Multi-agent Collective Construction using 3D Decomposition

Akshaya Kesarimangalam Srinivasan*, Shambhavi Singh†, Geordan Gutow*,
Howie Choset* and Bhaskar Vundurthy*

Abstract— Consider a Multi-Agent Collective Construction (MACC) problem that aims to generate a plan for fictitious cubic *robots* to build a three-dimensional structure comprised of cubic *blocks*. These cubic robots can carry one cubic block at a time; robots may move left, right, forwards, backward, or climb up or down one block. To construct structures taller than one cube, the robots must build supporting scaffolding made of blocks and remove the scaffolding once the structure is built. Prior works sought to create a planner that considered the structure as one monolithic assembly, which becomes intractable for larger workspaces and complex structures. To this end, we present a decomposition algorithm that breaks the structure into substructures that can be planned for independently. We use Mixed Integer Linear Programming (MILP) to plan for each of these substructures and then aggregate the solutions to construct the entire structure. Extensive testing on 200 randomly generated structures shows an order of magnitude improvement in the solution computation time compared to an MILP approach without decomposition. Finally, we leverage the independence between substructures to detect which substructures can be built in parallel.

(a) Input structure   (b) Structural decomposition

(c) Ordering of substructures   (d) Parallelizable construction

Fig. 1: Visualization of our solution to the MACC problem.

I. INTRODUCTION

The paper considers a Multi-Agent Collective Construction (MACC) problem that aims to construct a given three-dimensional structure of homogeneous cubic blocks using a set of robots in a voxel world in the minimum *makespan* (number of time steps) [1]. Previous solutions to the MACC problem include optimization [1], heuristics [2][3], [4], and Reinforcement Learning (RL) [5], [6]. Because of the computational complexity of this problem, all prior work is forced to trade off plan quality and run time. Most existing works [5], [2] produce highly sub-optimal plans in exchange for extremely fast run times; in contrast, [1] yields a globally optimal plan but is very slow. This paper introduces a technique that can produce near-optimal plans with shorter runtimes compared to [1].

Our main contribution is an algorithm that decomposes a structure into substructures (see Fig 1), whose construction may be planned independently, and finds an order in which they can be built. The resulting construction plan requires a larger makespan than the optimal solution but requires an order of magnitude less computational time. This suboptimality is mitigated by identifying which substructures can be built simultaneously and overlapping their construction.

We start with the problem formulation in section II and present a brief review of existing approaches in section III. Section IV details our decomposition and ordering algorithms and discusses parallelization to achieve further speedup. Numerical results are presented in section V while section VI concludes the paper and proofs are presented in the appendix.

II. PROBLEM FORMULATION

The MACC problem formulation in this work is identical to that considered in [1]. It is set in a 3D voxel world containing a predefined structure composed of cube-shaped building *blocks* and block-sized *robots*. The robots collaboratively construct the structure by moving these blocks using any of the following permissible actions (Fig. 2) in a given time step:

1) Move one step in the four compass directions or wait
2) Climb up or down one block at an adjacent cell
3) Pick up or place a block at an adjacent cell of the same height

Except for blocks on the ground level, all blocks must have a block beneath them, i.e., no blocks may "float" in space. Robots can only pick up or place down the topmost block at any location. They must therefore construct scaffolding to access cells not surrounded by blocks of the same height, but the structure is not considered complete until all scaffolding has been removed. An unlimited supply of blocks is available at the boundary of the grid world, and robots that exit the
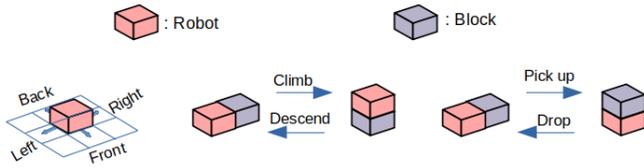
Fig. 2: Permissible actions for the robots

grid world can enter from any boundary cell in the next time-step. The world is initially empty, and the robots start and finish outside the grid world. The task is to find a sequence of actions for the robots to build the structure subject to the rules above. Algorithm performance is compared via three metrics:

1) *Computation Time*: time taken to compute the action sequence for building the structure
2) *Makespan*: total number of time steps to complete building the structure
3) *Sum of costs*: total number of actions used by all the robots

III. RELATED WORKS

Interest in using robots for building construction and assembly has led to a variety of proposed technologies [7], including additive manufacturing techniques [8], auto-mated robotic assembly [9], and automated bricklaying [10]. A particular focus area is settings like open pit mining and extraterrestrial or underwater construction where human presence is difficult or dangerous [11][12][13][14]. In these applications, teams of smaller robots have the potential to be more effective than a few larger robots as they are cheaper, easier to deploy, and can work in parallel [2][15][16]. Considerable prior work has therefore addressed how to plan for and coordinate such robot teams.

MACC has been explored in both two [17] and three-dimensional worlds with varying types of agents and blocks. For example, teams of quadrotors build structures made of beams and columns in [18]. Harvard's TERMES considers homogeneous blocks and agents [19] to demonstrate the effectiveness of teams of small robots at collectively building structures much larger than themselves. Subsequently, [2] casts the predefined structure as a matrix with each element representing the number of blocks at that location. Then it seeks to minimize the number of pick-up and drop-off operations for a single agent by restricting agent movements to the edges of a minimum spanning tree derived from this matrix. This forces the robot to retrace precomputed paths to multiple destinations adversely affecting the makespan. A distributed multi-agent reinforcement learning algorithm used in [5] extends single-agent advantage actor-critic to enable multiple agents to learn a homogeneous, distributed policy. However, this approach does not generalize well to unseen environments or a larger number of agents.

The work most closely related to the current effort is the optimization approach presented in [1]. The MACC problem is solved using MILP or Constraint Programming (CP). The

MILP model treats all robots as one flow through a time-expanded graph. Each decision variable defines the actions of a robot. The CP approach uses a simpler network flow, modeling the specifications of the world as logical and element constraints. Both formulations first find the minimum makespan, then an action sequence with that makespan that minimizes the sum of costs. The MILP approach guaran-tees an optimal makespan and the optimal sum of costs for that makespan. The paper tested both approaches on six structures, and for these instances, MILP required less computation time than the CP formulation. The approach presented in Section IV calls this MILP formulation as a subroutine. The low achievable makespans of the structures studied in [1] indicate that small structures are simple to construct. However, even for some of the six small structures, the optimization models needed up to five days to compute a solution. As will be demonstrated in section V, MILP solution computation time is excessive for complex struc-tures. Thus, we need approaches with a practical solution computation time for structures of varying complexity.

There has been some work, including [20], [21], for solving these large MILP problems. [21] uses a two-level approach to make large MILP problems with binary variables tractable. It first groups variables and forms a semi-coarse model. It then aggregates constraints by partitioning them into groups and adding the violated constraints to the semi-coarse model iteratively till all the constraints in the full model are added. Inspired by 3D model decomposition work [22], this paper reduces the number of variables in the MILP problem by solving for one substructure at a time. Constraints are aggregated at each stage to represent the intermediate structure to be built.

IV. APPROACH

A decomposition algorithm is proposed to break input structures into simpler substructures. A bottom-up planner then determines an order in which the substructures can be built. MILP then computes an optimal sequence of actions for every substructure [1].

A. Structural Decomposition

In a 3D voxel world of dimensions $(X \times Y \times Z)$, we denote a predefined structure S using $\bar{z}(x,y)$ where $\bar{z}$ indicates the number of blocks stacked at grid location (x,y) and z points to the height of a specific block, $x \geq 1, y \geq 1, z \geq 0, \bar{z}(x,y) \geq 1$ and $x \leq X, y \leq Y, z \leq \bar{z} \leq Z$. We begin by ensuring that all predefined structures are valid.

Definition 1: A structure S is **valid** if, for every block $B_1 \in S$ at (x,y,z) with $z > 1$, there exists $B_2 \in S$ at $(x,y,z-1)$.

Note that robots can use blocks at a lower height as scaffolding for higher blocks. As a result, it is preferable to ensure any useful blocks (for scaffolding) are part of the same substructure as the higher block under consideration. This minimizes duplication of efforts for building the tem-porary scaffolding. We use the notion of a shadow region to capture this relationship.

Algorithm 1: Structural Decomposition

Data: $\bar{z}$
Result: Set of substructures $S_1, S_2, \cdots$

1   *towers* ← elements of $\bar{z}$ in decreasing order; $i \leftarrow 1$;
2   **for** *h in towers* **do**
3   **if** *Topmost block of h not in a substructure* **then**
4   Initialize substructure $S_i \leftarrow \emptyset$;
5   Shadow indices, $s_{idx} \leftarrow$ cells in shadow region of tower h using **Definition 2**;
6   **for** *s in s_{idx}* **do**
7   **if** *s is not in any substructure* **then**
8   add s to S_i;
9   **end**
10   **end**
11        $i \leftarrow i + 1$
12   **end**
13 **end**

Fig. 3: Substructures identified by Algorithm 1

Definition 2: For a **tower** of height z located at (x, y), i.e, z blocks stacked at (x, y), all cells (x', y', z') s.t.

$$|x - x'| + |y - y'| < z$$
$$z' \leq z - (|x - x'| + |y - y'|)$$

are part of the **shadow region** of the tower.

We now present **Algorithm 1** that decomposes the input structure by iterating through all the towers in decreasing order of their heights. At each step, blocks in the shadow region of the tower that are not already part of another substructure become part of the current substructure. Substructures are inherently associated with a specific order in that the validity of a substructure depends on which substructures are already present in the world.

Definition 3: Let $S' = \bigcup_{i=1,2,\cdots,j-1} S_i$ be a valid structure. Then S_j is a **valid substructure** if $S' \cup S_j$ is also a valid structure.

Remark 1: Henceforth, the index of a substructure refers to the order it was found by the decomposition algorithm.

Theorem 1: Each substructure in the order it is generated by **Algorithm 1** is a valid substructure.

B. Bottom Up Planning

For the input structure shown in Fig. 1a, note that **Algorithm 1** starts with a tower of height 3 and generates five valid substructures (see Fig. 3). The validity of a substructure, say S_3, can be verified by assuming that structures S_1 and S_2 are already present and checking for the validity of structure $S_1 \cup S_2 \cup S_3$. This is true for all five substructures, as indicated by **Theorem 1**.

While there are 120 possible orders for constructing these five substructures, not all orders allow successful completion while adhering to the permissible actions (Section II). For instance, consider the order S_2, S_3, S_4, S_1, and S_5. Substructure S_3 relies on the construction of S_1 while S_5 cannot be built if all the remaining substructures are already constructed.

To find an order of construction of substructures, we define the traversability property for substructures. As with validity, this depends on which other substructures are already present in the world.

Definition 4: A substructure S_j is **traversable** if for every block $B \in S_j$ there exists a *feasible path* to at least one neighbor cell at the same height as the block B, in the presence of the structure $\bigcup_{i=1,2,\cdots,j-1} S_i$.

Definition 5: A **feasible path** is a sequence of permissible actions, including block placements and removals without disturbing blocks from a previously constructed substructure.

Definition 6: A sequence of d substructures is **feasible** if $S_i, \forall i \in \{1, 2, \cdots, d\}$, is *traversable* and $\bigcup_{k=1,2,\cdots,i} S_k$ is *valid*.

Obtaining a feasible sequence of substructures is treated as an assembly sequencing problem where the substructures are the components, and the input structure is the final product. The key idea is to build substructures in the reverse order in which they can be removed/disassembled from the goal structure, with every intermediate structure being valid. Such an order is obtained via a bottom-up planning algorithm inspired by [23], [24].

We now state the sufficient conditions to determine if a block $z(x, y)$ is removable:

1) There does not exist a block above it (at $(x, y, z + 1)$) from any remaining substructures
2) There exists a feasible path to a neighboring block of the same height in the current state of the environment
3) There exists enough space to build scaffolding to height z when there are no neighboring blocks

Condition 1 follows from the problem formulation (Section II) and Definitions 1 and 3. To check condition 2, a traversability matrix is constructed considering all existing substructures. In this matrix, the $(i, j)^{th}$ element is 1 if the (i, j) location is reachable from the boundary and is 0 otherwise. A cell is reachable if a neighboring cell is reachable and has a maximum height difference of one. The boundary cells are always considered reachable.

We use Dynamic Programming to determine the traversability matrix and compute contour polygons of unreachable cells (value 0). These contours represent impassable walls in the environment. Blocks are treated as not *removable* if they are enclosed on all four sides by blocks from another substructure or inside a contour of impassable walls. Note that in certain cases (the presence of a staircase on the wall's interior), scaffolding would, in principle, allow passing these walls. The following Lemma 1 addresses Condition 3.

Algorithm 2: Substructure Removability Check

Data: Substructure S_i
Result: Removability of S_i (True or False)

1 *Traversability Matrix* ← reachable positions in the (x,y) grid
2 *Countours* ← polygons representing impassable enclosures in the traversability matrix
3 **for** $B \in$ blocks of S_i **do**
4   **if** *B surrounded by blocks $\notin S_i$ in all four directions* **then**
5   **return** False;
6   **end**
7   **if** *B inside any contour $\in$ contours* **then**
8   **return** False;
9   **end**
10 **end**
11 **return** True;

Algorithm 3: Substructure Ordering and Merging

Data: O_d, reverse of substructure from **Algorithm 1**
Result: O_f, feasible sequence of substructures

1 $O_f \leftarrow \emptyset$;
2 **while** $O_d \neq \emptyset$ **do**
3   **for** S_i in O_d **do**
4   **if** *S_i is removable (**Algorithm 2**)* **then**
5   move S_i from O_d to O_f;
6   **end**
7   **end**
8   **if** *no substructure was removed and $O_d \neq \emptyset$* **then**
9   Merge the first two elements of O_d;
10   **end**
11 **end**
12 **return** reverse order of O_f

Lemma 1: In the absence of a neighboring block for $B \in S_j$, **Algorithm 1** ensures that there exists enough space to build scaffolding to reach B.

A substructure S_i can be considered removable if all of its blocks are removable. For a block B in S_i that has other blocks of S_i on top of it, the top blocks will be removed first while removing S_i, ensuring block B is removable. **Algorithm 2** utilizes the traversability matrix to determine if a substructure is removable.

In addition to determining the removability of a substructure, Condition 1 hints at the *dependency* of substructures on each other. For instance, if a block $B_i \in S_i$ is above a block $B_j \in S_j$, the removal of B_j cannot begin until B_i is removed from the environment. From the point of view of construction, S_i can be treated as being dependent on S_j for its construction. We thus have the following Definition 7.

Definition 7: A substructure S_i is said to be **dependent** on substructure S_j (denoted $S_i \rightarrow S_j$) if $\bigcup_{k=1,2,\cdots,i} S_k$ is a valid structure while $\bigcup_{\substack{k=1,2,\cdots,i \\ k \neq j}} S_k$ is not a valid structure.

Algorithm 3 couples the removability of substructures (**Algorithm 2**) with mutual dependency (Definition 7) to compute a feasible order for the construction of substructures. If no substructure is removable in an iteration, there exists S_i and S_j such that $S_i \rightarrow S_j$ but S_j prevents S_i from being traversable. Then, S_i and S_j are merged to get a traversable substructure. For instance, the order of substructures found by Algorithm 1 is S_1 to S_5 as shown in Fig. 3. The final feasible order generated by Algorithm 3 is S_1, S_2, S_5, S_3, S_4 as shown in Fig. 1c.

Lemma 2: The reverse of the order of removing substructures is a feasible assembly order.

Theorem 2: Given valid substructures, Algorithm 3 always generates a feasible sequence of substructures.

C. Solution Computation and Parallel Construction

Once the substructures and a feasible order are found, any of the conventional approaches (MILP[1], RL[5],

Tree-based[2]) can be used to find the sequence of actions to build each substructure. In this work, we utilize MILP to determine a plan to construct the structure with the following theoretical guarantees.

Lemma 3: Given a valid input structure, our decomposition and ordering algorithms, combined with MILP optimization for each substructure, return a feasible action sequence to build the entire structure.

Theorem 3: Consider a structure decomposed into d substructures. The makespan (number of time steps) to construct the sequence of substructures is no more than d times the makespan required to construct the structure without decomposition.

Further, when MILP is used, some substructures can be built in parallel to reduce the makespan. This is achieved by modifying **Algorithm 3** such that at every iteration, **all** substructures that are removable from the current state of the environment are determined before moving any substructure to O_f. These substructures can potentially be built in parallel.

Let $P = \{S_1, S_2, \cdots, S_p\}$ be a set of substructures that can be built in parallel. The first substructure in P is built as described for sequential construction. For every subsequent substructure S_i in P, the actions to build all previous substructures $S_j, j < i$ are added as a constraint to the S_i's MILP. This ensures that S_i's solution avoids agent-agent collision with the agents building the previous substructure and does not use more agents than permitted. In the worst case, this devolves to sequential construction (e.g. due to insufficient agents). Parallel execution reduces the makespan at the cost of added constraints in the MILP formulations for each substructure. Experiments show this does not significantly affect the solution computation time.

V. RESULTS

A. Experimental Setup

The effectiveness of the proposed algorithm is evaluated on: (a) six cases from [1] and [5], (b) one hundred random structures in a 10x10x4 grid world, and (c) one hundred random structures in a 7x7x4 grid world.

TABLE I: Comparison of the sum of costs of solutions for our approach with other non-optimal methods

Structure	Tree based [2]	RL based [5]	Ours
1	1144	3040	**179**
2	836	1026	**128**
3	1590	3056	**326**
4	2120	3252	**263**
5	2180	2804	**381**
6	836	1276	**161**

The MILP approach in [1] obtains the action sequence to build each structure or substructure. All models are solved using Gurobi 9.0.2, a SOTA solver for MILP on an Intel® Core™ i7-7700K CPU @ 4.20GHz × 8 with 94GB of memory. Each MILP model iterates through increasing makespans until the model becomes feasible and then performs optimization to find a solution. 'Solve Time' denotes the time taken to optimize the final feasible model, and 'Total Solve Time' adds the time to iterate through infeasible makespans. In every case, a limit of 10,000 seconds was set on 'Total Solve Time'. For each structure like the one in Fig. 1 **Algorithm 1** generates substructures (see Fig. 1b) and **Algorithm 3** identifies a feasible order for construction (see Fig. 1c). With a maximum of 20 robots, our approach solves this structure in 61 seconds with a makespan and sum of costs of 67 and 207 respectively.

B. Comparison on Six Test Structures

We next conduct experiments to compare our approach with two existing sub-optimal approaches [2] and [5] on six structures (see Table III) used in these works. Table I reports the sum of costs for a Heuristic-Based approach [2] and the average sum of costs for a Distributed Reinforcement Learning method [5] (over successful trials). It also reports the sum of costs using MILP with decomposition (our approach). Since the existing methods directly minimize only the number of pick-up and drop-off actions, our approach achieves costs up to an order of magnitude smaller than these methods on the six test structures.

Finally, we compare our approach with [1] which obtains the optimal sum of costs for a given input structure. Table III presents a comparison of [1] and our decomposition technique with and without parallelism. For all cases here, we limit the maximum number of robots to 20. Decomposition significantly improves the solution computation times over pure MILP while maintaining a similar sum of costs. However, the makespan increases due to the serial construction of substructures. Introducing parallel construction largely mitigates this increase as it reduces the number of time steps by an average of 46% compared to serial construction of the decomposed substructures.

C. Comparison with Exact Approach on Random Structures

We present a comparison of metrics for randomly generated structures in two grid world sizes: 10x10x4 and 7x7x4 with a maximum of 20 and 6 robots respectively. The chosen number of robots is sufficient to execute parallel construction in our experiments. 100 structures were generated with different occupancy percentages, i.e., the number

(a)

(b)
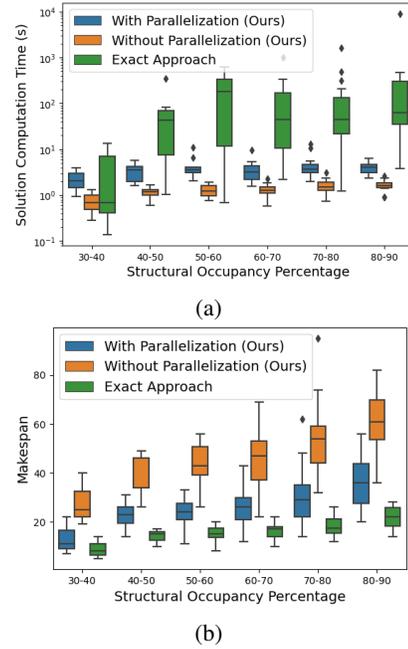
Fig. 4: (a) Computation Time Vs Occupancy Percentage, (b) No. of timesteps vs Occupancy percentage

TABLE II: Results for tests on random structures

A - Exact Approach [1], B - Decomposition (ours), C - Decomposition+Parallel (ours)

Environment Size	10x10x4			7x7x4		
Method	A	B	C	A	B	C
Sum of costs	-	384	**102.4**	83.7	97.1	**54.05**
Makespan	-	84	**56.2**	18.0	51.2	29.47
Solve Time (sec)	-	**48.1**	78.5	229.4	**1.5**	4.3
Total Solve Time (sec)	>10,000	**567.9**	787.4	423.5	**37.8**	126.03

of structure blocks divided by the size of the world. Half had an occupancy percentage between 40% to 60%, one-fourth had less than 40%, and one-fourth had more than 60%. The solutions for these structures are visualized in https://akshayaks.github.io/, and the average metrics are shown in Table II.

Fig. 4b shows that makespan increases with increasing occupancy percentage. On average, for the 7X7X4 environment, the number of time steps increased from 76 to 122 as the occupancy percentage increased from 30 to 70%. Note that the number of variables in the MILP (independent of the number of robots) is linear in the makespan and the solution computation time increases exponentially with the number of variables. Thus, decomposition is beneficial for run-time: solving smaller MILP models (one for each substructure that can usually be built in fewer time steps than the full structure) is much faster than solving one large model.

With increasing occupancy percentage, the improvement in the solution computation time also increases, as seen in Fig. 4a. Although the sum of costs remains comparable, the decomposition approach has a higher makespan than MILP across all occupancy percentages. However, makespan improves with the parallel construction of substructures as shown in Fig. 4b. Higher occupancy percentage structures

TABLE III: Results for the six test cases

Metrics for solutions obtained using with and without decomposition and parallelization on a set of six test structures used by [1] and [5]. **Bold** indicates improvement over [1]. A - Exact Approach [1], B - Decomposition (ours), C - Decomposition+Parallel (ours)

Test Structure									
Method	A	B	C	A	B	C	A	B	C
Sum of costs	173	176	179	124	128	128	-	**326**	326
Makespan	13	48	17	13	48	14	-	106	**44**
Solve Time (in sec)	1030.3	16.6	**11.1**	61.0	11.5	**10.2**	-	49.6	**26.4**
Total Solve Time (in sec)	1115.0	**241.3**	259.4	139.0	235.9	**198.1**	>10,000	377.2	**318.1**

Test Structure									
Method	A	B	C	A	B	C	A	B	C
Sum of costs	-	**204**	263	-	**365**	381	160	**153**	161
Makespan	-	113	**75**	-	130	**90**	21	50	40
Solve Time (in sec)	-	36.7	**31.2**	-	37.1	**27.8**	1215.3	15.6	**12.2**
Total Solve Time (in sec)	>10,000	**610.7**	758.6	>10,000	**639.4**	688.5	1715.2	**256.9**	287.9

thus show a greater reduction in makespan. Small, sparse structures get little benefit from decomposition as they only require a few time steps to be built and the original MILP problem is not very large.

VI. CONCLUSIONS

In this paper, we presented an algorithm to decompose any valid input structure into substructures and obtain a feasible order to build the substructures. Experimental results showed that MILP optimization with decomposition has an order of magnitude improvement in the solution computation time compared to MILP without decomposition. However, the former demonstrated greater makespan when the substructures were built sequentially or with basic parallelization.

Developing more sophisticated algorithms to construct substructures in parallel is a promising future direction. For example, one can determine the action sequence to construct each substructure that can be built in parallel. Then the action sequences can be post-processed to enforce constraints. Further, the decomposition into substructures can be modified to optimize metrics like the number of scaffolding blocks required or the parallelism provided. Finally, the similarity between substructures can be leveraged to calculate the action sequence required to build the latest substructure using solutions of previous substructures.

APPENDIX

Proof: [Theorem 1] Consider the decomposition algorithm described in Algorithm 1. The first substructure, S_1, found corresponding to the tallest tower in the structure, will contain all the blocks in the shadow region of this tower. By Definition 2, the shadow region includes all cells starting from the topmost block of the tower to all the lower-level neighbor cells till $z = 1$. Thus, adding S_1 to an empty

environment results in a valid structure, and hence S_1 is a valid substructure. Using induction with contradiction:

Base Case: For every block $B_1 \in S_1$ at height $z \geq 2$, $\exists B_2 \in S_1$ at height $(z-1)$. Thus S_1 is a valid substructure. Inductive Rule by contradiction: Suppose $S_1, \cdots, S_k$ were all valid substructures. Now suppose S_{k+1} is an invalid substructure. Then there exists a block B_0 below a block B_1 in S_{k+1} such that B_0 is not in any of the S_1 to S_{k+1} substructures. However, as per **Algorithm 1**, if B_0 were below B_1, and if B_0 was not a part of any of the previous k substructures, then B_0 will be a part of S_{k+1} (as B_0 and B_1 would belong to the same shadow region). Hence, by contradiction, B_0 is a part of substructure S_{k+1}, and $\bigcup_{i=1,2,\cdots,k+1} S_i$ is valid. Thus substructure S_{k+1} is a valid substructure. ∎

Proof: [Lemma 1] Consider two blocks $B_1 \in S_1$ and $B_2 \in S_2$, where scaffolding for B_1 at height z is obstructed by B_2. This implies B_2 is less than $z-1$ Manhattan distance away from the tower T_1 containing B_1. Then B_2 falls in the shadow region of T_1, indicating that $B_2 \in S_1$. Alternatively, if B_2 belongs to a tower taller than z, then $B_1, B_2 \in S_2$. Both of these scenarios contradict the assumption that $B_1 \in S_1$ and $B_2 \in S_2$. Consequently, if there is no neighboring block for B_1, there exist no blocks from other substructures to prevent the construction of the required scaffolding. ∎

Proof: [Lemma 2] The reverse of the order of removal can be used as a feasible order of construction if each removal operation is the reverse of a feasible assembly operation. This is ensured if the removal operation satisfies two criteria: task feasibility and structure stability [23]. Task feasibility is met if a collision-free incremental path exists to separate the substructure from the structure, and these actions are reversible. This follows from the sufficient conditions described in Section IV-B. Structure stability is true if the

two substructures remain joined after being assembled. This follows from Theorem 1. Consequently, the reverse of the order of removal of substructures is a feasible sequence for constructing the substructures. ∎

Proof: [Theorem 2] Recall that an ordering is feasible if each of the substructures is traversable and every intermediate structure is valid. Consider a structure decomposed into d substructures. Assume $S_1, S_2, \cdots, S_k$ to be a feasible sequence while $S_1, S_2, \cdots, S_{k+1}$ is not. This implies that substructure S_{k+1} is either not traversable or $\bigcup_{i=1,\cdots,k+1} S_i$ is not a valid structure. In other words, S_{k+1} must not be the $(d-k)^{th}$ substructure to be removed. However, **Algorithm 2** checks for traversability to determine a removable substructure. Further, Theorem 1 states that only valid substructures (O_d) are generated using **Algorithm 1**. Since **Algorithm 3** iterates through O_d to compute the final sequence of valid substructures (O_f) as the reverse of the order in which substructures are removed, S_{k+1} is indeed the $(d-k)^{th}$ substructure to be removed. Hence, by contradiction, $S_1, \cdots, S_{k+1}$, as determined by **Algorithm 3**, is a feasible sequence of substructures. ∎

Proof: [Lemma 3] The MILP approach will find a solution for a valid structure at the minimum makespan for which the MILP model becomes feasible. It follows from Theorems 1 and 2 that **Algorithms 1** and **3** return a feasible sequence of substructures given a valid structure. Since a feasible sequence indicates valid substructures, MILP approach will return a solution for every substructure. ∎

Thus from Theorem 1, Theorem 2, and Lemma 3, it is proved that our approach is complete.

Proof: [Theorem 3] For N agents and d substructures, let T_i^N be the minimum makespan required to build substructure S_i when substructures S_1 to S_{i-1} are already built. Let T_{whole}^N be the minimum makespan required to build the entire structure starting from an empty world. The shadow region from Definition 2 encompasses all blocks that can act as scaffolding for a given tower. **Algorithm 1** uses this notion of shadow region to restrict all supporting blocks to either the same substructure or one of the previous substructures. As a result, constructing a substructure does not take any additional time other than what it would take to construct that part in the entire structure. We thus have the following relation for any substructure S_i generated via **Algorithm 1**:

$$T_i^N \leq T_{whole}^N \ \forall \ i \ \in \{1, 2, \cdots, d\} \tag{1}$$

Further, for the sequential construction of d substructures, the total makespan is upper bounded as:

$$\sum_{i=1}^{d} T_i^N \leq d * T_{whole}^N \tag{2}$$

∎

References

[1] E. Lam, P. J. Stuckey, S. Koenig, and T. K. S. Kumar, "Exact approaches to the multi-agent collective construction problem," in *Principles and Practice of Constraint Programming*, H. Simonis, Ed. Cham: Springer International Publishing, 2020, pp. 743–758.

[2] T. Kumar, S. Jung, and S. Koenig, "A tree-based algorithm for construction robots," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 2014, pp. 481–489, 05 2014.

[3] A. Grushin and J. A. Reggia, "Automated design of distributed control rules for the self-assembly of prespecified artificial structures," *Robotics and Autonomous Systems*, vol. 56, no. 4, pp. 334–359, 2008.

[4] A. Panangadan and M. G. Dyer, "Construction in a simulated environment using temporal goal sequencing and reinforcement learning," *Adaptive Behavior*, vol. 17, no. 1, pp. 81–104, 2009.

[5] G. Sartoretti, Y. Wu, W. Paivine, T. K. S. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *International Symposium on Distributed Autonomous Robotic Systems*, 2018.

[6] S. R. B. dos Santos, S. N. Givigi, and C. L. Nascimento, "Autonomous construction of structures in a dynamic environment using reinforcement learning," in *2013 IEEE International Systems Conference (SysCon)*, 2013, pp. 452–459.

[7] M. Gharbia, A. Chang-Richards, Y. Lu, R. Zhong, and H. Li, "Robotic technologies for on-site building construction: A systematic review," *Journal of Building Engineering*, vol. 32, p. 101584, 08 2020.

[8] C. Ye, N. Chen, L. Chen, and C. Jiang, "A variable-scale modular 3d printing robot of building interior wall," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2018, pp. 1818–1822.

[9] K. Jung, B. Chu, and D. Hong, "Robot-based construction automation: An application to steel beam assembly (part ii)," *Automation in Construction*, vol. 32, p. 62–79, 07 2013.

[10] Y. Wu, H. H. Cheng, A. Fingrut, K. Crolla, Y. Yam, and D. Lau, "Cubrick cable-driven robot for automated construction of complex brick structures: From simulation to hardware realisation," in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2018, pp. 166–173.

[11] J. Werfel and R. Nagpal, "Extended stigmergy in collective construction," *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 20–28, 2006.

[12] M. W. Cohen and V. N. Coelho, "Open-pit mining operational planning using multi agent systems," *Procedia Computer Science*, vol. 192, pp. 1677–1686, 2021, knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 25th International Conference KES2021.

[13] B. Khoshnevis, "Automated construction by contour crafting—related robotics and information technologies," *Automation in Construction*, vol. 13, no. 1, pp. 5–19, 2004, the best of ISARC 2002.

[14] J. Werfel and R. Nagpal, "Three-dimensional construction with mobile robots and modular blocks," *I. J. Robotic Res.*, vol. 27, pp. 463–479, 03 2008.

[15] H. Durrant-Whyte, N. Roy, and P. Abbeel, *Construction of Cubic Structures with Quadrotor Teams*, 2012, pp. 177–184.

[16] M. S. d. Silva, V. Thangavelu, and N. Napp, "Autonomous multi-material construction with a heterogeneous robot team," 09 2018.

[17] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal, "Distributed construction by mobile robots with enhanced building blocks," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 2787–2794.

[18] Q. Lindsey, D. Mellinger, and V. Kondepogu, "Construction with quadrotor teams," *Autonomous Robots*, vol. 33, 10 2012.

[19] H. Durrant-Whyte, N. Roy, and P. Abbeel, *TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction*, 2012, pp. 257–264.

[20] M. A. Bragin, P. B. Luh, B. Yan, and X. Sun, "A scalable solution methodology for mixed-integer linear programming problems arising in automation," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 531–541, 2019.

[21] F. Lin, S. Leyffer, and T. Munson, "A two-level approach to large mixed-integer programs with application to cogeneration in energy-efficient buildings," *Computational Optimization and Applications*, vol. 65, 09 2016.

[22] A. Jain, T. Thormählen, T. Ritschel, and H.-P. Seidel, "Exploring shape variations by 3d-model decomposition and part-based recombination," *Computer Graphics Forum*, vol. 31, pp. 631–640, 05 2012.

[23] L. Homem de Mello and A. Sanderson, "A correct and complete algorithm for the generation of mechanical assembly sequences," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240, 1991.

[24] S. Chakrabarty and J. Wolter, "A structure-oriented approach to assembly sequence planning," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 14–29, 1997.