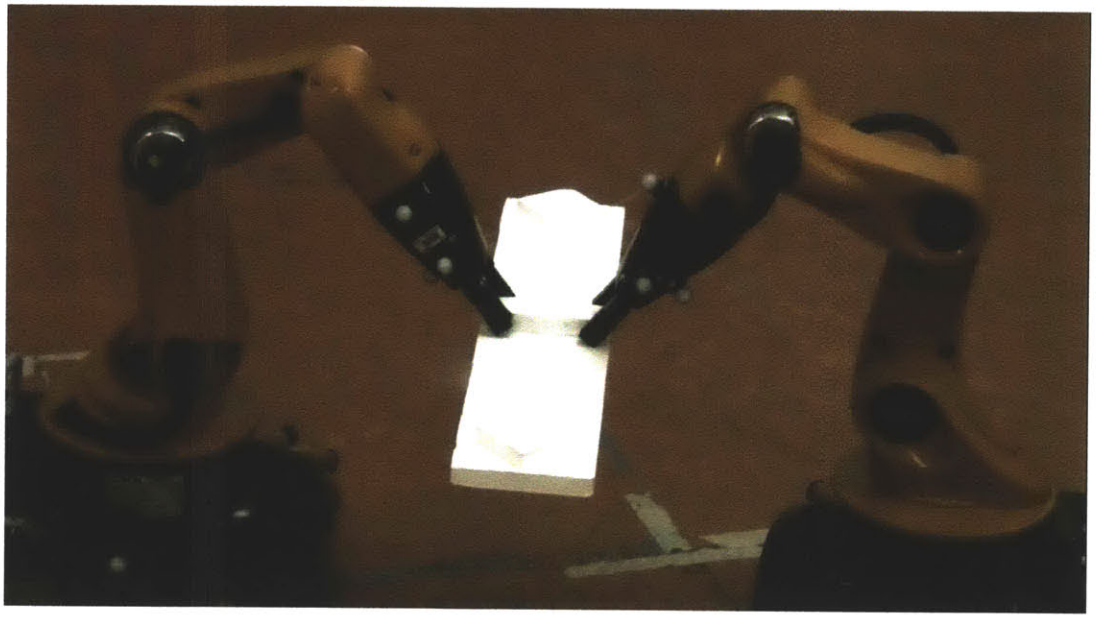


Figure 8-5: Image of a delivery robot performing a delivery. Since the robots do not have vision and the assembly parts are not tracked by the Vicon system, the handoff and communication must be precise.

8.10 Differences Between Theory and Implementation

Our experimental platform utilizes a full implementation of our theory, with differences as noted in Table 8.1. The main differences are a result of the scale of the assembly task. Our algorithms are generalized for use on an arbitrarily large number of robots. It is therefore essential that communication be limited to neighbors, and that techniques such as gradient descent are used to find local maximums of demanding mass. In our experiments, we use a maximum of four robots. In this case all robots are each others' neighbors, so we relax these explicit constraints. In our examples, the effect is the same as if the constraints were still in place.

The other difference is that the part ordering is calculated offline instead of in real-time during the assembly process. This was done for simplicity and consistency between trial runs.

Aspect	Theory	Implementation
Delivery	Gradient descent to local assembly robot with highest demanding mass	Delivery to robot with highest demanding mass
Assembly	Abstract assembly task	Concrete assembly task: part placement
Partitioning	Discrete partitioning with hull vertex swap	Discrete partitioning with hull vertex swap
Ordering	Part priority calculated in real-time based on dependencies, reachability, part supply, and assembly time	Part priority calculated offline based on dependencies, reachability, and part supply
Communication	Neighbors only (single hop)	All robots

Table 8.1: Summary of differences between our theoretical algorithms and system implementation.

Chapter 9

Experiments

9.1 Two-Dimensional Experimental Results

9.1.1 Overall Results

Ten full assembly trials were attempted with two robots and a blueprint of a simple square structure seen in Figure 8-3. Each trial consists of four delivery-assembly iterations. Each iteration involves a delivery robot retrieving a part from the source, delivering it to the assembly robot, and transmitting its location to the assembly robot; the assembly robot in turn retrieves the part from the broadcast location, moves to the assembly location, and places the part on the assembly. There are therefore many possible points of failure, especially given that the parts are retrieved and manipulated without vision.

The results are summarized in Table 9.1. Assembly and delivery utilization is defined as the percent of time that the assembly or delivery robot, respectively, was busy with a task as opposed to waiting.

Over all ten trials, there was only one failure. This occurred when the assembly robot dropped a part after retrieving it from delivery but before placement on the final structure. This was likely due to a low battery which caused there to be insufficient force in the gripper; the battery was replaced for the last two trials and no further issues were seen.

In the remaining nine trials, there were no significant failures. All part retrievals and handoffs were performed successfully. There were no dropped parts, collisions between

Trial	Runtime (M:SS)	Assembly utilization	Delivery utilization	Success (Y/N)
1	6:04	83.4%	64.5%	Y
2	5:59	86.5%	65.9%	Y
3	6:09	80.7%	64.6%	Y
4	6:22	86.5%	63.7%	Y
5	6:24	87.8%	63.6%	Y
6	5:58	86.8%	63.7%	Y
7	6:05	88.3%	65.6%	Y
8	6:20	87.0%	66.2%	N
9	6:12	87.6%	64.4%	Y
10	6:17	85.9%	65.1%	Y
Avg	6:11	86.1%	64.7%	90%

Table 9.1: Summary of two-robot assembly trials for a square.

robots, or inadvertent contact with any parts. The structure was completed in all nine trials.

9.1.2 Runtime and Efficiency

The average runtime over the trials was 6 minutes and 11 seconds, with a standard deviation of 9.6s. An activity log for a typical run can be found in Figure 9-1. The solid bars indicate when each robot was busy with a task, and the lack of a bar indicates that the robot was waiting. The chart shows that the assembly robot was busy for nearly the entire assembly process, whereas the delivery robot was busy for significantly less time. Indeed, over all trials the average assembly utilization was 86.1% whereas the average delivery utilization was 64.7%. This suggests that the optimal assembly to delivery robot ratio for this task is roughly two to three, although it is unlikely that the marginal utilization of additional robots is strictly linear. Additional testing would need to be performed in order to validate a choice of ratio.

Surprisingly, there is very little correlation between the assembly utilization and delivery utilization (r=0.08). This suggests that the efficiency of one robot is not impacted as much by how much it is waiting for another robot to finish a task, but is perhaps influenced by how quickly it is able to finish its own work - and therefore how much it must wait for the next step in the process.

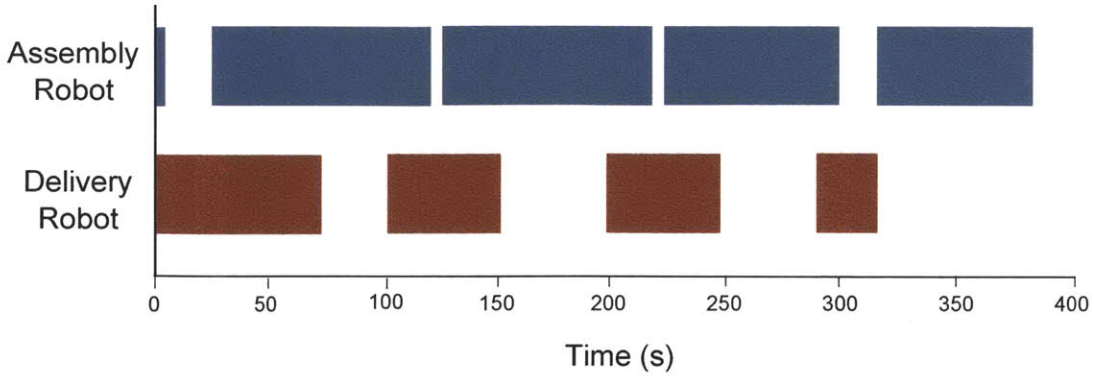


Figure 9-1: Robot activity over time in trial 4. Solid blocks of color indicate when a robot was busy with a task, as opposed to idle.

Trial	**Runtime** (M:SS)	**Assembly utilization**	**Delivery utilization**	**Success** (Y/N)
1	19:32	83.1%	58.6%	Y
2	20:09	81.3%	69.7%	Y
3	22:17	84.5%	51.5%	Y
Avg	20:39	83.0%	59.9%	100%

Table 9.2: Summary of two-robot assembly trials for a tower.

9.2 Three-Dimensional Experimental Results: Two Robots

9.2.1 Overall Results

Our next task introduces a new blueprint to investigate the robustness of the system for constructing a three-dimensional structure. The new blueprint uses the same "log cabin" style, but has six layers instead of two. An example of this structure can be seen in figure Figure 9-2. This new assembly process requires a total of twelve delivery-assembly iterations per trial. There can also be stability complications, as parts form the foundation for more parts; an early misplacement can cause the entire structure to fall.

Otherwise, all aspects of the assembly process are the same as described in Section 9.1. The results from the three-dimensional construction are summarized in Table 9.2.

Over all three trials, there were no significant failures. All part retrievals and handoffs were performed successfully. There were no dropped parts or collisions between robots. A few times the assembly robot brushed the structure, but in none of the trials were the parts

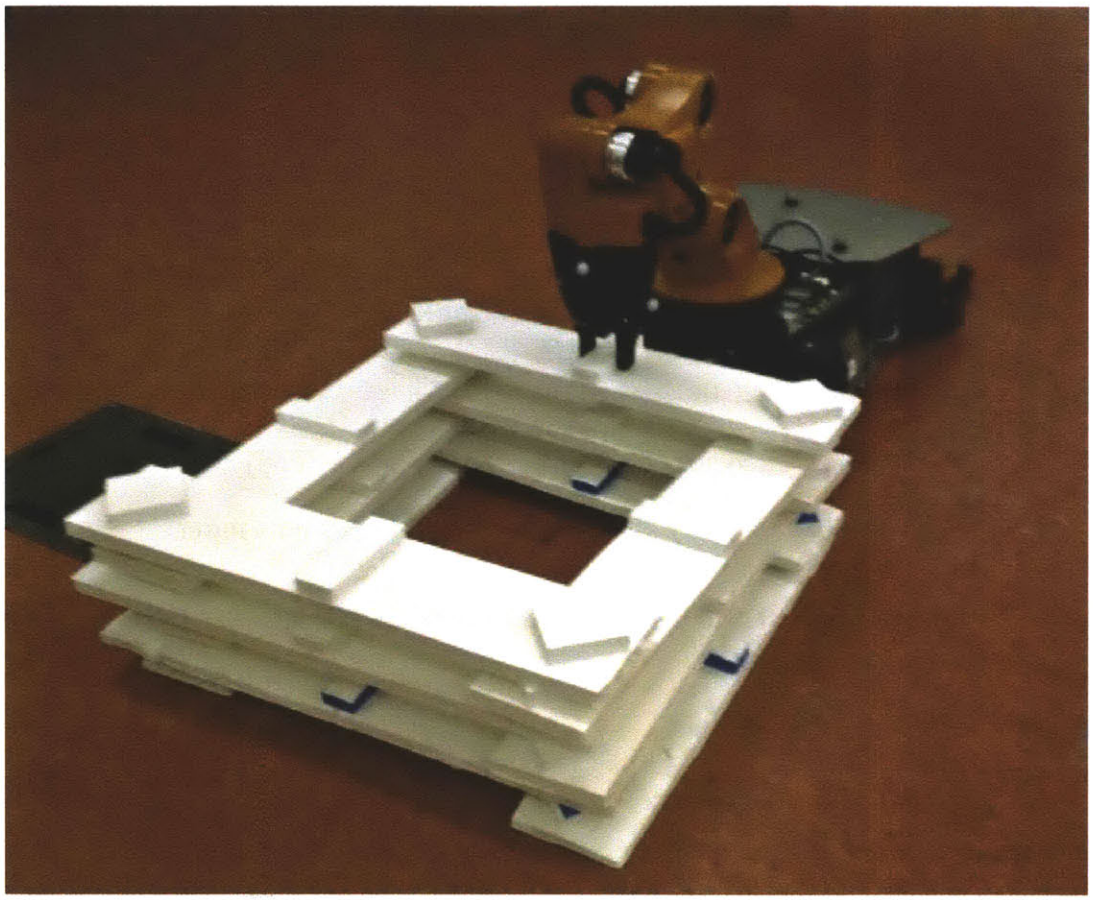


Figure 9-2: An assembly robot places the final part on the three-dimensional tower. The tower is composed of six layers of the log cabin construction, or three of the simple squares from Section 9.1. This tower is the result of trial #1 from Table 9.2.

Trial	Runtime (M:SS)	Assembly 1 utilization	Assembly 2 utilization	Delivery 1 utilization	Delivery 2 utilization	Success (Y/N)
1	9:16	84.8%	90.1%	95.4%	86.3%	Y
2	8:53	88.9%	87.3%	96.6%	96.1%	Y
3	8:46	89.9%	88.7%	93.0%	94.2%	Y
4	10:29	83.3%	86.8%	90.9%	94.4%	Y
5	12:34	87.8%	87.9%	82.0%	92.7%	Y
6	10:33	87.2%	87.9%	96.7%	93.9%	Y
7	8:42	89.7%	87.2%	97.9%	94.0%	Y
8	11:23	93.0%	84.9%	81.5%	93.7%	N
Avg	10:05	88.1%	87.6%	91.8%	91.9%	87.5%

Table 9.3: Summary of four-robot assembly trials for a tower.

knocked out of place or moved in such a way that compromised the structure. In all three trials, the structure was stably completed.

9.2.2 Runtime and Efficiency

At 20:39, the average runtime over the trials was roughly three times that for the two-dimensional square, as expected. Delivery and assembly utilization were also similar. There were not enough trials to comment on overall trends or correlations.

9.3 Three-Dimensional Results: Four Robots

9.3.1 Overall Results

To increase the complexity of our task, we then constructed the tower using a total of four robots. Two robots were designated as delivery robots, and the other two as assembly robots. All other aspects of the assembly process were the same. The results of these trials are summarized in Table 9.3.

In all of the trials except the last, the structure was completed successfully with no errors. On the last trial, there was a timing issue where both assembly robots attempted to place parts simultaneously and the parts collided. The rest of the assembly process continued as planned, although the two collided pieces had to be re-placed manually to support the remainder of the structure. This may have also led to variability in the metrics

for that trial, as both assembly robots experienced difficulty placing these parts accurately due to the other robot's movements. The delivery robots needed to wait for this interaction to finish before delivering more parts, decreasing their utilization and lengthening the time of the overall process.

9.3.2 Runtime and Efficiency

The average runtime for these trials was less than half than that for the two-robot trials, indicating that there must have been an increase not only in parallelism but also in utilization. Indeed, average assembly utilization across both robots was marginally higher at 87.9% and average delivery utilization was much higher at 92.8%. This suggests that the relationship between number of robots and the amount of work is not simply linear, but perhaps more complicated. It would seem that for large-scale assemblies the number of delivery robots should be roughly equal to the number of assembly robots, with perhaps slightly more assembly robots to raise the assembly utilization above 90%.

9.4 Experimental Results with Part Unavailability

In order to test our part unavailability algorithm, we used a blueprint of a simple set of stairs. There are three steps, and each step has one, two, or three parts respectively. The top part on each step is a "finished" top part, which we represent with a red cube. All of the cubes underneath top parts are "foundation" parts, represented with a blue cube.

We ran three trials of the assembly process in which parts did not run out. As can be seen in Figure 9-3, the assembly robot first completes the shortest stair, then the middle stair, and then the third stair. We then ran three trials where after the assembly robot places the first red part, the supply of red parts runs out. The assembly robot adapts and places all of the foundation parts, and then when the red parts are resupplied the assembly robot completes the structure. This sequence can also be found in Figure 9-3.

A summary of our results for the two sets of trials can be found in Tables 9.4 and 9.5, respectively. As can be seen, there was no significant difference in time or utilization

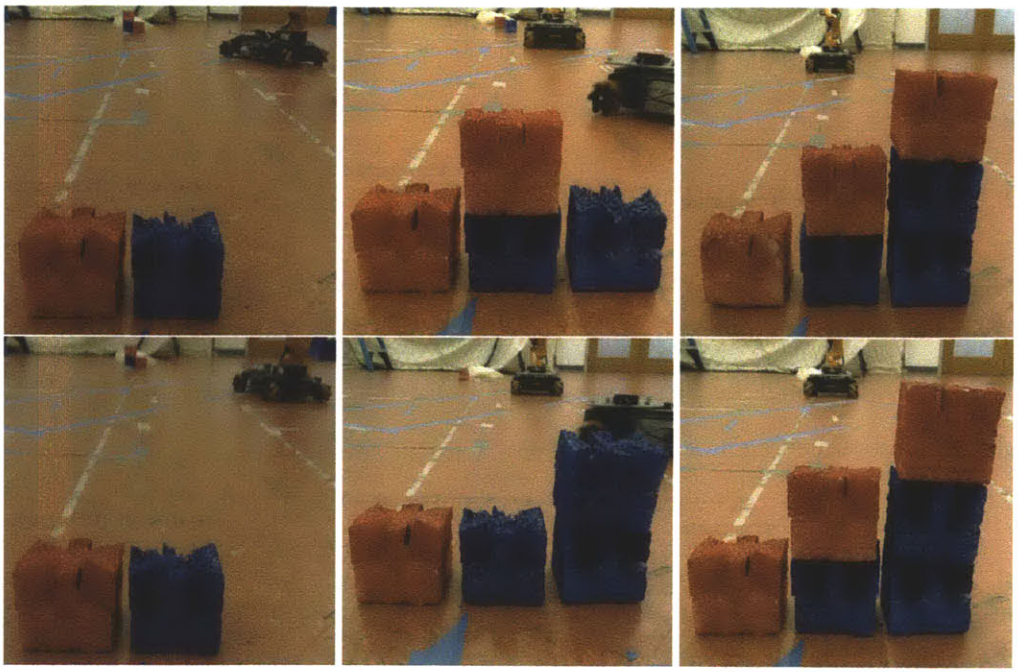


Figure 9-3: Assembly sequence when no parts run out (top three images) and when the top/red cubes run out after one has been placed (bottom three). Even with the part supply running out, the robots continue to work and ultimately complete the structure after part supply has returned.

Trial	**Runtime** (M:SS)	**Assembly utilization**	**Delivery utilization**
1	8:44	91.5%	96.5%
2	8:56	88.6%	95.8%
3	8:35	87.7%	94.8%
Avg	8:45	89.2%	95.7%

Table 9.4: Summary of two-robot assembly trials of stairs. In these trials, part supply never ran out.

Trial	**Runtime** (M:SS)	**Assembly utilization**	**Delivery utilization**
1	8:18	87.3%	94.9%
2	8:55	87.4%	98.0%
3	9:28	89.2%	97.0%
Avg	8:54	88.0%	96.6%

Table 9.5: Summary of two-robot assembly trials of stairs. In these trials, part supply of the red parts ran out after the first placement, but was resupplied after three more placements.

between the two trials. This indicates that the assembly robots adequately adapted to the loss of part supply and did not lose efficiency.

9.5 Handoff Experimental Results

In our later experiments, deliveries were achieved by direct handoff between the delivery robot and assembly robot. That is, instead of setting down the part during the delivery, the delivery robot presented the part to the assembly robot that then retrieved it directly from the delivery robot manipulator.

Over our recorded trials, there were 114 attempts of such a handoff with the log cabin parts, 112 of which succeeded. This is a success rate of 98.2%. Assuming that each handoff is an independent event with this probability of failure, the probability of a successful run (no errors in handoff) are 83.8%.

The first failure was due to high network latency, such that the location of the delivery robot was miscommunicated to the assembly robot. The second failure was due a loose grip on the part by the assembly robot; while the assembly robot navigated to the target location, the orientation of the part slipped and was therefore not placed correctly.

Chapter 10

Conclusion and Future Work

10.1 Summary of Contributions

In this work, we presented a number of algorithms for the efficient assembly of structures using a distributed robotic system. In order to equally partition work across a team of robots, we first presented a distributed algorithm for equal mass partitioning. It begins with a Voronoi partitioning of the points, and then tests swapping vertices between the hulls of neighboring partitions to maximize a heuristic function. The algorithm is fully decentralized, and relies only on local knowledge to perform these swaps. It is guaranteed to converge to a local maximum.

We then presented an algorithm for the ordering of assembly tasks to increase the parallelism and efficiency of the assembly process. It considers the physical dependency and reachability constraints of an assembly, and uses this information with the help of a scoring function to weight parts according to their priority in the assembly process. These weights can then be used in addition to hard constraints to develop an ordering of operations. The algorithm is guaranteed to converge to the target structure if possible, and exhibits desirable efficiency qualities.

Using this algorithm as a base, we then presented adaptations to those algorithms to consider further implications of a real-world assembly scenario. In a situation in which part supply is limited or becomes unavailable, we modified our ordering algorithm to take this into consideration and reweight assembly tasks so as to maximize parallelism once the

parts are restored. We also considered the case where assembly operations are non-uniform in the amount of time required to complete them. Our original algorithm was ignorant of true time (instead representing time in terms of algorithm steps), so the modification adapts it to remain efficient in the face of differing time constraints. Both adaptations to these algorithms retain the desirable qualities of the original ordering algorithm, including its guaranteed convergence if the structure is possible to be completed.

Under the same paradigms and assumptions, we also constructed an algorithm for the efficient disassembly of structures using a team of heterogeneous robots. Many of the techniques and qualities of the original algorithm are preserved, but require small tweaks in order to be appropriate in the context of disassembly. Again, convergence is preserved.

To demonstrate the correctness and applicability of our algorithms, we finally presented an experimental testbed on which we implemented our algorithms. The platform consisted of two to four robots, divided into two teams for assembly and delivery. It took advantage of the communications abilities inherent in the Robot Operating System, but built on top of that our partitioning and ordering algorithms, as well as the high-level planning and actuation required for actual assembly tasks. We performed a number of experiments with this platform, and presented the results as validation of our algorithms.

10.2 Future Work

Because of the multitude of technological challenges that arise from a full robotic assembly system, there remains a large amount of work in this area.

For the most part, our system ignored the challenge of perception by utilizing a Vicon motion capture system. In practice, perception would have to be an important module of any robotic assembly system. Detecting the assembly site, component parts, and target structures, as well as other robots or even humans on the factory floor, would all be crucial tasks. Additionally, specialized perception would likely be needed to monitor the specific assembly operations themselves to identify when they are complete or if they need to be repeated. Hammering a nail, for example, would require perception to know when the nail was flush with the component.

The assembly or manipulation event itself was also largely unexplored by this work. We assumed that assembly tasks were relatively simple and atomic, although not necessarily uniform in time requirements. In practice assembly operations could be arbitrarily complicated or multi-part. The grasping, maneuvering, and sensing operations involved in performing these tasks can be equally complicated or specialized.

Adapting to changes or unexpected events in the assembly process remains an active area of research. A failed assembly operation, modifications to the assembly by a human on a factory floor, or accidental damage to the target structure are all events that could occur at an active assembly site. A robust robotic platform would need to detect, analyze, and respond adequately to these situations.

We also imagine humans will play a large part in the assembly process, largely due to a dichotomy of skill. Humans are, by their nature, better at planning and/or performing certain tasks than robots, and vice-versa. A truly efficient and robust robotic platform could recognize when tasks require human input or assistance, signal for this help, and wait until it has been completed to continue. The addition of humans into the assembly process also involves a number of other considerations, including workplace safety and communication protocols.

Even within the context of assembly ordering, there is still much work to be done. Although this work attempted to analyze some of the complexities of an assembly task that could factor into efficient partitioning and ordering of assembly tasks, there are many other areas where improvements could be made. For example, if part supply is not currently extinguished but will be at a known time in the future, proper planning would account for this fact and utilize it in the ordering algorithm.

As these challenges continue to be solved, we look forward to the further integration of robotics into industrial assembly for a safer, more efficient, and more robust process.

Bibliography

[1] U. B. of Labor Statistics. (2011, Apr.) May 2010 national occupational employment and wage estimates united states. [Online]. Available: http://www.bls.gov/oes/current/oes_nat.htm

[2] ——. (2011, Aug.) Census of fatal occupational injuries (cfoi) - current and revised data. [Online]. Available: http://www.bls.gov/iif/oshcfoi1.htm

[3] ——. (2011, Aug.) Census of fatal occupational injuries - archived data. [Online]. Available: http://www.bls.gov/iif/oshcfoiarchive.htm

[4] D. Stein, T. R. Schoen, and D. Rus, "Constraint-aware coordinated construction of generic structures," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[5] 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

[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] S. E. Fahlman, "A planning system for robot construction tasks," *Artificial Intelligence*, vol. 5, no. 1, pp. 1 – 49, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370274900083

[8] 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.

[9] K. Petersen, R. Nagpal, and J. Werfel, "Termes: An autonomous robotic system for three-dimensional collective construction," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[10] N. Ayanian and V. Kumar, "Decentralized feedback controllers for multiagent teams in environments with obstacles," *Robotics, IEEE Transactions on*, vol. 26, no. 5, pp. 878 –887, oct. 2010.

[11] S. Berman, Á. M. Halász, M. A. Hsieh, and V. Kumar, "Optimized stochastic policies for task allocation in swarms of robots," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 927–937, 2009.

[12] M. A. Hsieh and J. Rogoff, "Complexity measures for distributed assembly tasks," in *Proceedings of the 2010 Performance Metrics for Intelligent Systems Workshop (PerMIS09)*, Baldimore, MD, USA, Sept 2010.

[13] A. Nease and E. Alexander, "Air force construction automation/robotics," *Automation in Construction*, vol. 3, no. 1, pp. 93 – 98, 1994, ¡ce:title¿Special Issue: 10th ISARC, May 24-26, Houston, Texas, USA ¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/092658059490037X

[14] H. Z. Chris Parker and R. Kube, "Blind bulldozing: Multiple robot nest construction." in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2003.

[15] S. Y. Lee, K. Y. Lee, S. H. Lee, J. W. Kim, and C. S. Han, "Human-robot cooperation control for installing heavy construction materials," *Auton. Robots*, vol. 22, no. 3, pp. 305–319, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10514-006-9722-z

[16] C. Schuil, M. Valente, J. Werfel, and R. Nagpal, "Collective construction using lego robots," in *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, ser. AAAI'06. AAAI Press, 2006, pp. 1976–1977. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597348.1597535

[17] A. Stroupe, T. Huntsberger, A. Okon, H. Aghazarian, and M. Robinson, "Behavior-based multi-robot collaboration for autonomous construction tasks," in *International Conference on Intelligent Robots and Systems*, 2005.

[18] 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.

[19] 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.

[20] A. Bolger, M. Faulkner, D. Stein, L. White, S. kook 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.

[21] M. Erwig, "The graph voronoi diagram with applications," *Networks*, vol. 36, no. 3, pp. 156–163, 2000. [Online]. Available: http://dx.doi.org/10.1002/1097-0037(200010)36:3¡156::AID-NET2¿3.0.CO;2-L

[22] J. Cortés, S. Martínez, 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.

[23] 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.

[24] 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.

[25] ——, "Distributed algorithms for equitable partitioning policies: Theory and applications," in *IEEE Conference on Decision and Control*, Cancun, Mexico, Dec 2008.

[26] H. Maini, K. Mehrotra, C. Mohan, and S. Ranka, "Genetic algorithms for graph partitioning and incremental graph partitioning," in *Proceedings of Supercomputing '94*. ACM Press, 1994, pp. 449–457.

[27] R. Leland and B. Hendrickson, "An empirical study of static load balancing algorithms," 1994, pp. 682–685.

[28] J. W. Durham, R. Carli, P. Frasca, and F. Bullo, "Discrete partitioning and coverage control with gossip communication," *ASME Conference Proceedings*, vol. 2009, no. 48937, pp. 225–232, 2009. [Online]. Available: http://link.aip.org/link/abstract/ASMECP/v2009/i48937/p225/s1

[29] 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.

[30] S. kook 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.

[31] S. Yun and D. Rus, "Adaptation to robot failures and shape change in decentralized construction." Institute of Electrical and Electronics Engineers, 2010.

[32] 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.

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

[34] A. Bolger, "Coordinated part delivery using distributed planning," Master's Thesis, Massachusetts Institute of Technology, CSAIL Distributed Robotics Laboratory, Jun. 2010.

[35] Locomotec. (2012, Jan.) Kuka youbot store. youbots. [Online]. Available: http://www.youbot-store.com/category/53-youbots.aspx

[36] R. A. Knepper, S. S. Srinivasa, and M. T. Mason, "Hierarchical planning architectures for mobile manipulation tasks in indoor environments," in *Proceedings of International Conference on Robotics and Automation*, May 2010.