Multi-Agent Collective Construction of General Modular Structures

Irina Kostitsyna[1], James Gloyd[1], and Kenneth Cheung[2]

***Abstract*— We present an algorithmic framework for a multi-robot modular assembly system. Motivated by the prospects of in-space assembly, we focus on the NASA Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) framework, in which multiple types of robots work together in a team to build large structures. Unlike with other multi-robot construction systems, the geometry of structures that ARMADAS robots can build is not limited to the class of histogram shapes. To address the intractability of path planning for a robot system with the exponentially growing number of dimensions, we present a decoupled planning approach, where the assembly and path planning is performed iteratively by one robot team at a time. We present a number of data structures which help us avoid collisions and deadlocks in the resulting robot schedule.**

I. INTRODUCTION

In-space assembly is crucial for advancing human space presence and in-space capabilities, as assembly allows for the construction of large structures that would otherwise be impractical or impossible to deploy due to size and weight constraints. While human directed and performed assembly is possible with extra-vehicular activity by astronauts, a markedly more advantageous option is to use autonomous robotic agents. We need not look far to see benefits provided by robotic automation terrestrially, and many of the same benefits to productivity and efficiency can be achieved through automation in space as well. One key benefit provided by robotic automation for in-space assembly is scalability, specifically, being able to increase project size, scope, and efficiency by simply supplying additional robotic teams, which then work together in parallel. Alongside this scalability comes an ability to distribute and redistribute robotic agents and teams to multiple endeavors at a time, adapting to changing requirements, goals, and conditions.

The NASA Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) project [1] has developed a highly modular and versatile multi-robot assembly system. Structures produced by the ARMADAS system are comprised of a class of ultra-light weight and strong materials, which consist of unit-size building blocks called *voxels*. These strut-based building blocks are assembled together into discrete grid patterns by a crew of two types of robots (see Fig. 1): the Scaling Omni-directional Lattice Locomoting Explorer (SOLL-E) [2] and the Mobile Metamaterial Internal Co-Integrator (MMIC-I) [3]. SOLL-E is a bipedal robot that can walk along the surface of the structure and can carry voxels in its backpack. It grips to the external faces of the voxels in the structure, and, unlike robots in other robotic construction systems, is not constrained by gravity: it can walk vertically on walls of the structure and even upside down on the ceiling. MMIC-I is an internal robot that can crawl through the structure and fasten voxels together using a bolting mechanism installed adjacent to its grippers. Working together in teams, these robots can assemble voxels into various structures, depending on the instructions the robots are given.

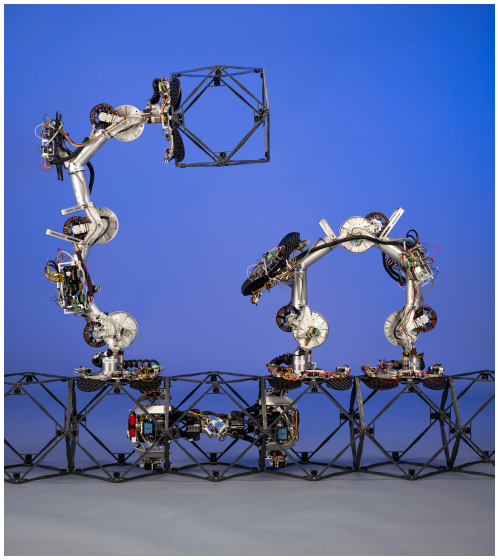


Fig. 1. Two types of ARMADAS robots on a row of voxels. A crane SOLL-E (left) is picking up a voxel from the backpack of a cargo SOLL-E (right). MMIC-I (inside the structure) is poised to crawl into the next voxel placed by the crane SOLL-E and fasten the voxel to the structure.

While the benefits are numerous, the task of autonomous assembly of large and complex structures presents a non-trivial algorithmic challenge. For structures consisting of thousands of building blocks, it is impossible to optimize the assembly plan by hand, and the level of sustained economic and exploration activity envisioned for in-space and on lunar presence by the government and commercial partners would require structures on a much larger scale [4]. This shortfall in current capabilities is the directive motivation for this work, and in this paper, we present an algorithmic framework for a multi-robot assembly system to address exactly that.

A. Algorithmic Approach to Assembly

Over the past decade, the topic of multi-robot construction has been gaining popularity in academic circles. The approach of having a large number of small and agile

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

[1]KBR, Inc., Coded Structures Lab, NASA Ames Research Center, Moffett Field, CA irina.kostitsyna@nasa.gov

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

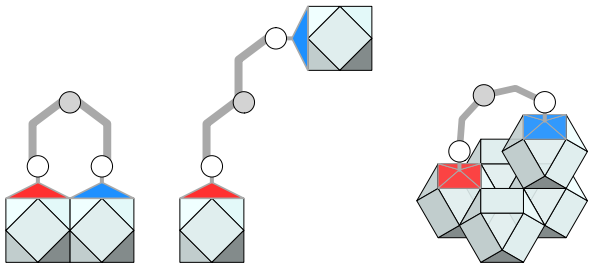


Fig. 2. Three examples of the 31 possible SOLL-E poses on the surface of a structure.

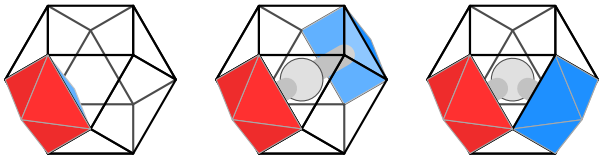


Fig. 3. The three poses of MMIC-I inside a structure, up to rotation.

mobile construction robots operating collaboratively makes for an appealing alternative to traditional assembly lines with their massive and expensive robotic arms. An already classic example of such an assembly system is TERMES [5], developed to perform construction by a fleet of car-like robots in a decentralized fashion. TERMES robots place box-shaped building blocks to form a modular structure on a grid.

Subsequent works have studied the combinatorial optimization question of construction planning for similar systems [6]–[8]. These systems operate under the constraint of gravity: robots must be supported by the building blocks underneath them, and the building blocks themselves can be placed only on top of other building blocks. Thus the gravity constraint limits the feasible geometries of the structures to the class of *histogram* shapes. With constraints on vertical traversal capabilities in these systems' robots (i.e., a robot can climb or descend a limited number of blocks in a single horizontal step, usually one or two), construction of some geometries requires the robots to construct and deconstruct ramp-like supports to access certain build regions. In some instances, such ramps can be constructed by rearranging existing building blocks, but in general, building the required supporting structures necessitates additional building blocks beyond those used in the final geometry.

A number of other systems feature two-legged robots manipulating cubic building blocks [9]–[12], but these are often similarly restricted by gravity.

In contrast, ARMADAS robots are not subject to the same gravity-based restrictions. SOLL-E can walk on walls and ceilings, and MMIC-I can crawl through a structure in any direction. As a result, the geometry of the ARMADAS structures is not limited to histogram structures, since the voxels are directly bolted to one another. The only restriction we make is that the structure must be face-connected.

All other systems mentioned above have one robot transporting one building block from source to destination. The ARMADAS system, however, separates the roles of transport and placement between robotic entities. This separation of robot roles was inspired by a desire to minimize the energy usage, in particular, by minimizing the mass associated with particular functionalities. For example, we avoid adding the bolting mass to robots that are required to repeatedly traverse long distances to and from the building block source [13]. Another potential benefit, although not considered in the current paper, is that a single crane robot can place building blocks while multiple other robots act as couriers to bring material from the source. While separating the robot roles increases build efficiency and versatility of the system, this also increases the challenge and complexity of path planning for the robots.

Closely related to the construction systems studied in this paper are the abstract models of modular reconfigurable robots explored in the algorithmic community [14]–[17]. These works explore questions of universal reconfiguration: is it possible to reconfigure from any modular shape to any other shape? Recent work [18] introduces a reconfiguration model with realistic movement constraints motivated by the ARMADAS system. The authors show that with the use of additional voxels as a scaffold, a structure of any shape can be assembled. In the case when no scaffold voxels are allowed, i.e., voxels are only ever added to the structure and never removed, there exist shapes that are impossible to assemble. Nevertheless, a large class of shapes, specifically shapes with the so called external-feature-size-2 property, can always be assembled in a monotone additive fashion.

Multi-robot path planning: One of the major tasks in multi-robot assembly planning is robot path planning. The robots have to travel from the depot, where they pick up new voxels, to the location of their placement, all while avoiding collisions and deadlocks with other robots. Multi-robot path planning is a vastly researched topic, with approaches ranging from exact solutions (e.g., A* planning in highly-dimensional state spaces, conflict-based search [19] and its variations) to heuristic solutions (e.g., decoupled approaches [20], AI-based approaches [21]).

Particularly relevant to our setting is the lifelong multi-robot path planning [22], where robots receive new tasks and targets upon finishing previous ones. This setting renders the exact approaches to planning impractical even for a small number of robots due to the exploding complexity of the problem state representation. The solution proposed in [22] uses a decoupled approach, iteratively constructing robots trajectories task by task and robot by robot. The authors introduce the notion of *endpoints*, serving as safe spaces for robots to park temporarily to avoid potential conflicts with new targets and future actions of other robots.

Another technique we utilize was presented in the context of the multi-labeled A* (MLA*) algorithm [23]. The state of a robot is incorporated into the search graph. This can be useful, for example, in the warehouse setting, where robots

already carrying a package have different state than the robots moving to pick up a package. Since certain varieties of our robots transport building blocks, it is easy to see how this inclusion is appropriate.

B. Problem Statement

The robotic part of the ARMADAS system consists of multiple teams of robots, each team working independent of others. A team is composed of a *cargo* SOLL-E robot, a *crane* SOLL-E robot and one MMIC-I robot. The cargo SOLL-E is responsible for picking up a new voxel at the depot location and bringing it in its backpack to the assembly location, where the crane SOLL-E picks the voxel up from the cargo's backpack and places it on the structure. Then MMIC-I enters the new voxel and fastens it to the structure along every face adjacent to the existing structure.

Given a target structure S, and k teams of robots, our goal is to find an efficient construction plan for the robots to build S while avoiding collisions and deadlocks. We assume that the initial state of the system includes a base structure with a specified depot position, and a supply of voxels ready to be served to cargo robots at the depot. The role of the base structure is to support the robots' initial positions, as well as serving as a seed structure from which the target structure will be assembled by attaching voxels one by one.

Deciding on the order in which S is to be constructed is a non-trivial task in its own right. We present our approach to this question in the accompanying article [24]. In this paper, we assume that the coordinates of placement of each next voxel can be queried during the course of assembly. The coordinates can be taken from a predetermined sequence, or they may be computed at each iteration based on the state of the intermediate structure and the robots' positions.

II. PATH PLANNING FRAMEWORK

In this section we present the path-planning approach for the ARMADAS system. Many grid-based multi-robot path planning approaches [6]–[8], [10] operate under the simplifying assumption that each robot fits into one cell of the grid. However, ARMADAS robots occupy multiple grid cells, which makes system state representation more complex, and both, collision detection and deadlock avoidance, more challenging. To address these challenges, we develop generalizations of several approaches to multi-robot path planning and combine them into the path-planning framework presented here. Despite our framework being developed specifically for the ARMADAS system, the underlying algorithms can be applied to other robotic assembly systems that feature a robot / building block codesign.

One approach to multi-robot motion planning is to consider the robots as one *meta*-agent, and represent their states in a high-dimensional state space. This reduces the task to shortest path planning in the graph encoding the states of the meta-agent and transitions between them, and results in an exact optimal solution to the problem. Unfortunately, even for one team of ARMADAS robots and a very small structure, the number of states in the meta-agent state space can be on the order of billions. We therefore must sacrifice optimality of the solution in favor of efficiency of computation. We decouple the state representation of the individual robots, and take the approach of planning each next step in the construction process for each robot individually. In order to avoid collisions, we develop a data structure that tracks robots' paths and reserves the occupied grid cells during the corresponding time intervals. To avoid deadlocks, we use an approach similar to the one proposed in [22], where we introduce a *parking* location reserved for each robot, which the robots can use to avoid blocking others.

State Space: The positions of the robots on and inside a structure form a discrete set of poses (see Figs. 2, 3 and the Appendix). We represent the state of a MMIC-I robot with an *internal* graph G_i, with nodes corresponding to all poses of the robot inside the structure and with edges corresponding to the basic elementary transitions between these poses (e.g., MMIC-I moving one of its sides to grip to a different voxel). Similarly we introduce a *surface* graph G_s representing the states and possible moves of a SOLL-E robot.

To distinguish between a cargo robot carrying a voxel and one with an empty backpack, we use an approach presented in [23]. We add a flag $withVoxel$ to the nodes of the surface graph, i.e., we make a second copy of the graph, with the flag set to $true$ in one copy and to $false$ in the other. The transition from a node with $withVoxel = false$ to $withVoxel = true$ happens only at the *depot* node. Thus, when planning for a cargo robot to go to the depot, pick up a voxel, and bring it to the construction site, we simply search for a path in the surface graph from a node corresponding to the cargo's current position with $withVoxel = false$ to the corresponding node at the construction site with $withVoxel = true$. This approach, as opposed to concatenating two paths to and from the depot, avoids potential deadlocks at the depot location.

For the decoupled robot path planning, we need to make sure that the robots trajectories do not cross in space and time. Rather than planning paths in the surface and internal graphs presented above, we make timed versions of these graphs. Let $t \in 0, \ldots, T$ for some large T. Let the timed surface graph G_s^T contain T copies of the nodes $V(G_s)$, one per time instance t. Denote a copy of a node $v \in G_s$ at time t as v_t. For each edge $e = (u, v) \in G_s$, we connect u_t to v_{t+1} for all $0 \leq t < T$ with a directed edge. Now, planning a path for a SOLL-E from a location u at time τ to a location v would correspond to searching for a path in G_s^T from u_τ to any of the nodes $\{v_t\}$. Similarly, we introduce a timed internal graph G_i^T for MMIC-I robots.

As the structure changes in time, the graphs G_s^T and G_i^T must be updated correspondingly. When a new voxel b is attached to the structure at time τ, it leads to a certain number of nodes in G_s^T being deleted. Specifically, all the nodes corresponding to SOLL-E poses blocked by b must be deleted from G_s^T for all $t \geq \tau$. Similarly, attaching b creates new nodes in both graphs. As the time of attaching new voxels depends on the lengths of the robots trajectories, updating G_s^T and G_i^T must occur during path planning.

Algorithm 1: Outline of the multi-robot assembly planning algorithm.

input : a base structure with a specified depot, initial robots positions, a target structure and its construction order

initialize global G_s^T and G_i^T // the timed surface and internal graph
initialize global $R \leftarrow \{$*initial robots poses at* $t = 0\}$ // the RESERVATIONS data structure
initialize global $S \leftarrow \{\}$ // initialize schedule
initialize parking locations to the robots' initial positions
while *structure is incomplete* **do**
    $next \leftarrow$ coordinates of the next voxel to be attached
    $team \leftarrow$ robot team with earliest available time
    `PlanOneTeamStep(`*team, next*`)`
end
`PlanOneTeamStep(`*team, next*`)`**:**
    current_pos $\leftarrow$ last robots' positions from $S[team]$ // nodes of G_s^T and G_i^T
    $h \leftarrow$ `ChooseHandoffPoses(`$next$`)` // accordingly selected tuple of handoff poses
    $\pi_1 \leftarrow$ `FindAndReservePath(`*current_pos.cargo, h.cargoFeedPose*`)`
    $\pi_2 \leftarrow$ `FindAndReservePath(`*current_pos.crane, h.craneWaitPose*`)`
    $\pi_m \leftarrow$ `FindAndReservePath(`*current_pos.mmici, h.mmiciWaitPose*`)`
    synchronize the robots // some robots may need to wait for others to arrive
    update $S[team]$ with π_1, π_2, π_m
    update $S[team]$ with handoff poses
    reserve paths to robots' parking locations // the schedule is not updated with these paths

Reservations Data Structure: For a robot to be in a certain location on (or inside) the structure, its corresponding pose must be *valid*. That is, the grid cells overlapping with the body of the robot must be empty of other robots, and, for SOLL-E robots, must be empty of voxels.

To avoid robot collisions, we use a data structure to keep track of robots trajectories in time and the corresponding cells occupied by the robots. Note, that the set of occupied cells depends on the pose of a robot as well as on whether the robot has a voxel present in its backpack. The RESERVATIONS data structure keeps track of the grid cells and the time intervals when they are occupied by one of the robots.

Parking Locations: Consider the following scenario. Two teams of robots are working on construction of a structure. The first team has just placed and fastened a new voxel. The second team needs to compute its robots' trajectories to bring and attach the next voxel. If the two voxels are placed in the neighboring locations, the first team of robots needs to move away to give space to the second team to work. We must be careful in deciding where to move the first team, as their new positions must not block the access of the second team. If there are multiple teams working on the construction, the choreography between all the teams becomes even more complicated. To resolve this issue, we specify certain locations, called parking locations, on the structure as each reserved for one specific robot. Robots can always go and stay indefinitely in their assigned parking location with the guarantee that they will not interfere with other robots. Following the approach of [22], for each robot r, we always maintain a trajectory for r from its current location to r's parking location reserved in the RESERVATIONS data structure. This ensures that robots are never blocked and can always move away. Note, that r would never actually take the trajectory to go the parking location in the middle of the construction process. The reserved trajectory simply serves as a guarantee that r always has a way out of its current location, no matter where the other robots are.

Handoff Poses: When deciding on how the robots are to place the next voxel, we need to consider various combinations of the relative positions of the crane SOLL-E, cargo SOLL-E, and MMIC-I. The crane must be able to reach the voxel in the cargo's backpack, pick it up, rotate to align with the location of the voxel's attachment, and put the voxel in place (refer to Fig. 1). Afterwards, MMIC-I enters the voxel and fastens each face of adjacency to the structure. For each valid combination of handoff poses, we define $cargoFeedPose$ to be the pose in which the cargo robot is serving the voxel to the crane, $craneWaitPose$ to be the pose in which the crane robot is waiting for the cargo to get into position, and $mmiciWaitPose$ to be the pose in which MMIC-I is waiting to start the bolting procedure. Additionally, we specify the transition poses for the crane from the wait pose to the *grab* pose (the pose in which it picks the voxel up from the cargo), and from the grab pose to the *build* pose. Similarly, we specify the transition poses for the internal robot between the bolting positions. For the handoff tuple to be valid, all the poses in the choreography of the three robots must be disjoint and the cells they overlap with must not be reserved by other robots.

Construction Algorithm: We are now ready to put together the above described components. An outline of the algorithm is presented in Algorithm 1. After initializing all the data structures, the algorithm iteratively constructs the next portion of the schedule for a team with the currently shortest plan, i.e., a team who least recently placed their last voxel (see the `PlanOneTeamStep` procedure in the algorithm description).

Consider an i-th iteration of the algorithm. For each $team$, let $\tau(team)$ be the length of its current schedule in $S[team]$. We maintain an invariant that before each call

to the `PlanOneTeamStep` procedure, the states of the data structures are the following: (a) for each $team$, the schedule $S[team]$ has the sequence of nodes in G_s^T and G_i^T corresponding to valid trajectories for the robots to attach a sequence of voxels assigned to them; and (b) the RESERVATIONS data structure has correct reservations for each team from $t = 0$ to $t = \tau(team)$, and from $t = \tau(team)$ some paths to the team's parking locations are also reserved.

During the i-th iteration, the team being processed by `PlanOneTeamStep` is given the coordinates of the next voxel to be built; see the accompanying article [24] for discussion of how this choice can be made. We choose a tuple of handoff poses to build the next voxel. Note, that for a handoff tuple to be valid, the placement of the voxel should not interfere with the existing trajectories of other robots during the voxel placement or anytime in the future. Therefore, for each handoff tuple, we identify the earliest time t' such that the grid cells occupied by the handoff poses and the voxel itself are not reserved for any $t \geq t'$.

Among all valid combinations, we select the tuple with the smallest t', and among those the one with the shortest distance between the depot to $cargoFeedPose$. Note, that this may not be an optimal choice due to the current state of the RESERVATIONS data structure, as we cannot guarantee that the cargo robot would arrive to that particular $cargoFeedPose$ the fastest. Nevertheless, we find that in general this leads to a good choice of a handoff tuple.

For each robot in the team, using A* algorithm we find a shortest path in the appropriate graph (G_s^T for SOLL-E or G_i^T for MMIC-I) from its current position to the corresponding pose of the handoff tuple, such that the path is disjoint from the current reservations made by other robots. We add the robots' poses along these trajectories to the RESERVATIONS data structure, and update the team's schedule $S[team]$. Next, we add poses corresponding to the voxel's hand-off and fastening to the schedule and the RESERVATIONS data structure. Finally, we find and reserve paths to the robots' parking locations, which must exist, as long as we have selected a valid handoff tuple. Indeed, the robots of other teams cannot block the paths to the parking locations, as at time $t = \max_{team} \tau(team)$ all other robots are guaranteed to be at their respective parking locations. Therefore, if there is no path to its parking for one of the current team's robots, it is due to the relative positions of its teammates during the hand-off and voxel assembly procedure. As long as we ensure that $cargoFeedPose$ and $craneBuildPose$ are both reachable from their parking locations after the voxel is attached to the structure, the robots will always be able to find a path to their parking location. Thus, the invariant is maintained after the `PlanOneTeamStep` as well. At the end of the execution of the algorithm, the schedule structure will contain trajectories for the teams of robots to assemble the target structure.

Implementation details and A heuristics:* Next we provide a few comments on the specifics of our implementation of the presented algorithm. In Section III we present an experimental evaluation of the algorithm.

First, it is not feasible to store the graphs G_s^T and G_i^T explicitly. Instead, we use their implicit representation, where for each node, knowing the current state of the structure, all valid neighboring nodes can be computed. Thus, we maintain an auxiliary data structure that stores the snapshots of the structure during the time interval $[\min_{team} \tau(team), \max_{team} \tau(team)]$ and returns its state for any given query time in this interval.

We have implemented two kinds of heuristic functions for the A* algorithm, basic heuristics, used for the crane SOLL-E and MMIC-I, and a heuristic with bookkeeping, used for the cargo SOLL-E. The basic heuristic for the crane SOLL-E is given by the following formula:

$$h_s(x) = \frac{3}{4}\left(dist_{L_1}(x.foot_A, target.foot_A) + dist_{L_1}(x.foot_B, target.foot_B)\right),$$

and MMIC-I:

$$h_i(x) = \frac{1}{2}\left(dist_{L_1}(x.side_A, target.side_A) + dist_{L_1}(x.side_B, target.side_B)\right).$$

To make the path search more efficient, we have also introduced a more sophisticated heuristic for the cargo SOLL-E:

$$h_s^{bk}(x) = dist_{G_s}(x, depot) + dist_{G_s}(depot, target),$$

where $dist_{G_s}(\cdot,\cdot)$ is the exact distance between two nodes in the a static surface graph G_s that is a slice of G_s^T at $t = \tau(team)$, i.e., the moment when the computation is taking place. To be able to quickly evaluate $dist_{G_s}(\cdot,\cdot)$, in addition to the dynamic structure, we store the dynamic distances from any node to the depot, which can be queried at any time in the interval $[\min_{team} \tau(team), \max_{team} \tau(team)]$. To maintain the dynamic distances data structure, we update the distances (within the static graph G_s) from all nodes to the depot at every event when a voxel is being attached.

III. ALGORITHM EVALUATION

We evaluate the performance of our algorithm on two structures shown in Figure 4. The first structure is a block of size $4 \times 5 \times 10$ voxels, and the second structure is an

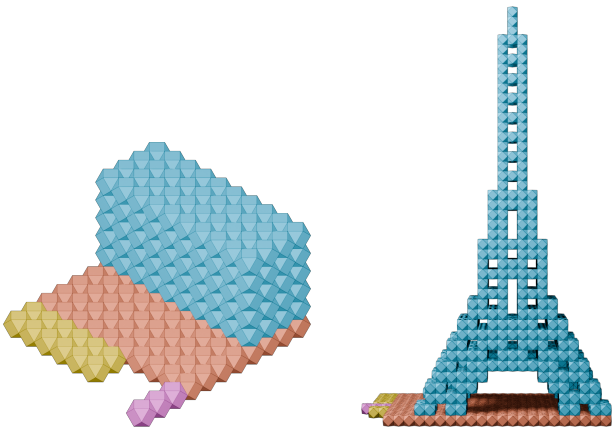


Fig. 4. Two test structures: a solid $4 \times 5 \times 10$ block (left), and an Eiffel tower consisting of 1324 voxels (right).

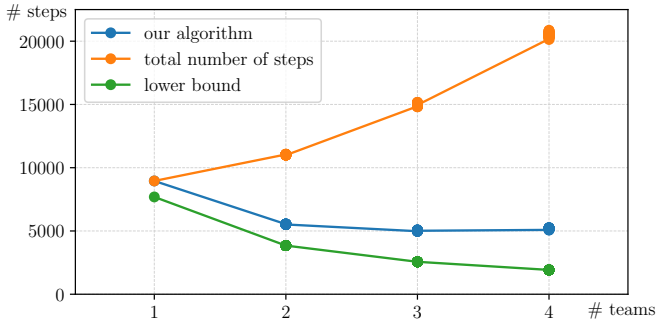


Fig. 5. The number of synchronous steps to assemble the block structure as a function of the number of teams: with our algorithm (blue), the lower bound (green). The total number of steps taken by the robots (orange).

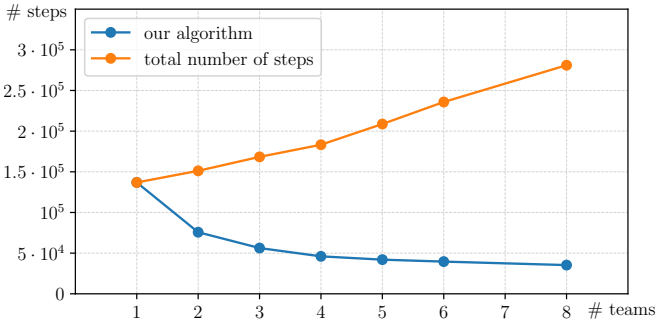


Fig. 6. The number of synchronous steps to assemble the Eiffel tower structure with our algorithm (blue), as a function of the number of teams, and the total number of steps taken by robot teams (orange).

Eiffel tower comprised of 1324 voxels. In the figure, voxels of the base structure are shown in orange, the depot location is shown in purple, the robot's parking locations are shown in yellow, and the structure itself is shown in blue. The chosen structures exemplify two distinctly contrasting topologies: a solid body and a sparse configuration. Other possible designs serve as transitional forms between these extremes, thereby allowing for a continuum of results to be deduced.

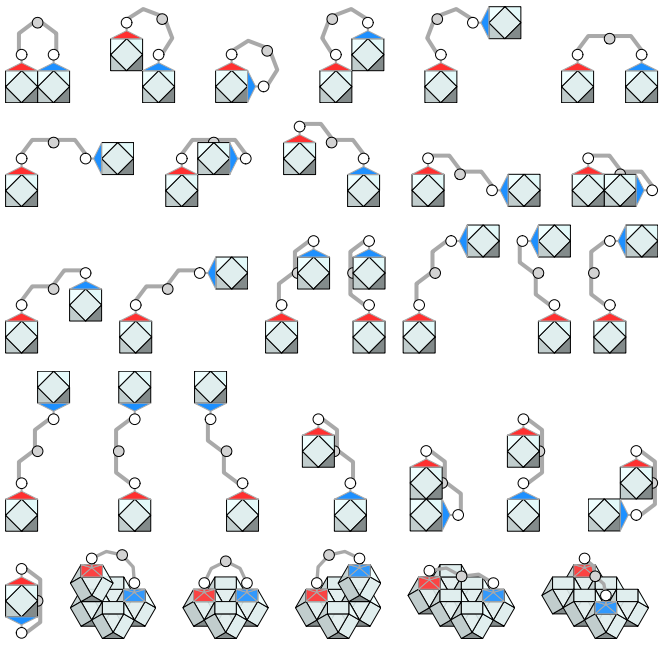
We first evaluate the performance of our algorithm on the smaller block structure. To estimate how well the algorithm performs compared to an optimal solution, we have implemented a path planning algorithm for one team with a meta-agent consisting of the two SOLL-E robots and a MMIC-I robot. As described above, it is unfeasible to implement a meta-agent approach that groups all three robots in one meta-agent. Our approach of grouping two SOLL-E robots into a meta-agent and having MMIC-I planning performed independently is close to an optimal solution. Indeed, MMIC-I trajectories rarely interfere with the SOLL-E trajectories, and the choreography of the two SOLL-E robots is computed to optimality when considering them as a one higher-dimensional robot. For the given structure, one team with a SOLL-E meta-agent and a MMIC-I took 7690 steps to construct the block structure. This value serves as our lower bound on the number of steps for one team of robots. For the case of k robot teams with $k > 1$, we divide 7690 by k to obtain a lower bound. Figure 5 shows the comparison of our algorithm (in blue) to the described lower bound (in green). In orange is shown the total sum of the number of steps performed by each team. The growth of this function is explained by the amount of interference between the multiple robot teams. This is due to the very limited amount of space the robots have to operate within, in the case of the block structure. The level of interference also explains the divergence of the time span of our algorithm from the lower bound. Nevertheless, the performance of our algorithm is surprisingly close to the lower bound for one and two teams, which indicates that for larger structures our algorithm should perform well.

Next, we evaluate the performance of our algorithm on the larger Eiffel tower structure. Figure 6 shows the comparison of our algorithm (in orange) to the total number of step taken by all the robot teams. Compared to the block structure scenario, in this case our algorithm shows a more rapid relative decrease in the time span with the increasing number of teams. As expected, this is due to less interference between the robots from different teams. The geometry of the structure allows for parallel construction most of the time without much interference, except for a bottleneck at the depot location where cargo robots pick up voxels. Observe that, consistently with the above, the orange function grows relatively more slowly compared to the block scenario.

In conclusion, the presented framework for a multi-robot assembly system can be successfully employed for construction of large modular structures, with sizes in the order of tens of thousands of voxels. To achieve a system scalable to much larger structures, however, one would be required to consider distributed computing approaches rather than a centralized approach, like the one presented in this paper, which is an interesting direction for future research.

APPENDIX

All possible poses (up to rotation) of SOLL-E robot with both feet gripping to a voxel face on the structure's surface:



References

[1] 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, 2024. doi: 10.1126/scirobotics.adi2746

[2] 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.

[3] 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.

[4] "Moon to Mars architecture." [Online]. Available: https://www.nasa.gov/moontomarsarchitecture/

[5] 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.

[6] E. Lam, P. J. Stuckey, S. Koenig, and T. S. Kumar, "Exact approaches to the multi-agent collective construction problem," in *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings 26*. Springer, 2020, pp. 743–758.

[7] 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.

[8] M. Rameš and P. Surynek, "Action duration generalization for exact multi-agent collective construction," in *16th International Conference on Agents and Artificial Intelligence (ICAART)*, 2024.

[9] Y. Terada and S. Murata, "Automatic assembly system for a large-scale modular structure - hardware design of module and assembler robot," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004. doi: 10.1109/IROS.2004.1389760 pp. 2349–2355.

[10] ——, "Automatic assembly system for modular structure," in *Proc. 22nd International Symposium on Automation and Robotics in Construction (ISARC)*, 2005. doi: 10.22260/ISARC2005/0028

[11] B. Jenett, A. Abdel-Rahman, K. Cheung, and N. Gershenfeld, "Material–Robot System for Assembly of Discrete Cellular Structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4019–4026, 2019. doi: 10.1109/LRA.2019.2930486

[12] 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.

[13] B. Bernus, G. Trinh, C. Gregg, O. Formoso, and K. Cheung, "Robotic specialization in autonomous robotic structural assembly," in *2020 IEEE Aerospace Conference*, 2020. doi: 10.1109/AERO47225.2020.9172620 pp. 1–10.

[14] Z. Abel, H. A. Akitaya, S. D. Kominers, M. Korman, and F. Stock, "A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves," in *40th International Symposium on Computational Geometry (SoCG 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 293, 2024. doi: 10.4230/LIPIcs.SoCG.2024.1. ISBN 978-3-95977-316-4. ISSN 1868-8969 pp. 1:1–1:14.

[15] I. Kostitsyna, T. Ophelders, I. Parada, T. Peters, W. Sonke, and B. Speckmann, "Optimal in-place compaction of sliding cubes," in *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 294, 2024. doi: 10.4230/LIPIcs.SWAT.2024.31. ISBN 978-3-95977-318-8. ISSN 1868-8969 pp. 31:1–31:14.

[16] 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.

[17] D. Feshbach and C. Sung, "Reconfiguring non-convex holes in pivoting modular cube robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6701–6708, 2021. doi: 10.1109/LRA.2021.3095030

[18] 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.

[19] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015. doi: https://doi.org/10.1016/j.artint.2014.11.006

[20] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[21] J.-M. Alkazzi and K. Okumura, "A comprehensive review on leveraging machine learning for multi-agent path finding," *IEEE Access*, vol. 12, pp. 57 390–57 409, 2024. doi: 10.1109/ACCESS.2024.3392305

[22] H. Ma, J. Li, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proc. 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2017. doi: 10.5555/3091125.3091243 pp. 837–845.

[23] 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 (ICAPS)*, vol. 29, no. 1, 2019. doi: 10.1609/icaps.v29i1.3474 pp. 181–185.

[24] T. Peters, K. Cheung, and I. Kostitsyna, "Assembly order planning for modular structures by autonomous multi-robot systems," *to appear at ICRA 2025*, 2025.