Reinforcement Learning for Multi-Agent Robotic Construction

Nathan Melenbrink
Harvard University
Email: emelenbrink@gsd.harvard.edu

Essam El Messiri
Harvard University
elmessiri@college.harvard.edu

Serguei Balanovich
Harvard University
sbalanovich@college.harvard.edu

Abstract—**This paper explores a variety of approaches to multi-agent swarm construction of bridges and similar structures. It explores work currently being done in this field for optimizing construction and extends it to a multi-agent network of agents to determine the local rules that are necessary for a group of robots to successfully execute such constructions. The paper explores both intuitive heuristic approaches and reinforcement learning approaches. Ultimately, the research shows that the only way to introduce the complexities required to build stable and valid structures over a limited space is by combining sub-optimal strategies into an overall optimal construction. In other words, agents must not be uniform in their approach to the constructions, and any single agent's contribution should not be able to hold up the structure on its own to allow the combined construction to be well-balanced and stable. This is surprisingly difficult to achieve, but this paper presents compelling early results suggesting that using powerful reinforcement learning techniques, such counter-intuitive local rules are possible.**

I. Introduction

The primary problem addressed in this paper is the optimization of a behavior selection routine for a multi-agent collective construction swarm. The agents have no global knowledge or memory, and must make decisions based only on local information. Specifically, an agent can read the structural stresses at its current node and use this information to determine which behavior it should be following.

The system we used included struts, bots (our agents) and a base. An agent that arrives at any point has the option to either place a strut or simply traverse in a certain direction. At any point in the construction process, an agent has up to 5 possible options for where struts can be placed (**as seen in Figure 1**) and has to choose between three possible behaviors: (1) Build (laying a strut to a previous inactive node, (2) Reinforce (placing a strut that connects two existing nodes), or (3) Traverse (simply moving to a neighboring connected node). When an agent uses up their strut(s), they use the "Traverse" behavior to move down (near the ground level) in order to replenish their supply of struts.

The goal was to design a simple local rule that each bot can adopt in order to optimize, for the whole system, the construction process. The baseline topological optimization (described further on) will yield to us some optimal means of construction for a given problem and the local system will be designed to attain this result as closely as possible. To achieve this, we will optimize for a variety of parameters, introduce several modifications to the robots functionality, and evaluate the results on a variety of different problems. Below are some lists of the different directions we expect to explore in this project.

II. Motivation

The conventions of structural engineering prioritize the analysis of a structure in its finished state, often failing to consider the feasibility of the construction sequence over time. While this is slowly changing as some construction firms are starting to implement sensors for structural health monitoring on construction sites, this is essentially collecting global data to compare to a priori simulations. The notion of swarm robotics for collective construction on the other hand, requires that the swarm is able to assemble a structure without centralized control, and is resilient to environmental conditions or loss of agents. This suggests that the agents would need to have some knowledge of local structural health. While the hardware for structural health monitoring has improved greatly, the question remains how to process this information in order to build structures that are guaranteed to be stable throughout the build sequence, and are resilient in dynamic environments. Broadly speaking, the motivation for this research is rooted in the need for a type of global-to-local compiler to provide this information in order to command construction robots with a myopic knowledge of the structure they're building.

A biological analogy for this work can be found in the construction behaviors of termites, which also provided inspiration for precedent projects like TERMES [1]. It is known that termite colonies are capable of producing massive mounds, even though individual termites have only a myopic knowledge of the structure they're building. What is unclear, however, is exactly how much knowledge each termite has, or even the degree to which termites use stigmergy to organize construction.

In addition to this poorly understood natural phenomenon, there is also a shortage of relevant publications in literature. To our knowledge, there has not been any publication that combines multi-agent systems with structural sensing. The lack of prior exploration in this field is a primary motivating factor for this research. Since there are ostensibly no existing findings to which we can compare or corroborate our work, this paper attempts to establish a baseline through a broad range of trials dealing with reinforcement learning for collective construction.

III. Background

Our project fell within the scope of swarm robotics for collective construction, and drew inspiration from relevant

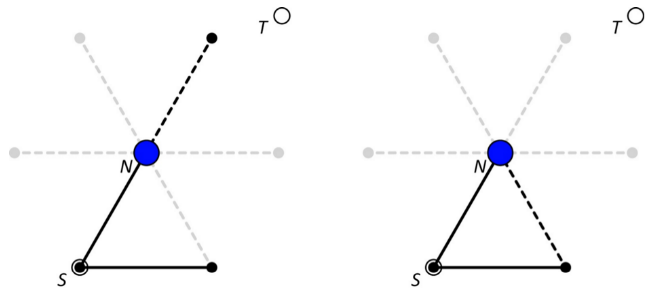


Fig. 1: Consider a global goal of building a structure from start point S to target point T. An agent arriving at point N has 5 options for strut placement. Operating according to the Build behavior (left), the agent will place the strut that brings the structure closest to point T. Alternatively, the Infill behavior (right) will dictate that the agent should place the strut that connects two existing nodes. Finally, the Traverse behavior (not shown) requires the agent to move to a neighboring connected node and reevaluate.

literature in this field, specifically the TERMES project. This was the first successful physical demonstration of collective construction, and provides formalisms for guaranteeing the successful construction of a given 3D shape by a robot swarm. After the publication of the TERMES project, Principal Investigator Dr. Justin Werfel has proposed a research agenda for collective construction using node and strut assembly as opposed to stacking blocks [1]. Nathan has been working on this research since September, though the work has focused mostly on brute force methods for understanding the relationship between global and local structural load patterns.

We began with the work done by Nathans team and extended it into local rule-based builders. Currently, the team has done work in global optimizations, studying the best ways to build structures given all the information about the system, and optimizing (in a brute force manner) for the best possible construction pattern. These brute-force studies have demonstrated (though not yet formally proven) that structures are generally more successful when agents attempt to prioritize an infill behavior over a build behavior (defined above). The team has also briefly explored interactions between basic build and reinforce behaviors in a multi-agent simulation. This early work is what we intend to build upon by using a set of local behaviors for the robots (build, infill, and traverse) and discovering the best methods for assigning local behaviors in order to better achieve the global objective of building stable structures with minimal resource usage.

In addition to the literature on collective construction, other relevant papers can be found on the topics of structural optimization and task allocation. Funes and Pollacks paper (1998) on evolutionary construction summarizes their key findings regarding the use of genetic algorithms for bridge and crane construction [2]. Both their formulation of a genetic algorithm and the methods they use to deal with torque and stress may prove useful in our work. Additionally, the manner in which higher-level phenomena like design elements emerge without being manually designed can help us explain patterns in our own project (like the potential importance of reinforcement or infill).

In addition to the genetic algorithms and construction, established models for task allocation in multi-robot systems have also been explored and motivated some of our own research. Dantu et al.s Karma hive-drone model introduced a more efficient system for task division amongst a large number of robots with a limited individual capacity for sensing and actuation [4]. The model reduces the burden for lower-level decision-making by the application developer. Since our algorithm will make decisions regarding the ratio of builder and reinforcer bots, some of Dantus ideas may be worth pursuing.

IV. Experimental Setup

The experimental setup is based on a number of assumptions that allow for simplification of the solution space. All simulations were conducted on a vertical 2D plane. Of course, a 3D environment would be a more accurate representation of any potential real-world application. However, a 2D solution space is already quite complex, so it was deemed appropriate to neglect 3D simulations until conclusions could be drawn from 2D.

Another important decision was to constrain the experiments to a regular equilateral triangular grid. While previous work on robotic strut construction has focused on 90-degree connections, this is problematic from a structural engineering point of view. In order to maintain stability, any truss structure should be triangulated. Rather than working with a 90-degree grid and forcing diagonals, we opted to instead work on a strictly triangulated grid. This allowed for trials to be run using homogeneous struts and nodes.

For the sake of simplicity and faster convergence, most trials were run on a 5x5 triangular grid space, consisting of 25 nodes, of which the bottom left node was initiated at the ”start point” and the top right node was the ”goal point”. Further extensions, especially for multi-agent simulations, used an 8x8 grid, where the target was a cluster of 4 nodes at the top right.

The results shown demonstrate the implementation of two different kinds of spring models, which were used as structural solvers. The first was a basic implementation, which only took into account axial force and neglected bending moments. Once the research reached a point where this seemed to be limiting the results, a second version was implemented, which included angular moment forces around the nodes. This is described in more detail in Approach I.

Finally, a critical component of this research was the Reinforce.js library, developed by A. Karpathy at Stanford University [4], which allowed for implementation of a number of Reinforcement Learning approaches.

A. *Establishing a Baseline: Topology Optimization*

For the purposes of evaluation, it was deemed necessary to develop a baseline to which we could compare the results

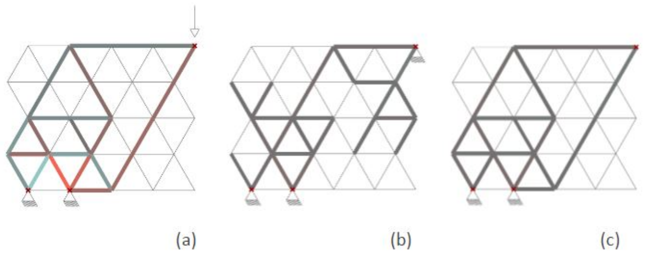


Fig. 2: In the Topology Optimization solutions above, 65 percent of the material must be removed. Two load cases are optimized together: Load Case (a) represents the truss right before connecting to the target point (the worst moment of construction), and the top right-hand point is given a point load. Load Case (b) represents the truss after it has reached the target point, so rather than inheriting a point load, the top right-hand point is modeled as a support point, similar to the bottom left-hand point. In this scenario, since there are no added point loads, the self-weight of all struts in the truss represent the only loads. The TO produces a topology that ”fans out”, attempting to reach as much of the original material as possible. Image (c) represents the results of both load cases optimized together. Since the point load applied in (a) is much greater than the loads imposed by self-weight in (b), the final form (c) inherits the topology of (a). The top right-hand node in (c) can be considered to be an average between loaded (a) and supported (b) conditions. This explains why a relative color gradient (which shows the struts under compression as red and struts in tension as cyan) is much less saturated in (c).

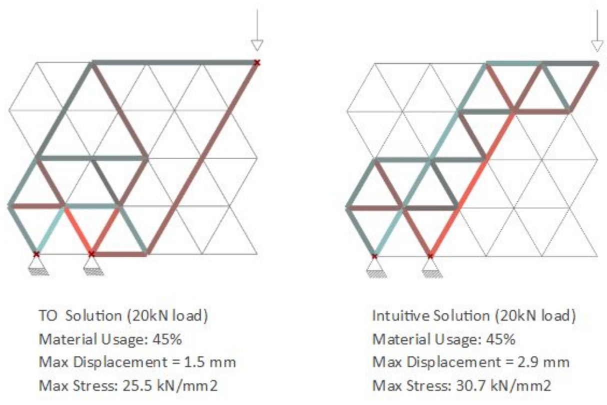


Fig. 3: Topology Optimization often produces solutions that are surprising or counter-intuitive. Here we show a comparison of a TO-generated form, and an intuitive solution designed by a human using the same number of struts. While the human solution may seem to be more robust, under the same loading conditions it is clear that the TO solution shows better performance.

of our simulations. Topology Optimization was chosen as a method for establishing a baseline. TO is a mathematical approach that seeks to optimize material distribution while minimizing compliance within a given design space, using the method of moving asymptotes, topological derivatives, GA, level sets, etc. It does not account for material discretization or buildability. In other words, a solution generated by topology optimization might be optimal in terms of material distribution and stability, but it's very likely that this form could not be constructed without additional supports or formwork during the construction process. This is further described in the images below.

For the purposes of these trials, Topology Optimization was performed using the software Millipede, developed by Prof. Panagiotis Michalatos of Harvard University's Graduate School of Design [5]. As is typical of any topology optimization, the TO solver is given a load case, consisting of constrained support nodes and point loads. It then seeks to optimize material distribution to a specified percentage. Usually, TO operates on a pixel (2D) or voxel (3D) grid space, and iteratively performs size optimization until the least structurally useful pixels/voxels are removed and the target percentage of material usage is achieved. In the case of these trials, however, it was necessary to modify the TO solver to operate on an array of struts rather than pixels.

These solutions provided by TO represent minimum compliance for a given amount of material, but tend to produce surprising or counter-intuitive trusses (see Fig. 3). Generally, TO-generated forms tend to consist of large, slender elements that act as tension rods, usually subdividing the solution space into large triangles. These elements essentially act as mirrored arches, the upper in tension and the lower in compression, which is a well-documented approach for bridge construction, etc. While these kinds of trusses represent an efficient use of materials, in many cases they would be impossible to build with discrete materials. This is clearly the case in Fig. 3. The long, slender elements that comprise its geometry could not have been built in sequence without the structure collapsing. On the other hand, the intuitive solution on the right provides a more robust, buildable alternative. Although this form allows for greater displacement and greater stress, one can imagine that it could have been assembled in sequence with much less risk of structural failure.

B. Evaluation Parameters

Since topology optimization seeks to minimize material usage while maximizing structural stability, it was deemed appropriate to evaluate these same parameters in the results of our trials. The metric for material usage was the number of struts placed (i.e. the number of edges in the graph) while the metric for stability was the maximum displacement of any node in the truss.

V. Approach 1: Heuristic-based Model

The work done by Nathan's team had established the problem space and had explored solutions to it on a global scale through topology optimization and holistic structure analysis. To take the project to the next step, the team had begun to devise a multi-agent method to actually build the structures the optimizer was suggesting and to begin exploring the impact of the time dimension in creating optimal structures.

For this purpose, the team had established some very basic, intuitive heuristics known as "behaviors" such as building and reinforcing (see Fig 1.) The build behavior focused on laying struts towards the goal and the reinforce behavior instead induced agents to build in counterintuitive directions to strengthen the overall structure. The basic idea was that by encoding these behaviors and by finding a good balance between them, a local rule based on inputs known to an individual agent could be devised in such a way as to ensure the construction of a global structure by a swarm of agents that would not only satisfy the parameters and requirements of an objectively good and sound structure, but would also be built in a practical manner that would prevent the structures from failing during the build phase.

With this in mind, we set out to expand on these local rules and to tweak some of the parameters to see what kind of overarching behaviors, structures, and build patterns we could observe from simple local rules. The motivation of this was two-fold. First, it would provide a suitable baseline as an objectively sound strategy to solve the problem posed in this project and all of our structures could be compared to the ones built by these heuristics. Second, the final global outcome was to be used as a means to provide some intuition behind effective behaviors. Both of these will come in useful for our implementation and discussion of reinforcement learning based methods, since those are far less predictable and difficult to classify or evaluate without a valuable baseline.

Ultimately, three heuristics were developed for this purpose. Outlines of these approaches as well as discussions of their effectiveness are provided below:

A. *Random Agent*

As the first, most fundamental baseline, the first heuristic was a random one. Under this local rule, an agent takes nothing into consideration and simply selects uniformly at random one of the 6 directions available to it and either builds along that direction or traverses along an existing strut, depending on whether the strut exists. The agents are bound by arbitrary limits within the state space between the source A and target B. The agent was chosen because of the extreme simplicity of its implementation and because of a desire to get a very low baseline that any reasonable agent would necessarily have to be able to beat. Such a baseline ensured that any agent that performed as poorly or worse than the random agent could be immediately discarded.

Fig 4 shows the results of running this heuristic.

From the above results, it is clear that there are two key characteristics of the random agent: (1) It builds structures that are structurally stable, and (2) it is clearly inefficient and takes a long time to reach the goal. Essentially, the agent fills in most of the space with struts, thereby achieving great structural soundness both at the conclusion and in the duration of its run. However, it wastes time and resources so despite being a great structure in terms of many of the quantitative parameters used for the evaluation of these structures (see evaluation section), this agent did not provide the results the problem was looking for. It is important to mention that this revelation - that building out the entire space leads to high scoring results - ultimately led to necessary changes to the evaluation function allowing it to severely penalize these kinds of undesirable structures.

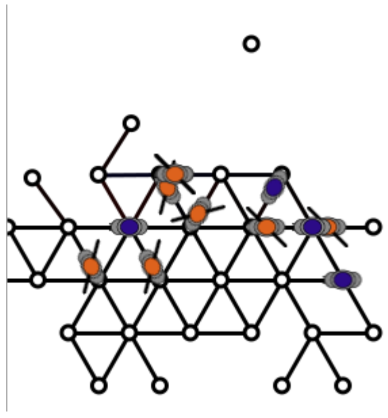


Fig. 4: Random Agent

B. *Aggressive Agent*

The second baseline was designed to build the structure the fastest, without concern for structural integrity or balance. The idea was to minimize time and struts used to reach the final goal such that a group of agents powered by this heuristic could build very unstable structures very quickly. The results of this heuristic were therefore meant to be the exact opposite of those of the random agent. Whereas the random approach build the most stable and wasteful structures in the shortest time, this agent was intended to build the least stable and most efficient structures in the shortest amount of time. As such, this was intended to be an upper bound baseline for material and time utilization and the goal of all subsequent agents was to attempt to get as close as possible to the time and resource management of the aggressive agent.

The heuristic is simple. Following a "greedy" approach, the aggressive agent has a preference for traversing struts that take it closer to the target and so construction is skewed in that direction. By evaluating the direct angle between the agent and the goal and by choosing the strut that is closest to this angle, the agent makes a "beeline" towards the goal. More formally, probabilities proportional to the strut angle's proximity to the desired angle of traversal are placed on the struts, and are then used to pick a strut to traverse. Fig 5 shows the results of running this heuristic.

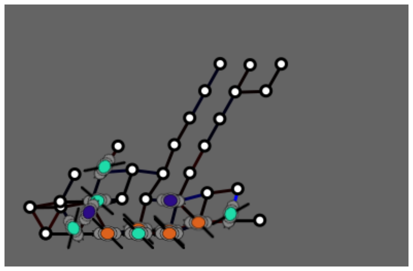


Fig. 5: Aggressive Agent

From the above results, two observations were made about this agent: (1) It reaches the goal much more quickly, using fewer struts, and (2) it results in very unstable structures with several overhanging protrusions. Thus the agent performed as intended - an impractical baseline showing the fastest possible construction approach.

C. Balanced Agent

The final heuristic was a more conservative alternative to the aggressive agent. This heuristic was intended to lie between the two extremes of the random and aggressive agents and was the only heuristic intended to be at all practical. This was the ”realistic” baseline, the one that could be achieved through intuition and would build solid reinforced global structures based on a simple local rule. The agent actually takes into consideration the two main behaviors (build and reinforce) and takes into account local information about the node, its displacement, the number of struts available, the stresses on these struts, and other criteria (more details on these in the evaluation section).

The heuristic was fairly simple. All agents would begin as builder agents with a 0.05 probability of being assigned a reinforcer role from the outset. Based on experiments and holistic observations, we set a value to be the maximal stress desired on a single strut to be 200 and the maximal average stress over struts connected to a single node to be at most 100. At its current node n, the agent a would measure s_1 as the max stress of all its connected struts and s_2 as the average stress over all struts. Then it would compute the proportion $s_1/200$ to see how close to the max desired the maximally stressed strut was and the proportion $s_2/100$ so compute the same for the average stress. The mean of these two proportions was taken (so $p = (s_1/200 + s_2/100)/2$) and p was then used as the probability of turning the builder agent into a reinforcing agent. Then, for every reinforcing agent, the probability of switching back to a builder was set to be 0.05. With every strut laid for reinforcement, this was increased by 0.05 such that after 20 struts of reinforcement placed, the agent would necessarily switch back to a builder.

These simple rules were designed to allow for an effective balance of builders and reinforcers. The idea was that building was preferable until the stresses on the struts of the structures that an agent could see became too high to be desirable and a reinforcement strategy would be adopted to strengthen the structure. Leaving all agents as reinforcers would cause behaviors similar to the random agent, however, so it was important to probabilistically return the agents back to aggressive building mode after enough reinforcements were placed. In addition to these rules, the agent followed a simple probabilistic rule to decide whether to build or traverse based on the number of struts accessible. In other words, if 4 of the 6 possible struts were already built, then the agent would move with probability 4/6 and lay another strut with probability 2/6. The laying of the struts was determined by the specific behavior (either build or reinforce). This traversal rule allowed the agent to explore its space to help it determine whether it would be better off building or reinforcing before actually placing materials. Fig 6 shows the results of running this heuristic.

As expected, the balanced agent was slow in reaching its goal and not nearly as efficient as the aggressive agent.

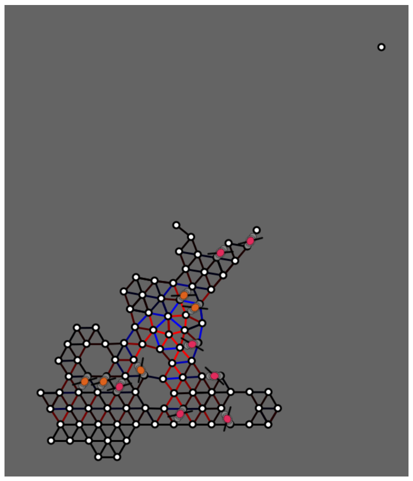


Fig. 6: Balanced Agent

Unlike the aggressive agent, however, the structures it built were more triangulated and reinforced, which was the desired result. It was considered for a time whether or not following through with optimizing these heuristics would be a worthwhile approach for this project. However, it was determined that while intuitive, these heuristics were ultimately guesses prone to human error and would not be as compelling as self-taught agents. Thus, this approach was left as a good balanced baseline and as a supplement to learning agents (more on this in the multi-agent RL section below) and was not explanded upon or optimized further.

Below are some further quantitative analyses on the structures that our different agents built and their comparisons.

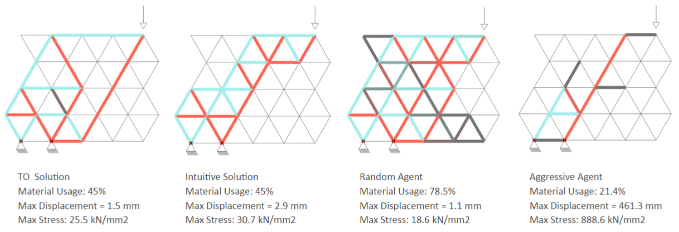


Fig. 7: Quantitative analysis of 4 baselines under similar load and support conditions

VI. Approach 2: Single Agent Reinforcement Learning

Having explored a number of basic heuristics to develop baselines for comparison and to gain intuition about local rules, we decided to implement a reinforcement learning approach. Reinforcement learning has the potential to reveal state policies that an application programmer may not have otherwise envisioned based on intuition alone. The learning process in RL includes three key components: the state space, action space, and reward function. Each state has a set of possible actions (in our case the set of nodes reachable at any stage), and each state has a reward associated with it (for example endowing a reward of “+1” when the goal is reached). The algorithm then iteratively updates the expected utility of being in a state and taking a particular action, commonly referred to as the "action-value function" or the "Q-value".

Therefore, for our purposes, there were three major parameters to consider in our implementation of Reinforcement Learning: the state representation, the interaction of agents using RL and heuristics, and the specific learning algorithm used. For the first of these parameters, two options were explored: states defined as nodes and more complex states encoding topological information. The second area for exploration was the number of agents used: How would single agent experiments differ from multi-agent experiments? What happens if some agents are running on heuristics while others are learning via RL? The final area for experimentation was the learning algorithm itself. Two approaches were taken: TD (temporal difference) learning and deep learning (discussed in more detail further on).

As described in the Experimental Setup section, the simulations sought to minimize material usage while maximizing structural stability, which is analogous to topology optimization. For the following Reinforcement Learning trials, these metrics were represented as penalties; in other words, the RL agent inherits a penalty (negative reward) for the number of struts in the truss, as well as the maximum displacement of the truss. The agent receives a reward for arriving at the goal, signifying the end of an "episode". It is important to note that the penalties are assigned at every step of the simulation (not only upon completion of the episode). This reflects the motivating constraint that the truss should be stable at every stage of construction.

A. Single-Agent Learning

The first set of trials were conducted using Temporal Difference (TD) learning on a single agent, building a truss from a start point towards a goal point on a 5x5 grid. Trials were run using both structural solvers (pinned connections and moment-resistant connections). In these first trials, the state was represented as a single variable (the index of the agent's current node) and there were up to 6 available actions for each state, representing each node's neighbors (edge nodes have fewer than 6 neighbors). An agent can take an action (move to a neighboring node) regardless of whether a strut already exists between those two nodes. If no strut exists, the agent first places it, then moves down it, following the determined action. Agents were penalized at every step for the global displacement of the structure. In some trials, agents were also penalized for the number of struts used to build the structure. Some trials penalized agents for the global structure's maximum displacement, while some assigned penalties instead for global average displacement across all nodes. In terms of the rate of learning and the resultant trusses, there did not seem to be any significant difference between these two metrics.

Results seemed to be more successful when the agent is not penalized for the number of struts in the truss. When the agent is penalized for this, it seems to exploit this by learning the shortest path to the target, despite the fact that this behavior creates unstable structures that are heavily penalized for their displacement.

While expected, one of the most exciting outcomes of these trials was that the agents learned that by building triangles,

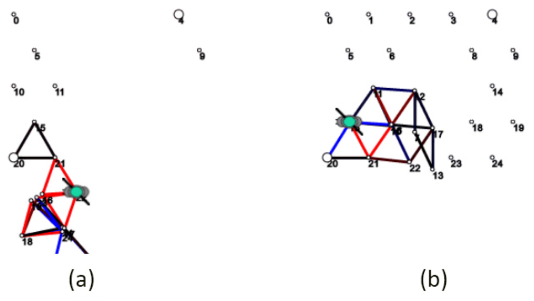


Fig. 8: Typical single-agent learning simulation where the State is only the node index. Image (a) shows the simulation after 100 iterations, while (b) shows the same simulation after 200 iterations. Notice that the agent learns to build triangles over time. However, with this setup, the agents failed to learn to build triangles continuously to the goal. We believed that this was due to the fact that the structural solver considered these to be pinned connections. Subsequent trials showed that simulations run using moment-resistant connections were able to learn to build triangles much more quickly.

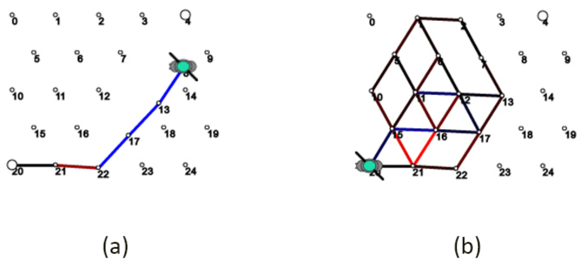


Fig. 9: These images show the 500th iteration midway through the sequence. The image in (a) shows a simulation where the agent is penalized for the number of struts used, as well as displacement. Even though it has constructed an unstable structure, it is rewarded for minimal material usage. After many trials of iteratively reducing the penalty associated with the number of struts, it was found to be more effective to remove that penalty entirely, and instead apply a penalty only for displacement. This results in trusses such as (b), which consistently builds stable structures. It inherently prefers to not add excess material, as this further increases displacement.

they were able to minimize displacement. By increasing the weighting of penalties assigned for displacement compared to strut usage, the agents learned to build triangles more quickly (see Fig. 6).

Another improvement on the rate of learning was the switch to a structural solver that took into account bending moments; in other words, considering nodes to be moment-resistant connections instead of pinned connections, as described in the Experimental Setup. In the previous pinned connections, struts would begin falling due to gravity so that they were soon dangling from a single node. This translates to a significant displacement, and meant that essentially the agent inherited a heavy penalty anytime it placed a strut. By contrast, when the nodes were modeled as moment-resistant connections, an angular spring allowed the newly placed struts to sag only slightly under their self-weight (see Fig. 7). Using this structural solver, agents were able to learn to immediately support any cantilevering struts, thus further reducing displacement (a fully triangulated truss will always have less displacement that a similar truss with a cantilevering strut).

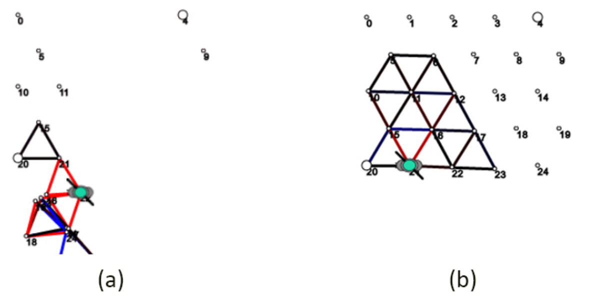


Fig. 10: Typical conditions after 100 iterations. The simulation parameters are identical except for that (a) uses the pinned connection structural solver while (b) uses the moment-resistant solver. The latter has clearly learned to build triangles (and thus stable structures) much faster than the former.

An interesting feature of the moment-resistant structural solver was that agents learned to exploit the fact that they were not penalized for the time it took them to complete the truss. In the previous solver, agents learned to build quickly, as displacements (and consequently penalties) continued to accrue over the course of the simulation. In the moment-resistant version, however, the truss quickly reached equilibrium, which meant that agents could take their time traversing around the strut without accruing additional penalties. However, as soon as an agent placed a single strut, it learned to immediately place a second to complete a triangle, thus removing the displacement due to the cantilevering strut. Still, over iterations, the agents became increasingly more reluctant to place any struts, so as to avoid penalty. Further work in this direction should include a small penalty for the number of time steps taken.

It is important to note that in the trials presented so far, the agent has no knowledge of local structural forces. While it receives a penalty based on global displacement, the State representation does not take into account any local forces or topological connections. While it was encouraging to see results even without including this information, one of the motivations of this research was to understand the effect that such local information would have on the rate of learning and the resultant trusses.

B. Topological information

Our initial single-agent approach to reinforcement learning was very encouraging, however, it was very limited in terms of the information available to the reinforcement learner. States were simply nodes and so no information regarding strut connections or stresses was codified into the system. To improve upon our learning algorithm, we adapted our single-agent approach to include topological information in our state representations.

The new approach had the following state representation: $b_1 b_2 ... b_6$, where b_i represents a binary variable $\in \{0, 1\}$ set to 1 when the ith strut exists. This binary representation was then transformed to a decimal $\{b_1 b_2 ... b_6\}_{10} = d \in [0, 63]$ and the node was appended to the end. For example, a state where the node (say, 17) had struts at locations 0, 2, 3 and 5 would have a state representation given by: "101101_{10}" + "17" = "45" + "17" = 4517. This way we encoded both location and topological information into the state.

This state representation was implemented and tested. The maximum displacement of the structure and the number of struts used by the end of each iteration are shown below:

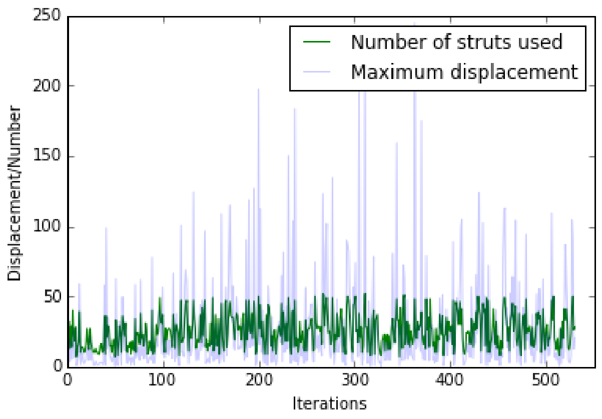


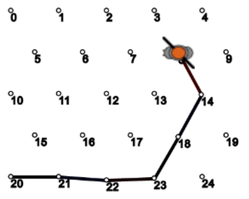
Fig. 11: Maximum displacement and number of struts with iterations

As we can see, the maximum displacement of the final structure varies wildly in the range 0 to 230. There is no sign of convergence to a lower displacement level or even a noticeable decreases over time. The number of struts varies within a narrower range but also shows no overall relationship to the number of iterations completed. The most probable reason for this behavior is that the state space is much larger given the extra information about topology ($24 \cdot 2^6 = 1600$ possible states). This means a much larger number of iterations is needed in order for the algorithm to suitably explore the space and have Q-values that converge to something more meaningful.

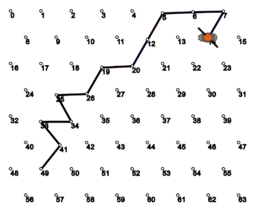
VII. Approach 3: Multi-Agent Reinforcement Learning

A. Setup and Baseline

Briefly, we outline the setup used in this portion of the research and draw on previous results to provide a baseline based on individual RL agents and the behaviors they converged to. For the setup, two different spaces were used - the standard toy model of a 5x5 grid on which most of the individual agents were tested, and the slightly larger 8x8 grid to test how the structures and the time to convergence scaled with such an increase in magnitude. It is important to note that if this baseline did not converge to anything (as happened for the vector-space learner described above), such a learner could not be expected to converge in a multi-agent setting. Thus, for these trials the more basic RL agent that did not utilize stress information had to be used. For the baseline, we used the single reinforcement learner in both spaces and noted the rapidity of convergence to a behavior as well as the subsequent structure most frequently constructed under this converged behavior. Below are the results of these trials:



(a) On a 5x5 Grid



(b) On an 8x8 Grid

Fig. 12: Single RL Agent

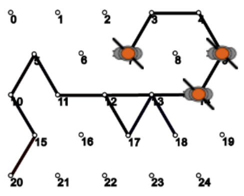
As before, in both instances the baseline learner built structures that were very linear and not very practical. They almost completely minimized material use, but did not build any sort of triangular structures for reinforcement purposes and ultimately created more of a cable between the two points rather than a solid, steady bridge. This may be useful if the target point is fixed (as it was in the simulation), but any force applied to the top of either of these structures would clearly destroy them. Quantitatively, as a baseline, the agent on the 5x5 grid converges to the above policy within approximately 150 iterations. The 8x8 grid agent requires approximately 240 iterations to learn the policy. Note that practically speaking, each iteration of the 8x8 grid inherently takes longer time since there are more than double the amount of slots for potential struts.

B. Independent Learners

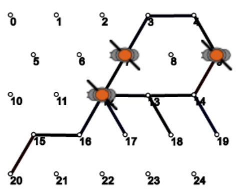
This was the naive multi-agent reinforcement learning implementation. Here, we introduced three total learners, each of which maintained its own Q function and generated its own policy to follow. As before, each was the more simple reinforcement learning agent in the sense that it did not utilize stress values of the struts around it in its state representation, but only considered its position in the grid. It was not anticipated that this approach would converge to anything, since the learning of the agents would likely be confounded on each iteration by the building actions of the other agents altering the state space each time. However, surprisingly, the agents learned to expect each other's actions and ultimately converged to a single strategy for building the structure. See Fig 13 for these results on the 5x5 grid.

On the 8x8 grid, the results were similar (note that the first two images represent unconverged structures that are nevertheless compelling). See Fig 14 below.

These were very surprising results. The differences between these structures and those of the single agent ones above are immediately apparent. While individual agents built single line "structures" that were no more than planks consisting of struts, these multi-agent swarms were actually creating very

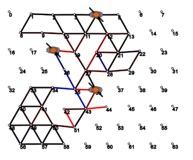


(a) Early stage

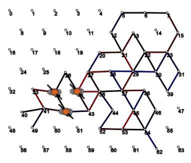


(b) Converged

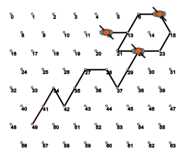
Fig. 13: Multiple Independent Agents on a 5x5 grid



(a) Early stage



(b) Middle Stage



(c) Converged

Fig. 14: Multiple Independent Agents on an 8x8 grid

interesting and structurally sound constructions. One important thing to note is that as a result of the necessity to learn quickly, our simulations reset upon reaching the desired goal, so each of these structures is 1 or 2 steps away from completion. However, the general pattern is very clear - the agents generally loop around the target node (8 in the 5x5 case and 14 in the 8x8 case) before building a final strut and resetting the learning stage again. The critical part of this simulation is that the target node is assumed fixed, and these hexagonal structures are effectively designed to serve as a powerfully reinforced fixture around the goal connected by a less reinforced path to the origin node.

The interesting comparison arises when we look at the differences in Fig 13.a against something like Fig 7.a. In 7.a, we see that the Topology Optimization advocates for a hexagonal wheel to be built around the origin node and for two long struts around the top and bottom of the target node to be extended from this origin. This makes sense as such a structure allows for a firm base but leaves the top part of the structure weakened by lack of reinforcement. It is interesting that the reinforcement learners, when working as a group, discovered an entirely opposite result. Instead of building the wheel at the base node, they constructed the simple weak structures at the base and built the reinforced wheel around the target instead. We examine the significance of this in the discussion section below, but it's important to note that this behavior will only arise if the agents are aware that the target node is immovable and learn to utilize the stability of the node to build these structures around it.

Quantitatively, around 500 iterations were necessary to learn this approach on the 5x5 grid. Interestingly, this is a little more than 3x the number of iterations necessary for the single agent and considering that there are 3 agents in total, it seems that the scaling is linear. And indeed, on the 8x8 grid about 700 iterations were necessary to converge, which follows this same pattern. This convergence speed will become more interesting as well explore ways to improve on it.

C. Policy Sharing

To avoid the competition of the agents to understand the space and to help accelerate their learning as well as to guarantee arrival at a more reasonable policy, a policy sharing approach was adopted from the Ming paper [7]. This was a simple step of averaging over all agent policies after a certain number of steps. In this case, the number of steps only affected the convergence rate and not the actual structures built at convergence. In fact, as is clear from the below, the policy sharing approach did not actually produce results that were much different to the independent learners. In fact, surprisingly the only improvement was in the rate of convergence and even that was strangely inconsistent and unexpected (see data below). Figure 15 shows the results of running a policy sharing approach on a 5x5 grid.

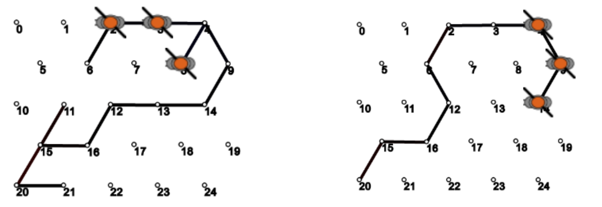


Fig. 15: Multiple Agents with Shared Policies on a 5x5 grid

The results were much more varied and interesting on the 8x8 grid. Figure 16 shows these for the regular policy sharing every 10 iterations:

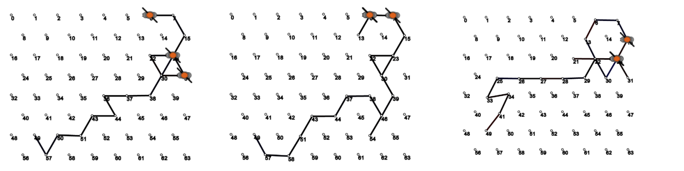
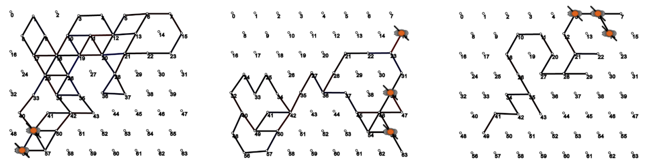


Fig. 16: Multiple Agents with Shared Policies on an 8x8 grid

Figure 17 shows the results for the policy sharing on an 8x8 grid for every 100 iterations. Note that (as in the data below), surprisingly this approach converged the fastest among all others.



(a) Early stage (b) Middle stage (c) Converged

Fig. 17: Multiple Agents with Shared Policies (every 100 iterations) on an 8x8 grid

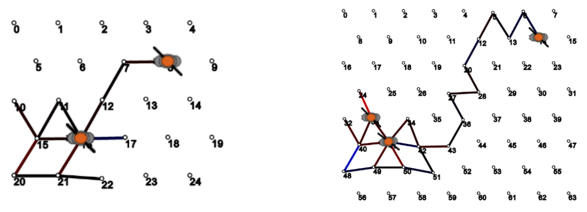
The structures of the shared policy agents were not very different from those of the independent agents. In the 5x5 space, it seemed as if the main "wheel" structure at the target was a bit wider, but the 8x8 grid had the standard hexagonal wheels so we suspect this was just a slight deviation that was not particularly significant. The interesting part about this approach was the convergence speed that was recorded for the different rates of policy sharing. For the 5x5 grid, the rates were 280, 330, and 500 iterations for sharing every 10, 50, and 100 iterations respectively. Clearly for this smaller grid, faster policy sharing yielded almost a 2x improvement over the baseline approach.

For the 8x8 grid, the results were quite the opposite. In fact, 1000, 700, and 500 iterations were necessary respectively for sharing every 10, 50, and 100 iterations. It seems that for small sizes, frequent policy sharing was effective, but for large grid sizes with more options frequent policy sharing averaged over what the agents had learned far too quickly and muddled the results. Effectively, too much policy sharing destroys the benefits of exploration for the agents, so for larger grid spaces more infrequent policy sharing is desirable.

Clearly policy sharing is very useful for improving runtimes for the learners. So long as the parameters are set correctly, policy sharing will continue to yield the same results as the independent learners but at much faster rates, so this is the superior approach.

D. Single Learner in Swarm

Since learners will inherently create structures in the process of learning that are unpredictable, it also made sense to consider a swarm learning approach that utilized agents that were more predictable and known to create solid structures. For this portion, heuristic builders based on the balanced approach were used and deployed in the simulations as a swarm along with a single learner to see if the learner could adapt to the environment and to learn something despite having no fellow agents to share policies with. From the data below, it is clear that this worked but did not improve on either the costs nor the actual structures built by the agents. Fig 18 shows the result of this approach on the two grids.



(a) On a 5x5 grid (b) On an 8x8 grid

Fig. 18: Single RL agent with 2 Balanced Agents

We compared these results to the baseline above as well as to the results in the heuristcs section. It is clear that what occurs in a situation like this is that the agents approach and reach their primary goal independently. The heuristic bots are inherently going to reinforce the structures from the outset, based on their nature. The reinforcement learning bot, on the other hand, will make a straight line for the goal as before.

Thus, the learner does not alter its converged behavior based on its swarm, but a swarm does limit the learner's ability to move and expand. Ultimately, the learner still builds the basic simple structures that are now augmented by the balanced builder agents' efforts, as can be seen above. While this is not necessarily a viable result in itself, it illuminates an interesting approach, to be considered in detail in the discussion, of optimizing a reinforcement learning agent or group of agents (as has been done here) and then to swap their behaviors upon completing the build, to be more of a reinforcement agent to preserve what has already been built. In a sense, then, this would yield a regular heuristic agent, but instead of a directional behavior, it would have the built-in behavior derived from a convergence to an optimum through the RL.

In terms of iterations, the rate of convergence, just as the learned structure, was unaffected by the introduction of the heuristic agents. On the 5x5 grid, this agent took the same 150 iterations to converge and on the 8x8 grid, 300 iterations were required.

E. Observations

Before getting into the overall discussion about the successes and failures of these approaches in the grand scope of the project aims, it is necessary to give a brief overview of some general observations about these multi-agent learners in particular. First, it is interesting that multi-agent learners will only converge if the individual agent RL approach has the capacity to converge. Second, it is a very surprising and clean result that the basic independent multi-agent approach scales linearly in convergence time with respect to the number of agents. Third, the introduction of policy sharing to the multi-agent learner is clearly effective. In fact, with the proper parameters, it allows for the reduction of the convergence time almost in half, which is extremely compelling. Fourth, the multi-agent approach actually produces unique and objectively better structures than the single agent can learn by itself. These results are highly compelling as they illustrate the effectiveness of the multi-agent approach once a good single-agent RL baseline is established, even if the single-agent approach is lacking in structural soundness and yields optimal but ineffective structures. Finally, the hybrid multi-agent approach reveals the very interesting intuition that combining behaviors together sequentially, even if these behaviors do not hold well on their own, leads to more effective structures. In other words, despite the fact that a single RL agent builds simple but very poor "structures" while the heuristic approach builds inefficient, but structurally sound ones, in combination the two would perform very well. In fact, as will be discussed in the future work section, it is very likely that a multi-agent approach built using the above policy sharing approach could be significantly augmented by transitioning to a heuristic reinforcement approach after the baseline structure (as in the diagrams above) is built.

VIII. Discussion

The aim of this project was to take the general idea of structure and bridge building between two points and to expand it into an application through swarm robotics. The motivation was to take studied scenarios and previously explored structures that would work well in those scenarios and to see if these could be reproduced by a collection of generally low-capacity agents in a swarm. By providing each agent with minimal computational power, the intent was to discover whether or not such a limitation could be overcome to still yield compelling structures when agents were put together. The project studied two main approaches - an intuitive baseline method of local heuristics, and a dynamic and systematically derived method based on reinforcement learning. Though many of the results were disappointing and impractical, several of the approaches yielded surprisingly successful results, promising the possibility of a practical application of these solutions to real problems.

First, we evaluated the baseline methods of heuristic-based agents. As expected, both the random and aggressive agents failed to yield any reasonable results. This proved to us early on that any rules that would be derived for this would have to be more complex and non-uniform to succeed. The agents could not all have the exact same behavior - either through a stochastic behavior or an allocation of deterministically different behaviors to each agent, each participant of the swarm would have to deviate from its peers so as to help achieve the overall desired result.

As a result of this conclusion, we studied the stochastic approach of the balanced agent. Though this proved more effective than the uniformly simplistic agents of the random and aggressive heuristic, it was still ineffective and did not yield the results we were looking for (the results we knew were possible based on the topology optimization). Though it was possible that a continuous tweaking of parameters or even a grid search over all possible parameters of the balanced heuristic would eventually lead to a very strong agent, we felt that this was rather arbitrary and perhaps not scalable to larger problem spaces and so it was abandoned in favor of the reinforcement learning approach.

The RL approach was surprisingly successful in unexpected ways. The individual learner we began with did not initially look promising, as it built basic and unrealistic linear structures, with no reinforcement and no concern for overall rigidity. The updated learner, concerning itself with the entire state space of stresses on the struts, failed as a result of the state space being far too large for it to ever converge. The DQN approach (more on that in the future work) also failed because of issues with representing a continuous state space in an episodic learning approach. As a consequence, early RL approaches seemed to not be sufficient to solve this problem and there was a concern that perhaps the heuristic approaches were the best hope after all.

However, modifications to the reward functions through the introduction of a more complex structural solver changed the results dramatically and allowed the agent to begin learning more structurally sound structures, as can be seen in the Approach I section above. In addition to this, the approach to combine multiple learning agents incredibly yielded much more sound structures than the individual learners. It seems that when the agents work together in the same space, they are able to strike the balance of maintaining a similar approach to all have the same general tendency towards the goal, while deviating enough from each other to create the compelling structures seen in the Approach II section above.

Ultimately, the approaches that were most successful were those that, as hypothesized, were able to develop local rules that were both geared towards building valid structures rapidly towards the goal as well as towards deviating from each other enough to create well-rounded an reinforced structures. A uniform swarm of bots will often create asymmetric, weak structures, even if the local rule is very sound. A swarm of very different robots that do not have a good balance or cannot switch roles will build very slowly and waste time and resources to get to the goal (as the random heuristic agent). Ultimately, the best performing swarms are those in which the local rule involves following a given seemingly suboptimal behavior with the knowledge that other agents will do the same and with the ultimate goal of the suboptimal approaches connecting into a single, optimal structure. Buildings and bridges require balance, counterweights, and symmetry to be built strong and to stay strong. Agents in the ideal swarm for this problem must understand this and be willing to build asymmetric structures in the hope that other agents will do the same in a way that the ultimate structure becomes sound. This is incredibly challenging to grasp intuitively and no simple heuristic will accomplish such a goal. Only through comprehensive reinforcement learning, based on multiple agents learning together and sharing information can a system be designed that, combined with some basic heuristics for reinforcement purposes, could build the best structures in the shortest time. Such agents are difficult to perfect, but this project has outlined many very effective means to create them and has taken the first steps to showing that such agents are even possible to create.

IX. Future Work

This was a fascinating project that was intended to serve as a stepping stone from an interesting baseline idea to a far more compelling practical application. With the work of Nathan's team having established the problem space as well as some heuristics and analyses on global solutions, this project took it to the next step and attempted to discover some methods of training individual agents to work in a swarm in order to accomplish the desired results. From this point, future work would involve improving upon the models presented in this paper as well as implementing them and testing their efficacy in realistic scenarios.

For improving the model, there are several approaches we began to look into, but avoided exploring in full depth as they were beyond the scope of the project. The first of these is a hybrid approach between the mutli-agent reinforcement learner and the heuristic balanced agent. The approach here would train the reinforcement learner to build the type of structure that was most prominent in the results (the weak base connected to a strong support around the target) and would reinforce this support as well as providing basic support at the base. Essentially, the total permissible struts would be limited and the heuristic approach would simply expend any remaining struts to properly reinforce the structure after it was completed by the reinforcement learner swarm. The RL agents would, upon completing their skeleton structure, transition over into the heuristic behavior, and would continue to build until all struts were used up.

For a second improvement, future work could delve further into the deep Q-learning (DQN) Approach. This was attempted during the course of these trials, but was abandoned due to issues with implementation. The decision to pursue DQN learning was based on the fact that it was capable of learning in a continuous state space. For example, many applications of DQN operate in a continuous (x,y) grid space where the agents attempt to approach a moving target while avoiding obstacles. These applications are continuous in space, but also in time, as opposed to TD learning, which is episodic. However, in our case, we would need some kind of state space representation that would have both continuous elements like (x,y) coordinates as well as an array of stresses present at a node, which would be best represented as discrete values, otherwise the simulation would never converge. Furthermore, there were implementation difficulties when trying to trick the DQN agent into working in episodic time (where the truss is cleared after every episode, but a policy array persists) as opposed to its default of continuous time.

Finally, for the practical application, a very compelling bit of future work would be actually constructing robots to build the simple structures on the toy model 5x5 grid. Certainly the simulations would first need to be honed to deal with realistic factors such as robot weight and the necessity of robots to collect trusses after each placement, but after these tweaks are made and the models optimized, applying this to actual robotics would be a fantastic approach to future work.

X. Related Work

A lot of interesting research has been done in some of the areas explored in this paper. For example, our work involved conceptual hurdles similar to those faced by Yamins and Nagpal in their paper on Global-to-Local programming [3]. In their paper, Yamins and Nagpal present a theoretical framework for the existence, construction, and resource tradeoffs involved in creating "global-to-local" compilers. This is similar to our problem; our global goal was construction towards the goal point and our local rules were the policies our agents learned for each possible state. Furthermore, we saw first-hand how the tradeoffs described in nagpal's paper play out on a global level.

Another interesting area of research that relates to our work is that of multi-agent reinforcement learning. Ming Tan's seminal paper on the topic laid the foundations for thinking about policy sharing between agents [7]. His findings included the fact that learning happened more quickly when policies were shared and that agents working on joint tasks in partnership can "significantly outperform" at the cost of slower learning at the start. We definitely found evidence of the Tan's first finding when we attempted multi-agent reinforcement learning with shared policies.

Finally, some interesting results have also been found in the area of deep Q learning. Volodymyr Mnih's 2013 paper, "Playing Atari with Deep Reinforcement Learning" showed that a deep learning model can be used to effectively learn policies from multi-dimensional sensory inputs using reinforcement learning [8]. The model they used was a convolutional neural network, trained with a variant of Q-learning. This is conceptually similar to our work with deep Q-learning.

Appendix A
Running the Code

1. Clone the repository:

https://github.com/sbalanovich/builder-bots

2. Make sure that index.html is calling the correct JS file you would like to run:

a) sketch_random.js - Random Heuristic

b) sketch_base2.js - Aggressive Heuristic

c) sketch_balanced.js - Balanced Heuristic

d) sketch_RL.js - Basic Reinforcement Learner

e) sketch_RL_vector.js - Vector-Based Reinforcement Learner

f) sketch_RL2.js - Best Reinforcement Learner

g) sketch_multi_a,b,c - Multi-agent RL, where a,b,c refers to the three approaches above, in order

3. Run index.html. Click the whitespace if nothing happens at first.

Appendix B
Multimedia

Movies Dropbox Link:
https://www.dropbox.com/s/dbi9k4tymqcsk66/final%20video.mp4?dl=0

Presentation Slides:
https://docs.google.com/presentation/d/1gXJ9LVr36rNTfIddrqWRjTCYVteWplG9Bd5wLrqrTl8/edit?usp=sharing

Appendix C
Contributions

Nathan provided the code base (which included heuristic behaviors which were then further developed over the course of this project), initial implementation of TD reinforcement learning and DQN learning, and implementation of both structural solvers.

Essam worked on the aggressive agent heuristic, the reinforcement learner using vector state form (for topological information), and the reinforcement learner encoding stresses on struts.

Serguei worked on the main part of the adjacency matrix representation for evaluation, the random agent heuristic, and the multi-agent approaches and analysis.

Each member of the team contributed equally to the presentation and paper. Beyond the baseline, all members contributed equally to the vision and execution of the project (though Nathan deserves to be credited with providing the majority of the initial vision and base code for the project).

Acknowledgment

The authors would like to thank Dr. Justin Werfel as well as Profs. Radhika Nagpal and Panagiotis Michalatos for their support of this research.

References

[1] J. Werfel, Kirstin Peterson, Radhika Nagpal, *Designing Collective Behavior in a Termite-Inspired Robot Construction Team*, Harvard University, 2014. http://science.sciencemag.org/content/343/6172/754

[2] Funes and Pollack, *Evolutionary Body Building*, Brandeis University, 1998.

[3] D. Yamins and R. Nagpal, *Automated Global-to-Local Programming in 1-D Spatial Multi-Agent Systems*, Harvard University, 2015. https://www.eecs.harvard.edu/ssr/papers/aamas08-yamins.pdf

[4] K. Dantu, *Programming micro-aerial vehicle swarms with karma*, Harvard University, 2011. http://dl.acm.org/citation.cfm?id=2070956

[5] Panagiotis Michalatos, *Millipede software*, Harvard University, 2005. http://www.grasshopper3d.com/group/millipede

[6] A. Karpathy, *Reinforce.js framework for Reinforcement Learning*, Stanford University. http://cs.stanford.edu/people/karpathy/reinforcejs/

[7] Ming Tan, *Multi-Agent Reinforcement Learning*, MIT, 1993. http://web.media.mit.edu/ cynthiab/Readings/tan-MAS-reinfLearn.pdf

[8] V. Mnih *et al.*, *Playing Atari with Deep Reinforcement Learning*, University of Toronto, 2013. https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf