Assembly Order Planning for Modular Structures by Autonomous Multi-Robot Systems*

Tom Peters[1], Kenneth Cheung[2], and Irina Kostitsyna[3]

***Abstract*— Coordinated multi-agent robotic construction provides a means to build infrastructure in extreme environments and improve efficiency in high performance applications. Planning methods are key to understanding and achieving the scope of such applications, and are typically tailored to specific models of construction material and a consideration of passivity or activity thereof. Here, we focus on the NASA Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) model, which includes passive lightweight structural modules and small robots that traverse the structure. We present an algorithm for calculating a build plan for robots under the constraints of this type of system. We then evaluate the quality of this plan experimentally. Many of the techniques we use can be applied to any robotic assembly system whose robots perform locomotion over the structure that they are building.**

I. Introduction

Autonomous construction systems employing multiple robotic agents and passive construction materials are described from a planning algorithm perspective as hybrid programmable matter. Multi-agent collective construction [1], and integrated material-robot systems, and some general reconfigurable robotic systems fit this description wherein a primary structural system is composed of matter that does not perform movement on its own, but must be acted upon by distinct robotic agents. Early examples performed stacking of modules, with wheeled [2], stepping, or flying robots [3], often limited to creating a 2.5D terrain, without any overhangs or holes. Robots in these systems are smaller than the structures the system can build, and the construction matter or materials can have features that allow them to fixture to each other and to help them align during placement [4]. When implemented with mechanical fasteners, this fixturing can be strong enough to form high performance mechanical metamaterials [5], [6]. A natural robot type for walking over fixtured systems are inchworm type robots [7], [8]. These robots allow for a wide range of operations using only low numbers of degrees of freedom and control states.

Across robot types, it has been investigated what structures can be built and in what order. Reinforcement learning has been considered [9]. Mixed integer linear programming was used to generate an optimal build plan [10]. This was later sped up by using a hybrid version, by first deconstructing the target structure into independent substructures that can be planned individually and separate from each other [11]. This approach was then improved to use a hierarchical subdivision of the structure [12]. It is also of interest to determine what conditions are required for a structure to be possible. Using a model where cubic modules slide around and are not allowed to disconnect the structure during construction, it has been shown that all structures lacking certain forbidden patterns can be built [13].

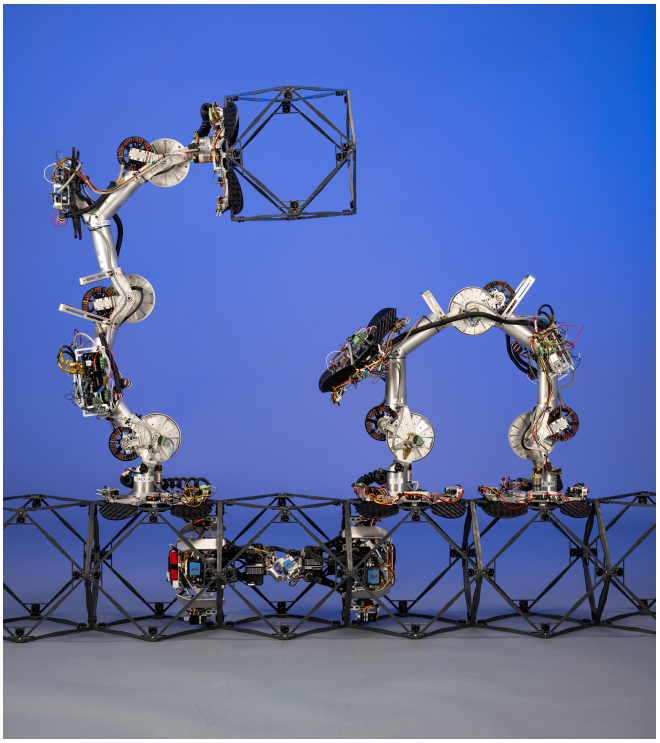

Fig. 1. A SOLL-E [14] picking up a voxel from the backpack of another SOLL-E. A MMIC-I [15] is present inside the voxels [6].

The example of coordinated multi-agent robotic construction that we address here is the Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) infrastructure. The building blocks are cuboctahedron voxels that are acted upon by two types of robots. The voxels connect to each other into a cubic grid. An initial study determined that a two-robot-type solution for building large scale constructions is the most energy efficient [16]. The first type of robot is called MMIC-I [15]: it crawls through the structure of voxels and bolts them together. The second is called SOLL-E [14]: an inchworm type robot with two feet

*This work was supported by the NASA Game Changing Development (GCD) Program, Space Technology Mission Directorate

[1]TU Eindhoven, Eindhoven, The Netherlands t.peters1@tue.nl

[2]Coded Structures Lab, NASA Ames Research Center, Moffett Field, CA, USA

[3]KBR, Inc., Coded Structures Lab, NASA Ames Research Center, Moffett Field, CA, USA

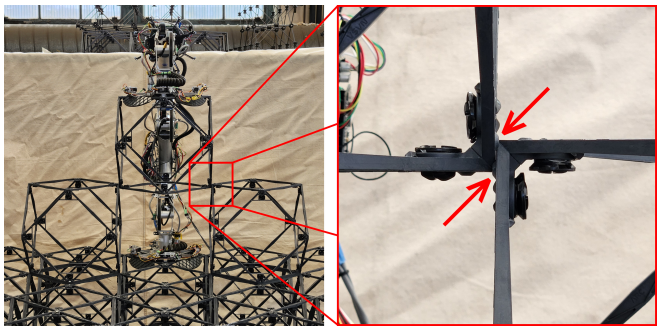

Fig. 2. [13] Photograph of armadas robot attempting to place a voxel between two other voxels and failing due to collisions of mechanical alignment features (red arrows).

and a so-called backpack with which it can carry voxels. Its feet are the exact size of the side of a voxel and can grip the side of a voxel. In this way, it can grip the structure in any orientation, both under gravity as well as in space. It can move over a construction of voxels by performing inchworm type movements, see Figure 1.

These robots can work together to create a structure C out of the voxels. We assume that there is a prebuilt plane *base* of voxels that the robots can utilize to move. The initial voxels of C can be bolted to this base. Furthermore, we assume that there exists a *depot*: a single spot on which SOLL-E's get a voxel loaded on their backpack.

Prior descriptions of ARMADAS system implementations demonstrated teams of three robots. A *cargo* SOLL-E, a *crane* SOLL-E, and a *bolter* MMIC-I. The crane and the bolter robot go to where a new voxel needs to be attached, while the cargo robot goes to the depot first to pick up a voxel. The cargo robot then goes to the construction site, where the crane picks up the voxel from the backpack and holds it in the correct place. Then, the MMIC-I bolts it in place. Lastly, the bolter and crane robots go to the place where the next voxel needs to be attached, while the cargo robot goes via the depot first to pick up a new voxel.

To enable precise alignment, voxels have mechanical alignment features, see Figure 2. These bumps and corresponding dents allow for more precision and more stability in the final structure. However, because they protrude slightly outside of the bounding box of a single voxel, new voxels need to move away slightly before being able to slide into place. This, along with normal deformities under the weight of the structure as well as the robots, might make it impossible for a voxel to get into the correct position. In particular, it is impossible for a voxel to pass through a unit-wide gap between two other voxels. We call this the *no-tight-building* constraint.

This constraint imposes a direct ordering on all connected voxels in a row as soon as one of them has been placed. This significantly limits the possible options to build different parts of the structure at the same time if the algorithm starts with a suboptimal voxel.

II. Path planning

For a detailed description of the path planning approach see the accompanying article [17]. In this paper we give a brief overview.

Due to the grid structure, there are only a finite number of positions and poses a robot can be in. We represent these poses as a graph $G = (V, E)$, where V represents all possible poses given the current structure, and $(u, v) \in E$ for $u, v \in V$ if the poses represented by u and v can be reached directly from each other. The shortest path in G is the fastest way to go from a starting pose S to a target pose T. Performing multi-agent path planning on a graph is difficult and is already NP-hard for squares in a grid [18]. Therefore, this is not solved optimally, but instead an approach is used that is good enough in practise. We use Multi-label A* (MLA*) to find good enough paths [19]. This is a version of A* for multiple robots with multiple labels. It can guide robots to their respective locations without collisions and it makes sure that the cargo robots go via the depot to pick up a voxel before going to the construction location. The heuristic for A* is based on the Manhattan distance between the feet of the robot and the target.

To make sure that robots are never completely blocked by other robots, each of them gets a dedicated endpoint [20]. Whenever a path is calculated towards a target, instead of stopping there, a path is also calculated from the target back towards the endpoint. Because MLA* makes sure there are no collisions, this guarantees that there always exists a path for every robot to go back to its endpoint.

III. Build order

Every voxel that is being placed can subsequently be walked on by the robots. However, it also limits their movement and might make the path to some areas longer, or even block off certain areas entirely. Therefore, the order in which voxels are placed can heavily impact the total construction time, or even make it impossible to finish the construction. Another important constraint imposed by the model is the no-tight-building constraint. We assume only incremental construction. Therefore, placing a single voxel immediately imposes a complete order on all other connected voxels in the same row or column. Lastly, when robots operate close to each other, they interfere with each other. Because a single robot calculates a path towards the construction site, and then reserves a path back to its endpoint, it might be difficult for another robot to reach the construction site in the meantime. Hence, if different robots need to place voxels right next to each other, it might be slower than letting a single robot place both voxels.

To minimize the total assembly time, we want to increase parallelism by letting robots operate on independent parts of the same structure. We use the following algorithm to determine the order of construction.

A. Vertical partitioning

We assume that the configuration C to be built is connected. However, it can contain both holes as well as over-

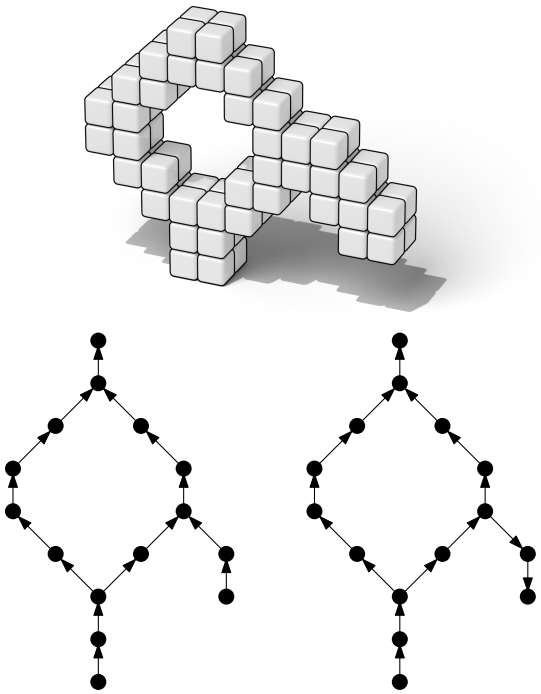

Fig. 3. A configuration C, its corresponding slice graph, and its corresponding slice graph with all nodes reachable from a ground root by having some dependencies reversed.

hangs. To ensure that we do not violate the no-tight-building constraint, we create a directed acyclic dependency graph G_C of configuration C. Every vertex v in G_C represents a subset of voxels of C. Let a *predecessor* (*successor*) u of v be a vertex such that there is a directed edge from u to v (from v to u), and let a *root* of G_C be a vertex without predecessors. We can only start building the voxels represented by some vertex v, if all voxels corresponding to its predecessors u are completed first. Hence, construction needs to start at one of the roots.

To construct G_C, we first create a *slice graph* of C [21]. To create a slice graph, we split the voxels up by z-coordinate. Then, for each z-coordinate, we split the voxels into the connected components per layer. For each of these connected components, we create a single node v that represents the voxels in that component. With slight abuse of notation, we say a voxel c is in v ($c \in v$) if c is contained in the connected component that is represented by v. We insert a directed edge (u, v) for two nodes u and v, if and only if there exist two voxels $c_1 \in u$ and $c_2 \in v$ such that c_2 is directly above c_1, see Figure 3. We use this slice graph as a base for G_C.

This slice graph is a directed acyclic graph with multiple roots, some of which containing voxels not on the ground. However, a root that is not on the ground can never be built, since a voxel always needs to be bolted to something. A voxel can never be placed without it being attached to other voxels or the base. To remedy this, we calculate the set of vertices U not reachable from a ground-root by performing a breadth first search starting from all ground-roots. Next, we take the vertex $v \in U$ with highest z-coordinate such that there exists a reachable vertex u for which v is a predecessor. We flip this dependency by making u a predecessor of v instead. Then, we start the process again by performing a breadth-first search until U is empty, see Figure 3. Once U is empty, the resulting graph is G_C.

Lemma 1: If the order of assembly within each node of G_C does not violate the no-tight-building constraint, then the order imposed by graph G_C does not violate the constraint either.

Proof: For G_C to violate the no-tight-building constraint if all individual nodes do not violate the constraint, there must be a node v representing a layer that has predecessors both above and below it. Before flipping any edges, G_C contains only edges going up. Assume for contradiction v has a predecessor w with $z_w = z_v + 1$, and a predecessor u with $z_u = z_v - 1$, and U is empty. The edge from (w, v) would only be created by flipping the original (v, w) edge. This means that at that point $v \in U$. Because U is now empty, v was the last node in U. Because only the highest non-reachable node has its edges flipped, at any point in time, if there exists an unreachable node in U, on that height or below, no edges are going down. Therefore, there is no other path from v to u, and because v was unreachable, u is also unreachable and $u \in U$. This contradicts U being empty and therefore v does not violate the constraint. ■

B. Further ordering

At this point, every node in G_C contains only voxels with the same z-coordinate, and every voxel corresponding to a node must be built before a voxel with a higher z-coordinate can be built. This is a strong restriction, since not all voxels in a layer are actually required before more layers can be built. To increase the available parallelism, we will simplify the dependency graph wherever possible without violating the no-tight-building constraint.

We repeatedly take an edge (u, v) such that v is the only successor of u and u is the only predecessor of v, and *contract* it. This means that we replace both u and v with a new node v'. Vertex v' represents the same voxels as u and v combined. The predecessors of v' are the same as the predecessors of u, and the successors of v' are the same as the successors of v. In essence, these two nodes represent a "pillar" in the configuration without any branching, see Figure 4.

The resulting dependency graph G_C ensures that robots can work on nodes that are mutually independent. However, sometimes there are more robots than nodes that can be built simultaneously. Therefore, every vertex v in the dependency graph is further partitioned.

The resulting dependency graph G_C ensures that robots can work on nodes that are mutually independent. To order the voxels inside a node v, we run a breadth-first search on all voxels in v, starting from any voxel in v that neighbors a voxel in a predecessor of v. We call the resulting graph G_v. To fix the potential violations of the no-tight-building constraint within G_v, we repeatedly pick a voxel x that violates the constraint and flip it's edges such that it has

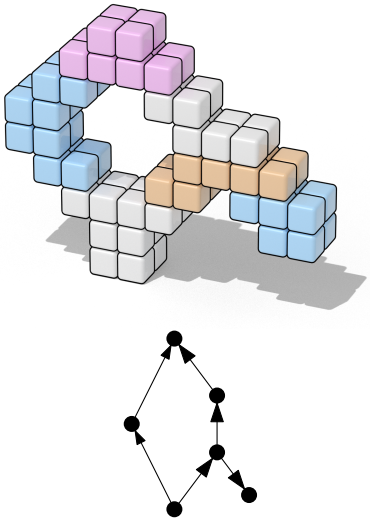

Fig. 4. The compressed dependency graph G_C of the configuration shown in Figure 3, and that same configuration with each of the vertices of the nodes of G_C highlighted in a different color.

no more incoming edges from a voxel with x-, y-, or z-coordinate higher than it. If multiple voxels x exist that violate the constraint, we pick x to be the voxel with minimum z-, then y-, then x-coordinate. If the flipping of these edges makes another voxel violate the no-tight-building constraint, then this voxel will be considered next. By construction, this makes it so that within each node of G_C, the build order does not violate the no-tight-building constraint. Together with Lemma 1, this makes it such that the whole order adheres to the constraint.

For a configuration consisting of n voxels, calculating the build order can be done in $O(n)$ time. During the first step, each edge and each voxel will only be considered once for the breath-first search. Since each voxel has a constant amount of neighbors, there are at most $O(n)$ edges in the graph and the whole breadth-first search runs in $O(n)$ time. Then, inside each component, we again run a breadth-first search, which takes $O(v)$, for a vertex of G_C consisting of v voxels. Because every voxels occurs in exactly one of these vertices, the total running time to calculate the build order is $O(n)$.

C. Robot assignment

Whenever a robot starts working on a node of G_C, they build all of the voxels in that node before moving on to another node. Nodes are built in order of the dependency graph. Whenever a robot is finished with building all voxels in a specific node, it moves on to any other node for which all predecessors have been completely built. However, sometimes there can be more teams than nodes that can be build at the same time in the dependency graph. Therefore, with each node, we also store the number of voxels in that respective node.

Whenever a robot r needs to be assigned to a new node, but there are no free nodes with all predecessors built, we can calculate the workload per node. To do so, we calculate the number of voxels that *depend* on a node v by summing for every node v its number of voxels together with the number of voxels of all nodes that have v as their ancestor. Then to get the workload, we divide this number by the amount of robots that are already assigned to v. Now r will be assigned to the node with the highest workload.

IV. Evaluation

To evaluate our technique, we compare this build order with a lexicographical sorted order. To create this sorted order, we take all voxels, and sort them in ascending order on z-coordinate, then on y, and then on x-coordinate.

We compare these two approaches on several configurations, using different numbers of teams. Because we expect that our approach increases the amount of parallelism, we test the approaches on configurations that allow for a significant amount of parallelism.

We pick three configurations, with 216 voxels each, see Figure 5. We chose configurations that have different properties and different amounts of inherent parallelism. Configuration 1 is a $6 \times 6 \times 6$ block, configuration 2 consists of four pillars of 3×3 voxels and 6 voxels high. Lastly, configuration 3 has nine pillars of 2×2 voxels and 6 voxels high. The pillars are at least six spaces apart, such that robots walking on one pillar do not interfere with those on another pillar.

In principle, more teams of robots does not automatically lead to a faster build time. Since the robots are bigger than the voxels, they need some space around each voxel to place it. Therefore, if there are two teams that want to place voxels close to each other, they might not be able to do this simultaneously, but instead have to resort to a sequential approach. Ideally, keeping the robots separated should lead to faster build times. Nevertheless, this is not always possible and largely depends on the geometry of the configuration and its size. If the configuration has narrow walkways or paths, there is likely more interference than when the configuration does not possess these properties.

This interference makes it such that each configuration has an "optimal" number of teams; while adding more teams would still help and decrease the build time, the improvements significantly diminish.

On the first configuration, we expect that our approach will separate different teams better than the lexicographical sorted order, since the breadth-first search allows the robots to build voxels in all directions, instead of close together. The lexicographical sorted order instead places every voxel next to the previous. This would mean that teams always work close together and are not spaced out. Therefore, our order should make for less interference between the teams and therefore a faster build time. On the configurations with more pillars, we expect that our approach separates the robots more, giving each team its own pillar to work on. The lexicographical sorted order instead often assigns robots to neighboring voxels in the same pillar. Again, this should

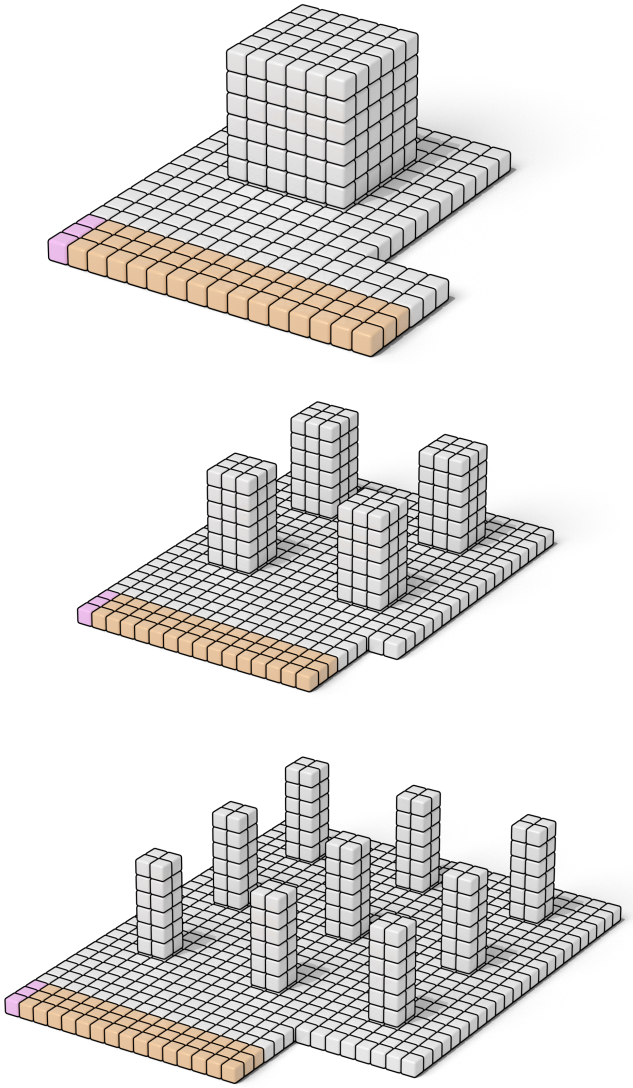

Fig. 5. The three configurations. The robots start on top of the orange voxels. When a robot stands on top of the purple voxels representing the depot, it can obtain a voxel in its backpack. The ground layer is already present before the robots start building.

lead to more parallelism in our order, and less interference between the teams.

A. Results

For each experiment, we calculate a baseline as follows. We take the number of steps it takes a single team to construct the lexicographical sorted order, and divide this number by the number of teams. This gives an estimated lower bound on the total number of steps needed to build a structure if none of the teams would interfere with each other. Note that this is not a true lower bound on the time necessary for construction. After all, sometimes robot interference is unavoidable, and even for a single team, the running time is dependent on the build order, since placed voxels might interfere with future paths. However, when comparing the different construction order approaches to this baseline, it should give an impression of how much interference happens.

For the first configuration consisting of a single block of voxels, the results are visible in Figure 6. Our order performs better than the sorted build order. The advantage increases with the number of teams, until there are six teams. At this point, the advantage stays approximately the same, with our approach taking approximately 78% of the time that the sorted order takes. Starting from around seven teams, adding more teams seems to not decrease the build time. This can be explained by the fact that the block is $6 \times 6 \times 6$ voxels. Adding more teams of robots makes them only interfere more. Therefore, the robots have to wait for each other to finish instead of working at the same time. It seems that the optimal amount of teams for this configuration is around seven. Moreover, although adding more teams still decreases the build time, the effect of adding more teams decreases in general, as can be seen from the baseline.

The difference between the actual results and the baseline represents the overhead caused by the interference between the robots. The observed build time for both the sorted order as well as our order does not decrease by much anymore, but the baseline does. This indicates that the more robots there are, the more they interfere and have to wait for each other.

A similar result can be seen for the four pillars of three by three voxels, visible in Figure 7. Here, the difference between our order and the sorted order increases until ten teams, then stays stable with our order taking approximately 76% of the time the sorted order takes. Moreover, it seems that adding more teams does not decrease the build time after 12 teams. Again, this can be explained by the fact that the pillars are only three by three voxels, and thus more than four teams per pillar would increase the interference.

Lastly, for the nine pillars, the build times for our order and the sorted order are approximately equal, with our order being slightly slower. This can be explained by the size of the pillars. Because the pillars are smaller, the amount of robots that get sent to each pillar in the lexicographical sorted order is smaller too. Most importantly, the number of robots assigned to each pillar in the lexicographical sorted order stays constant throughout the construction. This leads to less interference using this order. On the other hand, our build order divides the robots evenly over the pillars. This inevitably leads to some pillars being finished earlier than others; they are closer to the depot, or the number of robots assigned to each pillar is uneven. Eventually, once some pillars are done, our algorithm assigns available robots to the pillars that still need the most work. This leads to too many robots assigned to finishing the last pillar at the same time, and hence much interference. This is a natural example of a configuration with an inherent bottleneck. In this case, the bottleneck consists of narrow substructures with a rather low optimal number of robot teams. Such substructures must be constructed with care and we must try to not assign too many robots to it.

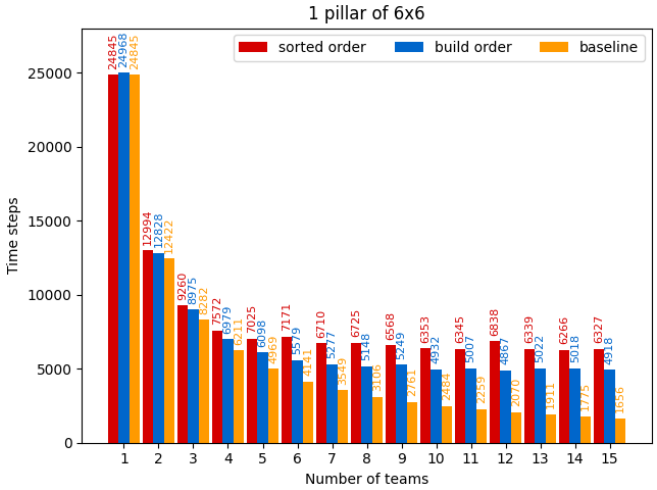

Fig. 6. The timesteps it takes for different numbers of teams to finish building a block of six by six by six voxels.

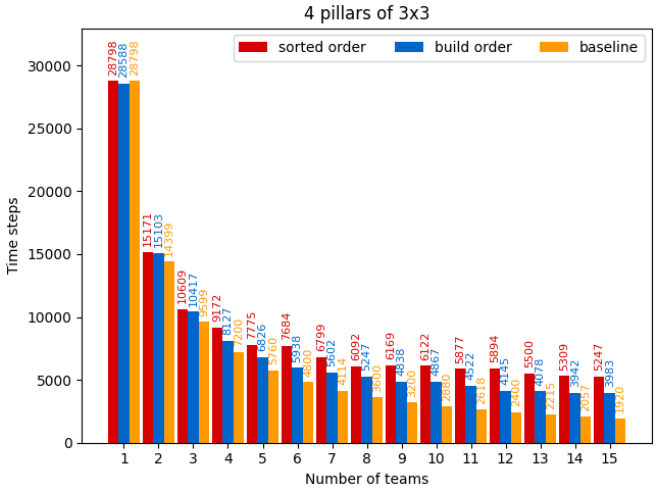

Fig. 7. The timesteps it takes for different numbers of teams to finish building four pillars of three by three voxels.

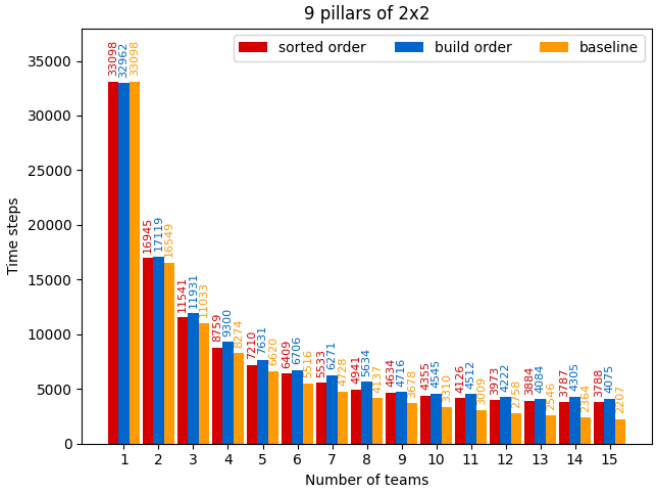

Fig. 8. The timesteps it takes for different numbers of teams to finish building nine pillars of two by two voxels.

V. Conclusion

We presented an algorithm for dividing a structure over teams of robots to be built. We tested our algorithm on three different configurations, each with different inherent amounts of parallelism. It turns out that our build order is better as long as the configuration does not contain inherent bottlenecks. For dense configurations, our build order spreads out the robots to limit interference. For configurations with different elements that can be built independently, our build order algorithm is also faster, since it divides robots evenly over the different substructures. However, when the configuration contains narrow walkways, our algorithm performs worse. This is due to the fact that having more robots work together on a narrow walkway does not necessarily lead to a faster build time, and instead might even slow it down.

A simple solution to this problem might be to try to detect these bottlenecks and only assign a limited number of robots to them. This would prevent too many robots crowding the limited space. Another approach is to change up the team configuration. The armadas architecture uses teams of three robots each. However, the cargo and crane SOLL-E have exactly the same specifications. It could be beneficial to have SOLL-E's swap between placing voxels and carrying voxels to the construction site. Especially when having multiple crane SOLL-E's close together in a confined space only slows down construction, since one SOLL-E needs to move out before the other can go in.

References

[1] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, "A review of collective robotic construction," *Science Robotics*, vol. 4, no. 28, p. eaau8479, 2019.

[2] K. Petersen, R. Nagpal, J. Werfel *et al.*, "TERMES: An autonomous robotic system for three-dimensional collective construction," in *Robotics: Science and Systems*, vol. 7, 2011, pp. 257–264.

[3] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams." in *Robotics: science and systems*, vol. 7. Los Angeles, CA, USA, 2011, pp. 177–184.

[4] J. Romanishin and D. Rus, "Dynamic and repeatable modular assembly: How kinematic couplings and proprioceptive actuators simplify robotic assembly," in *International Symposium on Experimental Robotics*. Springer, 2023, pp. 308–318.

[5] K. C. Cheung and N. Gershenfeld, "Reversibly assembled cellular composite materials," *Science*, vol. 341, no. 6151, pp. 1219–1221, 2013. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1240889

[6] C. E. Gregg, D. Catanoso, O. I. B. Formoso, I. Kostitsyna, M. E. Ochalek, T. J. Olatunde, I. W. Park, F. M. Sebastianelli, E. M. Taylor, G. T. Trinh, and K. C. Cheung, "Ultralight, strong, and self-reprogrammable mechanical metamaterials," *Science Robotics*, vol. 9, no. 86, p. eadi2746, 2024.

[7] C. Wagner, N. Dhanaraj, T. Rizzo, J. Contreras, H. Liang, G. Lewin, and C. Pinciroli, "Smac: Symbiotic multi-agent construction," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3200–3207, 2021.

[8] B. Jenett and K. Cheung, "BILL-E: Robotic platform for locomotion and manipulation of lightweight space structures," in *25th AIAA/AHS Adaptive Structures Conference*, 2017, p. 1876.

[9] G. Sartoretti, Y. Wu, W. Paivine, T. S. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *14th International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 35–49.

[10] E. Lam, P. J. Stuckey, S. Koenig, and T. S. Kumar, "Exact approaches to the multi-agent collective construction problem," in *26th International Conference on Principles and Practice of Constraint Programming*. Springer, 2020, pp. 743–758.

[11] A. K. Srinivasan, S. Singh, G. Gutow, H. Choset, and B. Vundurthy, "Multi-agent collective construction using 3d decomposition," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 9963–9969.

[12] S. Singh, G. Gutow, A. K. Srinivasan, B. Vundurthy, and H. Choset, "Hierarchical propositional logic planning for multi-agent collective construction," in *Construction Robotics Workshop*, 2023.

[13] J. Brunner, K. Cheung, E. Demaine, J. Diomidova, H. D. Christine Gregg, and I. Kostitsyna., "Reconfiguration algorithms for cubic modular robots with realistic movement constraints," in *Proc. 19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, 2024, pp. 34:1–34:18.

[14] I.-W. Park, D. Catanoso, O. Formoso, C. Gregg, M. Ochalek, T. Olatunde, F. Sebastianelli, P. Spino, E. Taylor, G. Trinh *et al.*, "SOLL-E: A module transport and placement robot for autonomous assembly of discrete lattice structures," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 10 736–10 741.

[15] O. Formoso, G. Trinh, D. Catanoso, I.-W. Park, C. Gregg, and K. Cheung, "MMIC-i: A robotic platform for assembly integration and internal locomotion through mechanical meta-material structures," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 7303–7309.

[16] B. Bernus, G. Trinh, C. Gregg, O. Formoso, and K. Cheung, "Robotic specialization in autonomous robotic structural assembly," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–10.

[17] I. Kostitsyna, K. Cheung, and J. Gloyd, "Multi-agent collective construction of general modular structures," *to appear at ICRA 2025*, 2025.

[18] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer, "Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch," *SIAM Journal on Computing*, vol. 48, no. 6, pp. 1727–1762, 2019.

[19] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label A* algorithm for multi-agent pathfinding," in *Proc. International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 181–185.

[20] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," *ArXiv e-Prints*, 2017. [Online]. Available: https://arxiv.org/abs/1705.10868

[21] C. Sung, J. Bern, J. Romanishin, and D. Rus, "Reconfiguration planning for pivoting cube modular robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1933–1940.