Hierarchical Planner for the cuboid-Multi-Agent Collective Construction Problem

Shambhavi Singh, Geordan Gutow, Bhaskar Vundurthy, and Howie Choset

Carnegie Mellon University, Pittsburgh, PA 15217, USA,
{shambhas,ggutow,pvundurt,choset} @andrew.cmu.edu

Abstract. Recent work in planning for cooperative mobile robots to assemble a structure much larger than themselves out of only *cubic* ($1 \times 1 \times 1$) blocks has been termed "the multi-agent collective construction (MACC) problem." In this work, we create a new planner that solves the MACC problem with *cuboid* ($n \times 1 \times 1$) blocks, which we call the cuboid-MACC problem. In doing so, we can now build structures with hollow features. The cuboid blocks introduce new challenges beyond those present in MACC: blocks are no longer rotationally symmetric, and a carried block can collide with other blocks or robots; both features restrict the paths robots can take while carrying blocks. To address cuboid-MACC, we present a novel hierarchical planning approach that handles the maneuverability constraints imposed by cuboid blocks. First, we use A^* to determine a sequence of abstract actions (block placements and removals) to build the structure without planning specific paths for the robots. We nevertheless ensure that such paths will exist through the concept of a reachable abstract action. Next, we identify dependencies between the chosen abstract actions by checking for conflicts between single-agent paths to execute the actions and store the dependencies as an acyclic Action Dependency Graph. Finally, we iterate on the specific paths taken by robots using a low-level multi-agent pathfinding algorithm by suitable modifications to conflict-based search (CBS). We demonstrate our planner on 200 randomly generated structures built with three types of cuboidal blocks.

Keywords: multi-agent systems, graph search, collective construction

1 Introduction

While humans continue to play a crucial role in many construction projects, recent years have seen an increased interest in studying autonomous robotic systems for building structures in remote and hostile situations on Earth, in space, or on other planets [1–7]. When structures are much larger than the robots that will build them, planning algorithms must ensure not just the completion of the structure, but that at every phase of construction it is possible for the robots to traverse the structure to accomplish their tasks. Inspired by termites that can build mounds that are significantly larger than themselves, prior works study a collective construction problem where multiple robots are tasked to manipulate

building "blocks" to assemble a much larger structure. This is referred to as the Multi-Agent Collective Construction (MACC) problem [8].

The MACC problem considers a 3D grid world where the robots and blocks are all the size of a $1 \times 1 \times 1$ cube. Prior works [9–13] impose a gravity constraint on the grid world and equip the robots to pick up, transport, and place the blocks. However, they are allowed to traverse at most one block height at a time. Thus, the robots may construct temporary staircase-like scaffolding to reach higher altitudes. Prior works rely on optimization [11] for solution optimality or heuristics [9, 14, 15], structural decomposition [12] and Reinforcement Learning [10, 16] techniques to speed up computation time. However, the optimization based approaches struggle to handle large structures in a reasonable time frame, while the fast heuristic and learning approaches produce very poor quality solutions [12]. In this work, we propose a hierarchical search-based approach to balance computational cost and solution quality. We first use A* to find a sequence of block placements and removals that a single agent could use to build the structure. Some of these block placements and removals can happen simultaneously but others are dependent; we conservatively identify these dependencies by detecting conflicts between single agent paths to execute the block actions, and encode them in an acyclic Action Dependency Graph. At the low-level, we iteratively allocate assembly tasks whose dependencies are complete to agents such that the final completion time is minimized by combining the Hopcroft-Karp algorithm [17] with a bisection search. Finally, we compute the specific paths for the assigned agents using a modified conflict-based search algorithm [18].

Another contribution of this work is to generalize the building blocks from uniform cubes to heterogeneous cuboids (elongated cubes). Such elongated blocks allow structures with canopies and hollow spaces such as cantilevers, roofs, and bridges, as shown in Fig. 1. While the larger footprint of the new blocks is beneficial to constructing more generic structures, it presents new challenges in the form of an additional planning dimension (orientation), as well as more complex collision avoidance constraints on the robots as they move through the world. To the best of our knowledge, there does not exist any prior work for this modified problem formulation, which we refer to as the cuboid Multi-Agent Collective Construction (cuboid-MACC) problem.

The rest of the paper is organized as follows. Section 2 presents a literature review of hierarchical planning methods, then discusses existing approaches to the traditional MACC problem and to pathfinding problems relevant to cuboid-MACC. Section 3 formally defines the cuboid-MACC problem. We describe our hierarchical planning solver for cuboid-MACC in section 4. Performance analysis on 200 random structures containing cuboids of length 1, 3, and 5 is presented in section 5. Section 6 concludes the paper. We release source code and the test set at https://github.com/biorobotics/cuboidMACC-DARS24. Animations of the plans obtained are available at https://drive.google.com/drive/folders/1sIe869tkjfeljM5nNOdU_zbTR_sPCe0O?usp=sharing.

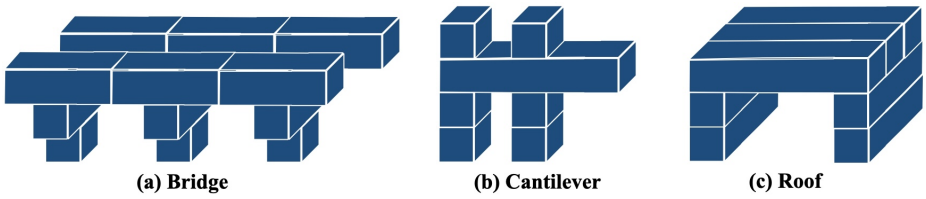

Fig. 1: Example 3D structures with hollow spaces can be used to form bridges, cantilevers, and roofs.

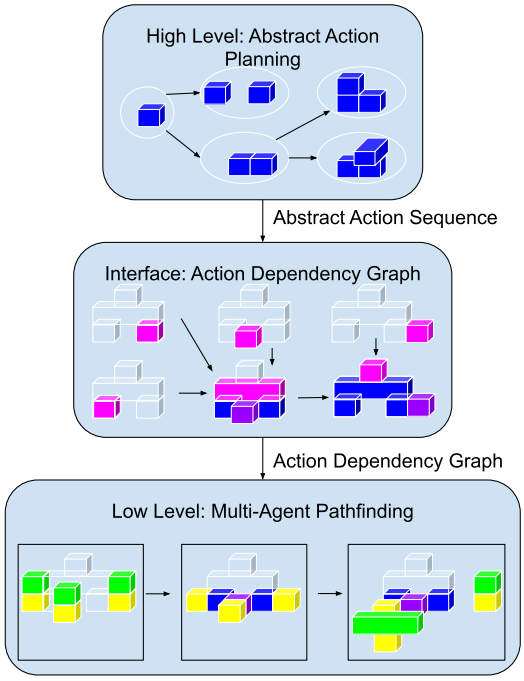

Fig. 2: The high-level planner finds a sequence of block placements and removals that builds the target structure. The interface identifies dependencies between these actions. The low level finds multi-agent paths to accomplish the actions from the high level. Blue - Structure blocks, Purple - Scaffolding, Pink - Block Placement Action, Yellow - Robot, Green - Block Carried by Robot.

2 Related Works

In this section, we briefly review the existing literature on hierarchical planning methods, the Multi-Agent Collective Construction problem, multi-agent pathfinding, and path planning for polygonal robots.

Hierarchical Planning methods leverage problem structure and domain knowledge to design hierarchies of planners that each solve an easier problem.

Plans can be generated using hand-designed abstractions, or high-level actions, and subsequently refined to find appropriate low-level actions. Recent work in multi-arm assembly [19] generates roadmaps, then solves a relaxed assembly problem on these roadmaps, before finding arm trajectories. Another work on self-assembling structures [14] calculates a partial order plan, then derives motion plans using local controllers. Additionally, methods propose ways to optimize refining abstract plans [20] to further improve computational expense. In this work, we employ a hierarchical planner to solve a modified MACC problem.

Multi-agent Collective Construction: The TERMES [8] project develops the problem of collective construction of three-dimensional structures using small mobile robots. The robots can pick-up and deliver blocks that compose the structure, and climb up and down stairs of blocks to find traversable paths on the structure. [9] develops a specialized heuristic planner to solve this problem. They employ a Minimum Spanning Tree-based search method, which provides instantaneous solutions even for large structures. Another method [10] uses a Distributed Reinforcement Learning for the collaboration of multiple agents without explicit interactions. Although fast to compute, these methods produce solutions requiring very large numbers of robot actions. An exact method uses Mixed Integer Linear Programming (MILP) formulation and finds a plan optimal in number of timesteps (or makespan) for all robots to enter, build the structure, and exit the environment [11]. Recent work suggests a decomposition of the goal structure into multiple smaller structures to reduce computation time [12]. Decomposition is effective in reducing the computational effort for sparse and scattered structures compared to the exact method, but less so for taller or densely built structures. Another work [13] uses a hierarchical planning framework for the MACC problem that plans for robot construction tasks and multi-robot paths hierarchically. Both solutions provide significant improvement in computational effort but have limited scope for generalization. The MACC problem only includes cubic blocks, cannot have structures with hollow spaces inside, and thus struggles to produce structures with resemblance to real-life building elements.

In our work, we modify the MACC problem to a heterogeneous version where robots may construct structures with cuboid shaped blocks. To find a solution, we must plan for each robot's path for transportation and assembly of the structure. Robot path planning is much harder than in regular MACC because the orientation of a carried block affects what movements are collision free. We build off of existing hierarchical methods to MACC [13], as well as from other domains [14, 19, 20] to develop a hierarchical planning pipeline (Fig. 2) for our modified problem. After planning an assembly sequence (Abstract Action Planning) that respects structure stability constraints, we identify dependencies between assembly actions (Action Dependency Graph), then solve a multi-agent pathfinding problem for robots carrying heterogeneous blocks.

Multi-agent path-finding (MAPF) problems aim to find collision-free paths for a group of agents to reach their target locations while optimizing an objective function. The work in [21] formalizes a classical MAPF problem

as a tuple (G, s, t) for k agents, where $G = (V, E)$ is an undirected graph, and s:$[1, ..., k] \rightarrow V$ and t:$[1, ..., k] \rightarrow V$ map agents to source and target vertices respectively. Time is discrete and each agent can take a single action at every time step. We implement and modify [18] to find shortest paths for multiple agents transporting and delivering blocks. A variant of MAPF, the Multi-agent pick-up and delivery (MAPD) problem [22], extends MAPF solutions to include task planning and task allocation to multiple agents. Although there has been significant development of solutions for large-scale MAPF and MAPD problem, path planning for robots with rigid polygonal or polyhedral shapes has typically not been addressed in multi-agent problems.

Path planning for polygonal robots often employs complex collision-checking computations on robot configuration spaces [23–25]. In a multi-robot scenario, the cost for collision-checking calculations can grow as the square of the number of robots. Using these algorithms thus leads to an explosive number of calculations if precise collision avoidance is required. Our work mitigates this cost by using cheap, conservative collision envelopes for the robots.

3 Cuboid Multi-Agent Collective Construction Problem

We consider a problem where cubic (1x1x1) mobile robots construct a specified structure composed of cuboidal blocks (Fig. 1). A block's geometry is specified by its length; in this work, all blocks are 1 unit tall and 1 unit deep. A block or robot's location $L = (x, y, z, rotation)$ is specified by the coordinates of its center and by its rotation about the vertical axis, which can be 0 degrees or 90 degrees. Because all items are mirror symmetric about their center, other orientations (270, 180, etc) are not considered. The ground level is $z = 0$. A special depot is also available outside of the grid world, where an infinite supply of blocks can be accessed. Robots can carry blocks; when carried by a robot, the block's center is 1 unit above the robot, and its rotation matches that of the robot.

Definition 1. *A* ***world state*** $\{(B_1, L_1)...(B_N, L_N)\}$ *is a set of pairs of blocks* B_i *and their locations* L_i *in the world. For every cell in the grid world, a world state uniquely determines if that cell is filled or not.*

Definition 2. *A block* B *has* ***support*** *at a location* $L = (x, y, z, rotation)$ *in a world state* W *if* $z = 0$*, or if* $(x, y, z - 1)$ *is filled, or if cells that will be filled by* B *at* L *have two filled cells 1 unit below which are equidistant from* L*.*

Definition 3. *A* ***robot state*** $X = (t, L, B)$ *specifies a timestep* t*, robot location* L*, and the carried block* B*.*

Definition 4. *For a block location* (x_b, y_b, z_b, r_b)*, the* ***access locations*** *are* $\{(x_b - 1, y_b, z_b, r_b), (x_b + 1, y_b, z_b, r_b)\}$ *if* r_b *is 0, and* $\{(x_b, y_b - 1, z_b, r_b), (x_b, y_b + 1, z_b, r_b)\}$ *if* r_b *is 90.*

In this work, we distinguish between abstract actions and primitive actions, similar to [13]. There are two types of abstract action: ADD, which inserts a block into the world state at a location with support, and REMOVE, which deletes a block from a specific location in the world state.

There are four types of primitive actions, each requiring one time step. We visualize the actions available to a robot carrying a block of size (3x1x1) in Fig 3. The simplest is to WAIT at the depot, available only if the robot is at the depot. Robots may MOVE, stepping up to 1 unit forwards or backwards or rotating 90 degrees in place. As part of a MOVE, the robot can change its height by up to one unit as if climbing or descending a staircase. If at the depot the robot may instead MOVE to ground level in any cell at the edge of the world, or vice-versa. The third action type is to PLACE a carried block. Blocks can be placed in front of or behind the robot, at the same height as the robot. The resulting block location has the same height and rotation as the robot, but x or y will differ by exactly 1 unit. A block can only be placed at location that has support. The fourth action type is to PICKUP a block from the world. PICKUP is available if the robot is at an access location of the block. Note that in the traditional MACC problem, size 1 blocks can be picked up or placed from any direction, which is not permitted in this formulation. All actions cost 1 unit, except for WAIT which has 0 cost. Since robots can climb/descend at most one unit at a time, they may place length 1 scaffolding blocks not part of the desired structure to access higher levels of the workspace. Such scaffolding blocks must be removed before the structure is considered complete.

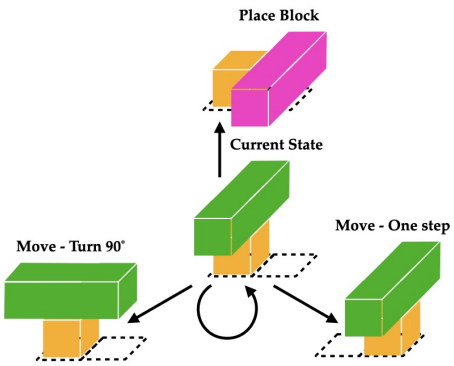

Fig. 3: An illustration of the primitive actions where a robot (Yellow) that is carrying an elongated block (Green) can: WAIT, MOVE one step forwards or backward, MOVE to rotate by 90°, and PLACE block (Pink) forwards or backwards. Carried blocks extend to the left and right of the robot.

Definition 5. *A primitive action $A = (K, B, L)$ specifies a kind of action K (WAIT, MOVE, PICKUP, PLACE), the block B to do the action with, and the location L to do the action to.*

Definition 6. *A robot state X and primitive action A are **compatible** if there exists some world state in which a robot at X could execute A. For a WAIT, X must be at the depot. For a MOVE, the action's location must be connected to X. For a PICKUP or PLACE, X must be at an access location. For PICKUP, X must not be carrying a block. For PLACE, X must be carrying the block.*

Definition 7. *A primitive edge (X, A) consists of a start robot state X and a compatible primitive action A. This determines the exit state, which is the robot state after completing A.*

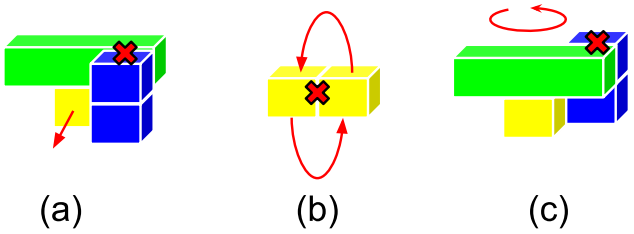

Fig. 4: a) Robots may not move into a state where they or a carried block intersect other blocks or robots b) robots cannot exchange locations c) robots cannot rotate if doing so would intersect other blocks or robots.

An additional challenge presented by the cuboid-MACC formulation is that the robots and blocks could potentially collide with each other as shown in figure 4. We enforce non-collision conservatively. At both the start and exit states of an edge, the robot and its carried block cannot intersect any other robot or block (a). Additionally, robots or their carried blocks cannot exchange occupied cells during a single timestep (b), and a robot may only rotate if every cell in the axis-aligned square that circumscribes the start and end cells of the carried block is empty (c). The depot location is always collision-free.

Definition 8. *A robot* ***can stand*** *at* x, y, z *if* $z = 0$ *or* $x, y, z - 1$ *is filled*

Definition 9. *A robot state is* ***legal*** *in a world state if it can stand there and neither the robot cell nor the cells occupied by its carried block are filled.*

Definition 10. *A primitive action* (K, B, L) *is* ***feasible*** *in a world state if a robot state exists from which it makes sense. MOVE and WAIT are always feasible. PICKUP is feasible if* B *is present at* L*. PLACE is feasible if the cells filled by* B *at* L *are all empty and* B *has support at* L*.*

Definition 11. *A primitive edge is* ***executable*** *in a world state if its primitive action can be executed from its start state. PICKUP, PLACE, and WAIT are always executable. MOVE is executable if the cells swept out by the robot and its block during the movement are all empty.*

Definition 12. *A primitive edge is* ***valid*** *in a world state if its start and exit states are legal and its action is both feasible and executable. A sequence of primitive edges is* ***valid*** *if every edge is valid.*

An abstract action thus corresponds to a set of sequences of edges, any of which would accomplish the ADD or REMOVE. The primitive sequences for ADD are all paths a robot could take from the depot to place the desired block and return; for REMOVE, the sequences are all paths from the depot to pick-up the desired block and return.

4 Hierarchical Planner for cuboid-MACC

We take a hierarchical approach to cuboid-MACC, as shown in Fig. 2. At the high level, we search for the shortest sequence of abstract ADD and REMOVE actions that completes the target structure and has at least one valid concatenation of corresponding primitive edge sequences. Such a concatenation is a plan for a single robot to construct the entire structure. Next, we construct an action dependency graph (ADG), a directed acyclic graph whose nodes are the abstract actions in the sequence. A valid primitive sequence to complete an abstract action is guaranteed to exist if all abstract actions corresponding to its ancestors in the ADG are already completed. Finally, we compute multi-agent primitive sequences to complete all the abstract actions by repeatedly selecting all abstract actions corresponding to nodes with no parents in the ADG, solving a multi-agent path finding instance, and removing those actions' nodes from the ADG.

4.1 High-Level: Assembly Action Planning

We formulate high-level planning as a graph search, with each node symbolizing a world state and each edge symbolizing an abstract action. Utilizing A* search with edge costs of 1, we find the shortest sequence of abstract actions constructing the target structure from an initially empty world.

Definition 13. *A location L is **reachable** in a particular world state with a block B if a valid primitive edge sequence exists from L with B to the depot.*

Definition 14. *ADD or REMOVE block B at location L is **executable** if at least one access location is reachable both before and after executing the abstract action. If ADD, the access location must be reachable in the original world state while carrying B; if REMOVE, the access location must be reachable in the resulting world state while carrying B.*

At each high-level node, the available edges correspond to the executable abstract actions at that world state (not to primitive edges). The heuristic function estimates the number of remaining block ADD and REMOVE actions to build the goal structure. If n_g unplaced goal structure blocks with n_p placed scaffolding blocks needs n_s additional scaffolding blocks, the exact cost-to-go is:

$$h^* = (n_g + n_s) + (n_s + n_p)$$

n_g and n_p are easy to compute, but the number of ways to place blocks to construct a scaffolding staircase is extremely large; exactly calculating n_s is challenging [13]. Instead we use an estimate of n_s. Let N be the number of unique heights in the structure where goal block ADDs are required but not executable. We estimate $n_s = N(N+1)/2$. This stems from the pattern that building scaffolding typically requires constructing many towers of decreasing heights (to reach height H, build H-1, H-2, ... 1 height towers). Detailed analysis of this heuristic is beyond the scope of this work, but it is not admissible.

Additionally, the performance of high-level search is dependent on efficient computation of which abstract actions are executable. The naive approach of searching from the boundary to every access location for every abstract action every time a search node is expanded is expensive. Instead, for every world state encountered during the high-level search, we maintain for each block type a set of locations we have proven reachable carrying that block type, and another set of locations we have proven are unreachable. Initially, no locations are unreachable and only the boundary is reachable. When an action location is queried and it is not in either the reachable or unreachable sets, we conduct breadth first search from the location until a known reachable or unreachable location is found. If the reachable set is found, all locations encountered during the search are marked reachable. Conversely, if the unreachable set is found or search terminates without finding anything, all locations encountered are marked unreachable.

4.2 Interface: Action Dependency Graph Construction

High-level planning provides a sequence S of abstract actions for which there exists a valid sequence of primitive edges to construct the target structure by executing the abstract actions in order. However, many of the abstract actions do not need to occur consecutively. For the ith abstract action in S, we wish to identify a subsequence D of $S[1 : i-1]$ of parent abstract actions such that $S[i]$ is executable for the world state produced by executing D. We will then store the relationship between $S[i]$ and the elements of D in the directed Action Dependency Graph (ADG).

For each abstract action $S[i]$, we use A* to find the shortest primitive edge sequence $E[i]$ that accomplishes $S[i]$ and returns to the depot, that is valid in the world state produced by $S[1 : i-1]$. Then, for every pair of abstract actions $S[i]$ and $S[j]$, $i < j$, we detect conflicts between the primitive paths $E[i]$ and $E[j]$ (see section 4.3). If a conflict is found, we add an edge from $S[i]$ to $S[j]$.

4.3 Low-Level: Heterogeneous Task Allocation and Multi-Agent Path Finding

The interface stage provides a directed acyclic graph whose nodes correspond to abstract actions and whose edges indicate an abstract action must be completed before its child abstract action. At the low-level, we remove abstract actions from this graph, allocate them to robots, and conduct multi-agent path finding to obtain primitive edge sequences.

The low-level proceeds in rounds. For each robot, we maintain a record of the earliest robot state at which it is available for a new task. Initially, all robots are available at t=0 and located at the depot.

Within a round, we remove all nodes from the action dependency graph that have no parents. These are the actions that will be completed during this round, and one robot will be involved in the multi-agent pathfinding for each action. For every robot-abstract action pair $\langle R, A \rangle$ we compute the horizontal Manhattan distance between the location of R and the nearest access location for A, plus

the time at which the R becomes available. This is an underestimate of the time R would complete A. We then solve a min-max job scheduling problem to assign robots to abstract actions, such that the predicted time at which the last action is executed (the "makespan") is minimized. We conduct a bisection search on the makespan; for a given candidate makespan, we find an assignment that achieves this makespan (if it exists) via the Hopcroft-Karp algorithm [17].

Once all abstract actions for the round are assigned ($\langle R_i, A_i \rangle$), we conduct conflict based search (CBS) [18] between the robots for this round to find primitive edge sequences that complete the assigned action, then return to the depot. The nodes of the CBS internal single-agent searches are robot states. The edges are primitive edges as described in Section 3.

The conflict-based search algorithm used here differs from the standard in four ways. First, in every round except the first we have universal constraints that always apply to all robots, preventing the robots from interfering with the primitive sequences from previous rounds. This is implemented via a variant of the traditional Edge Constraint that we call a Cell Set constraint.

Definition 15. *A **Cell Set** constraint* $(t, \{(x_1, y_1, z_1)...(x_N, y_N, z_N)\})$ *specifies for time* t *a set of cells that neither a robot nor a carried block may intersect.*

Second, after the first round R_i is subject to edge constraints forbidding every edge that completes A_i before one of its parents. We are guaranteed to know this time since all parents of the A_i were processed in a previous round. Third, the world state is mutable by the other robots. Thus, the validity of primitive edges depends on the time. At the root node of the CBS tree, we must plan for all robots in the round without knowing when the other robots will change the world. To handle this, we introduce the notions of a "maybe legal" state, a "maybe feasible" and "maybe executable" primitive action, and a "maybe valid" edge, all of which are influenced by the "pending actions."

Definition 16. *A **pending action** at time* t *is an abstract action that may or may not be completed by time* t*.*

Definition 17. *A robot state (t,L,B) is **maybe legal** in a world state if 1. it can stand at L or the cell below L will be filled by a pending action and 2. for every cell occupied by the robot or its carried block, that cell is either empty or a pending action will empty it.*

Definition 18. *A primitive action is **maybe feasible** if it is feasible. A PICKUP action is maybe feasible if a pending action will place the block to be removed. A PLACE action is maybe feasible if the block would collide with an existing block that a pending action will remove, or pending actions will place enough blocks that the block will have support.*

Definition 19. *A primitive action is **maybe executable** if it is executable. A MOVE action is maybe executable if for each cell swept out by the robot and its block during the motion, the cell is empty or a pending action will empty it.*

Definition 20. *A primitive edge is* ***maybe valid*** *if both its start and exit states are maybe legal and its action is both maybe feasible and maybe executable. A sequence of primitive edges is* ***maybe valid*** *if every edge is maybe valid.*

During the root internal CBS single agent searches, edges are permitted if they are maybe valid with pending actions consisting of the abstract actions of the other agents for this round. This creates the possibility of a new type of conflict: a robot follows an edge that is valid only because of some combination of pending actions, but at the time the edge is taken that combination has not occurred. We define three new conflict types, which correspond to states that were not legal, actions that were not feasible, and actions that were not executable. Note that for each conflict, we describe the constraints added to each branch of the conflict tree.

Definition 21. *A* ***Robot-World*** *conflict occurs when the exit state $X = (t, L, B)$ of agent i's edge is not legal because of a PICKUP or PLACE action P taken by agent j at time τ. If $t \leq \tau$, the first branch bans agent i from occupying L until $\tau + 1$. The other branch requires agent j to complete P before t. If $t > \tau$, the first branch bans agent i from occupying L after $\tau - 1$. The other branch requires agent j to complete P after t.*

Definition 22. *An* ***Action-World*** *conflict occurs when the action A taken by agent i at time t is not feasible because of a PICKUP or PLACE action P taken by agent j at time τ. If $t + 1 < \tau$, the first branch bans agent i from any edge with action A at or before τ. The other branch requires agent j to complete P before $t + 1$. If $t + 1 \geq \tau$, the first branch bans agent i from any edge with action A after $\tau - 1$. The other branch requires agent j complete P after $t + 1$.*

Definition 23. *A* ***Edge-World*** *conflict occurs when the action A of agent i's edge $E = (X, A)$ is not executable at time t because of a PICKUP or PLACE action P taken by agent j at time τ. If $t + 1 < \tau$, the first branch bans agent i from taking A from the start state of E at any time $\leq \tau$. On the other branch, require agent j to complete P before $t + 1$. If $t + 1 \geq \tau$, the first branch bans agent i from taking A from X at any time $\geq \tau$. The other branch requires agent j complete P after $t + 1$.*

When selecting a conflict to resolve, prioritize Action-World conflicts, then Robot-World conflicts, then Edge-World conflicts before vertex or edge conflicts.

The last variation from standard CBS is in the objective function. CBS can minimize the sum of the edge costs of the single agent plans (the "sum-of-costs" formulation) or the maximum single agent plan cost (the "makespan" formulation) [18]. In the multi-agent construction literature (e.g. [11,26]) it is common to minimize lexicographically (makespan, sum-of-costs). The difficulty in using this objective for CBS is that, when a single agent plans, it should seek smaller edge costs only on paths whose makespan is not the largest amongst all agents. Below the root node, only one agent plans and so the makespan limit is known. Only the root CBS node at which all agents must plan is a challenge. At the

root, we first plan with each agent seeking to minimize its individual makespan. Then we replan for each agent, using the makespan limit obtained from the first plan. This second set of plans is checked for conflicts and CBS proceeds as usual.

5 Results

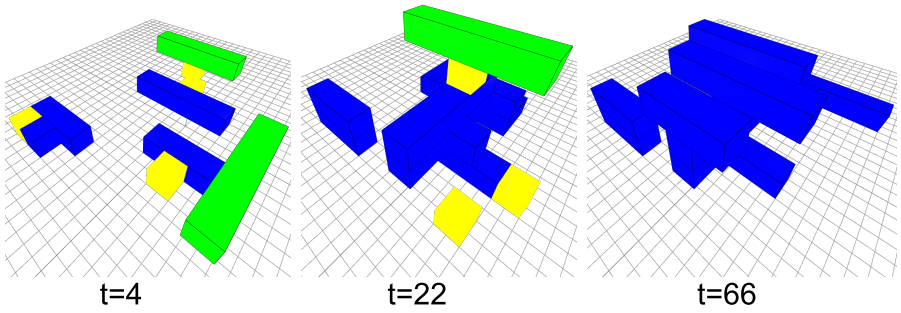

Fig. 5: Images showing the progress of construction of one of the random structures. Robots are in yellow, carried blocks in green, structure blocks in blue.

We demonstrate the capability of our approach on a test suite of 200 randomly generated designs containing 15 blocks stacked up to 4 blocks high in a $10 \times 10 \times 5$ world. These structures contain $1 \times 1 \times 1$, $3 \times 1 \times 1$, and $5 \times 1 \times 1$ blocks that are introduced for the first time in the current cuboid-MACC formulation, and so no prior MACC approaches can build them. Three timesteps from the multi-agent plan obtained for one of these structures are shown in Fig. 5.

We release source code and the test set at https://github.com/biorobotics/cuboidMACC-DARS24. Task allocation is handled by the NetworkX 3.2.1 implementation of Hopcroft-Karp. Our algorithms are implemented in Python 3.12, except for the incremental reachability computations which are implemented in C++ and wrapped by pybind11. We do not leverage multi-threading. Timings are reported for an Intel i7-11800H with 16GB of RAM. Our CBS implementation makes use of bypassing [27]. We allow up to 1000 seconds total walltime for each structure.

Of the 200 test structures, the high level planner is unable to find an abstract action sequence for 54 of them. In each such case, rather than reaching the time limit, the available RAM was exhausted during the A* search. The 1000 second limit is reached during the low-level in a further 20 structures. Summary walltime statistics for the 126 complete successes are reported in Table 1. We observe that the maximum runtime for high and low-level search is extremely high in comparison to the median. This suggests that structures of similar size (all structures contain 15 blocks) can vary widely in difficulty. In 102 cases the

low-level took the longest, while in the remaining 24 the high level was the slowest.

Table 1: Statistics on walltime spent in each stage in the random test set.

	High-Level (s)	Interface (s)	Low Level (s)
Min	0.451	0.172	1.42
Median	4.03	0.596	22.0
Mean	26.8	0.762	56.0
Max	367.8	3.51	998.6

6 Conclusion

In this work, we describe a generalization of the MACC problem to structures composed of heterogeneous cuboid shaped blocks and demonstrate a hierarchical planning approach to this new problem. We also provide a detailed development of the interaction between the mutability of the world and conflict-based multi-agent pathfinding. The definition of a "maybe valid" edge seems to result from conjunctions and disjunctions of preconditions for the edge to be valid; in our future work, we aim to make this connection precise. A more abstract treatment of world mutability and multi-agent conflicts, in terms of action pre and post conditions, may permit extending this approach to more complex settings. We will also investigate further performance improvements; in particular, the hierarchical breakdown of the problem precludes optimality guarantees, so incorporating suboptimal methods throughout the pipeline could provide significant speed ups without sacrificing any theoretical properties. We made use of an inadmissible heuristic for the high level A* search, but the task allocation and low-level planning both attempt to solve their subproblems to optimality. The propensity of the high-level search to generate extremely large numbers of search nodes and exhaust the available RAM might suggest adopting limited memory search techniques, but the authors suggest that problem-specific heuristics would also be of significant benefit here. Anecdotally, it is not difficult for a human to derive abstract action sequences to build the structures studied in this paper; capturing this ability in heuristic guidance for the search would be fruitful. Finally, the MAPF problems dominate the runtime for most structures; more sophisticated algorithms for multi-agent pathfinding would be beneficial here, such as the use of incremental single agent search in [28].

References

1. In-Won Park, Damiana Catanoso, Olivia Formoso, Christine Gregg, Megan Ochalek, Taiwo Olatunde, Frank Sebastianelli, Pascal Spino, Elizabeth Taylor, Greenfield Trinh, and Kenneth Cheung. 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)*, pages 10736–10741, 2023.

2. William R Doggett, John Dorsey, John Teter, David Paddock, Thomas Jones, Erik E Komendera, Lynn Bowman, Chuck Taylor, and Martin Mikulas. Persistent assets in zero-g and on planetary surfaces: Enabled by modular technology and robotic operations. In *2018 AIAA SPACE and Astronautics Forum and Exposition*, page 5305, 2018.
3. Harley Thronson, Bradley M Peterson, Matthew Greenhouse, Howard MacEwen, Rudranarayan Mukherjee, Ronald Polidan, Benjamin Reed, Nicholas Siegler, and Hsiao Smith. Human space flight and future major space astrophysics missions: servicing and assembly. In *UV/Optical/IR Space Telescopes and Instruments: Innovative Technologies and Concepts VIII*, volume 10398, pages 373–388. SPIE, 2017.
4. Stefan Scherzinger, Jakob Weinland, Robert Wilbrandt, Pascal Becker, Arne Roennau, and Rüdiger Dillmann. A walking space robot for on-orbit satellite servicing: The recobot. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 2231–2237, 2022.
5. Sho Nakanose and Keiko Nakamura-Messenger. Gitai usa: Providing safe and affordable means of labor in space. In *ASCEND 2023*, page 4744. 2023.
6. Steven J Schuldt, Jeneé A Jagoda, Andrew J Hoisington, and Justin D Delorit. A systematic review and analysis of the viability of 3d-printed construction in remote environments. *Automation in Construction*, 125:103642, 2021.
7. Kirstin H Petersen, Nils Napp, Robert Stuart-Smith, Daniela Rus, and Mirko Kovac. A review of collective robotic construction. *Science Robotics*, 4(28):eaau8479, 2019.
8. Kirstin Petersen, Radhika Nagpal, and Justin Werfel. Termes: An autonomous robotic system for three-dimensional collective construction. In *Robotics: Science and Systems Conference (RSS)*, 2011.
9. T.K.S. Kumar, Sangmook Jung, and Sven Koenig. A tree-based algorithm for construction robots. *Proceedings of the International Conference on Automated Planning and Scheduling*, 2014:481–489, 05 2014.
10. Guillaume Sartoretti, Yue Wu, William Paivine, T. K. Satish Kumar, Sven Koenig, and Howie Choset. Distributed reinforcement learning for multi-robot decentralized collective construction. In *International Symposium on Distributed Autonomous Robotic Systems*, 2018.
11. Edward Lam, Peter J. Stuckey, Sven Koenig, and T. K. Satish Kumar. Exact approaches to the multi-agent collective construction problem. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 743–758, Cham, 2020. Springer International Publishing.
12. Akshaya Kesarimangalam Srinivasan, Shambhavi Singh, Geordan Gutow, Howie Choset, and Bhaskar Vundurthy. Multi-agent Collective Construction using 3D Decomposition. *arXiv e-prints*, page arXiv:2309.00985, September 2023.
13. Shambhavi Singh, Geordan Gutow, Akshaya Kesarimangalam Srinivasan, Bhaskar Vundurthy, and Howie Choset. Hierarchical propositional logic planning for multi-agent collective construction. *Construction Robotics Workshop*, 2023.
14. Alexander Grushin and James A. Reggia. Automated design of distributed control rules for the self-assembly of prespecified artificial structures. *Robotics and Autonomous Systems*, 56(4):334–359, 2008.
15. Anand Panangadan and Michael G. Dyer. Construction in a simulated environment using temporal goal sequencing and reinforcement learning. *Adaptive Behavior*, 17(1):81–104, 2009.

16. Sérgio Ronaldo Barros dos Santos, Sidney N. Givigi, and Cairo Lúcio Nascimento. Autonomous construction of structures in a dynamic environment using reinforcement learning. In *2013 IEEE International Systems Conference (SysCon)*, pages 452–459, 2013.
17. John E Hopcroft and Richard M Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.
18. Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. In *Artificial Intelligence*, 2012.
19. Jingkai Chen, Jiaoyang Li, Yijiang Huang, Caelan Garrett, Dawei Sun, Chuchu Fan, Andreas Hofmann, Caitlin Mueller, Sven Koenig, and Brian C. Williams. Cooperative task and motion planning for multi-arm assembly systems. 2022.
20. Pascal Bercher, Shawn Keen, and Susanne Biundo-Stephan. Hybrid planning heuristics based on task decomposition graphs. *Proceedings of the International Symposium on Combinatorial Search*, 2014.
21. Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Barták. Multi-agent pathfinding: Definitions, variants, and benchmarks. *CoRR*, abs/1906.08291, 2019.
22. Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Adaptive Agents and Multi-Agent Systems*, 2017.
23. H. Choset, K.M. Lynch, S. Hutchinson, G.A. Kantor, and W. Burgard. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005.
24. III Kennedy, Monroe, Dinesh Thakur, M. Ani Hsieh, Subhrajit Bhattacharya, and Vijay Kumar. Optimal Paths for Polygonal Robots in SE(2). *Journal of Mechanisms and Robotics*, 10(2):021005, 02 2018.
25. Pankaj K. Agarwal, Eyal Flato, and Dan Halperin. Polygon decomposition for efficient construction of minkowski sums. *Computational Geometry*, 21(1):39–61, 2002. Sixteenth European Workshop on Computational Geometry - EUROCG-2000.
26. Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring shape variations by 3d-model decomposition and part-based recombination. *Computer Graphics Forum*, 31:631–640, 05 2012.
27. Eli Boyarski, Ariel Felner, Guni Sharon, and Roni Stern. Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 47–51, 2015.
28. Eli Boyarski, Ariel Felner, Daniel Harabor, Peter J Stuckey, Liron Cohen, Jiaoyang Li, and Sven Koenig. Iterative-deepening conflict-based search. In *International Joint Conference on Artificial Intelligence-Pacific Rim International Conference on Artificial Intelligence 2020*, pages 4084–4090. Association for the Advancement of Artificial Intelligence (AAAI), 2020.