Hierarchical Propositional Logic Planning for Multi-Agent Collective Construction

Shambhavi Singh[1], Geordan Gutow[2], Akshaya Kesarimangalam Srinivasan[2], Bhaskar Vundurthy[2], Howie Choset[2]

***Abstract*— We consider a Multi-Agent Collective Construction (MACC) scenario in which several identical cube-shaped robots, each capable of carrying one cubic block, must assemble a pre-specified structure composed of these blocks. To reach higher levels of the structure the robots must build scaffolding using extra blocks and remove it once the structure is finished. Incorrect placement of a few blocks can make it impossible to build some parts of the structure not considered until much later causing long-chains of interdependent actions. We address this challenge by introducing a hierarchical planning procedure. First, a sequence of block placements and removals that builds the structure is identified using propositional logic and graph search. This high-level plan abstracts the precise paths taken by the agents and hence has a much shorter planning horizon than the full problem. Next, block placements or removals that can occur in parallel are identified and the potential parallelism is expressed using an action dependency graph. Finally, parallel action streams are allocated to individual agents and a multi-agent path-finding planner is employed to plan collision-free low-level paths for the agents to build the structure. We test this approach on six standard test structures and show improvements over prior work. Our approach has a lower computation time compared to a time-optimal optimization-based approach as well as lower sum-of-costs compared to sub-optimal approaches.**

I. INTRODUCTION

Practical automated construction will require the coordination of multiple agents. This work aims to construct simple three-dimensional structures using several identical agents. In other words, we solve the problem of using cube-shaped robots to build a pre-specified structure, composed of similar cube-shaped blocks, in a 3D grid world. In addition to placing blocks that are part of the structure, robots are sometimes required to construct scaffolding to place or remove blocks from a higher level.

Previous approaches to this Multi-Agent Collective Construction (MACC) problem include optimization [1], heuristics [3][8], and distributed Reinforcement Learning (RL) [6][9]. While optimization-based techniques produce time-optimal solutions for the MACC problem, they do not scale well with the size of the action space. On the other hand, heuristic and RL approaches are known for generating solutions quickly, with a compromise on the number of actions needed to construct the structure.

In this work, we present a hierarchical approach to the MACC problem that returns a construction plan that is, in practice, near-optimal with respect to the total number of actions taken by all the robots. At the same time, our approach offers an order of magnitude improvement in the solution computation time compared to [1]. To this end, we first utilize a high-level planner to decide the order of placement/removal of blocks, including the temporary scaffolding required to build the structure. A low-level planner is then employed to plan agent-level coordination for block placement and inter-agent collision avoidance.

II. PROBLEM FORMULATION

A. Problem Statement

The environment is a 3D grid world containing cube-shaped robots and blocks. Robots can move up, down, left, and right in the plane, as well as climb or descend one block at a time. The robots can pick up, carry, and place a block in another location. The edges of the workspace are assumed to have an unlimited supply of blocks.

Definition 1 (Neighbor)**.** Two locations $(x_i, y_i), (x_j, y_j)$ that are 1 unit apart in Manhattan distance

Definition 2 (Scaffolding blocks)**.** Extra blocks that are not part of the pre-specified structure but are placed to facilitate robots to climb or descend from locations higher than 1 unit

The complete action space for the robots includes:

- *move* - move to a neighbor location (x_i, y_i) from (x_j, y_j)
- *deliver* - place a block at location (x_i, y_i) standing at a neighbor location (x_j, y_j). Must be carrying a block, and the height of blocks at the two locations must be the same.
- $pick-up$ - pick-up a block from location (x_i, y_i) standing at a neighbor location (x_j, y_j). Must not be carrying a block and the tower at (x_i, y_i) must be one unit taller than the tower at (x_j, y_j).
- *enter* - enter the world to a boundary location (x_i, y_i)
- *exit* - exit the world from a boundary location (x_i, y_i)

B. Performance Metrics

- Sum of costs: total number of actions taken by all robots
- Makespan: total number of time steps taken to build the structure and remove all the scaffolding blocks

This work was supported by the Air Force Office of Scientific Research

[1]Shambhavi Singh is an intern at the Robotics Institute, Carnegie Mellon University and a student at Birla Institute of Technology and Science, Pilani, India (email: shambhas@andrew.cmu.edu)

[2]Geordan Gutow, Akshaya Kesarimangalam Srinivasan, Bhaskar Vundurthy Howie Choset are affiliated with Carnegie Mellon University, USA. (email: {ggutow, akesarim, pvundurt, choset}@andrew.cmu.edu).

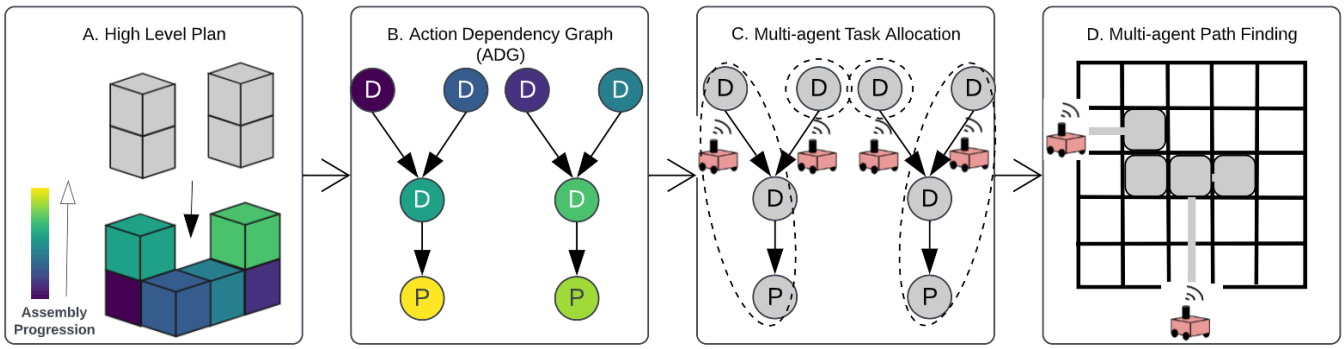


Fig. 1: Pipeline - A. The high-level plan generates a single sequence of block actions - Pick-up(P) or Deliver(D), B. An Action Dependency Graph (ADG) encodes which block actions must occur before others, C. Parallelization of actions by allocation to N robots, D. Generation of the complete plan by a multi-goal multi-agent path finding algorithm

- Computation Time: total time taken to compute the solution

III. RELATED WORKS

The MACC problem considered here traces back the TERMES project [4][2], in which teams of small robots were used to build structures much larger than themselves cooperatively. Multiple approaches have since solved versions of this problem like the one described in Section II. Recent work includes an optimization approach [1] that uses Mixed Integer Linear Programming to compute a plan that guarantees the optimal makespan. They encapsulate the environment dynamics and kinematics of robot movements as constraints in their model and aim to minimize the sum of costs once a minimum makespan is found. However, the number of integer variables in the mixed integer program is exponential in the makespan making this solution technique computationally expensive for structures with large makespan.

At the other end of the spectrum, heuristic-based approaches [3], compromise on solution quality. This work represents the 3-dimensional goal structure as a 2-dimensional matrix in the workspace. Robots traverse on a minimum spanning tree on an edge-weighted graph of the workspace. The approach attempts to minimize the number of pick-up/place actions in a plan. Another solution to the problem uses a Distributed Reinforcement Learning model [6] to train agents to build structures using the Asynchronous Advantage Actor Critic (A3C) algorithm.

The complexity of the MACC problem induces a trade-off between solution quality and resource efficiency. Although [1] finds a time-optimal solution, it suffers from high computation times for even small and sparse structures. The heuristic [3] approach finds solutions instantaneously, but experiences diminishing solution quality in both makespan and sum-of-costs as structures require coordination of multiple agents [7]. The RL-based approach [6] returns incomplete or high-cost solutions as structures become dense or complex.

Hierarchical planning has been used for long-horizon planning problems to simplify the decision-making process. The work in [11] uses Partial Order Planning to create hierarchical time-flexible plans. These plans are then refined by serially exploring them and finding complete solutions to achieve the lowest cost solutions. We follow a similar approach to solve the multi-agent collective construction problem.

IV. APPROACH

The major steps in our approach as outlined in Fig. 1 are:

(A) High-level Plan: Generate a sequence of block pick-up or deliver actions to build the structure
(B) Action Dependency Graph: Identify dependencies within the block pick-up or deliver action sequence
(C) Multi-agent Task Allocation: Task N agents to complete actions in the sequence respecting dependencies
(D) Multi-agent Path Finding: Plan collision-free paths for the set of agent-action pairs

A. High-Level Plan

In our abstract representation at the high-level, we restrict the actions to *deliver* and *pick-up* and refer to these as *block actions*, as defined here:

- *deliver* - while carrying a block, place block at (x_i, y_i) from a neighboring cell,
- $pick-up$ - while not carrying a block, remove block at (x_i, y_i) from a neighboring cell

Definition 3 (Action Location)**.** The coordinates (x_i, y_i) a block is delivered to or picked up from

Definition 4 (Parking Location)**.** The coordinates (x_j, y_j) a robot is standing at when it performs a block action

Finding the sequence of these abstract actions is modeled as a graph search problem, where a vertex is a state of the world, and edges are block actions that transform the state. We are primarily interested in minimizing the sum of costs at this stage, so in the abstract action space, we define the edge costs for placement and removal of each block using Algorithm 1. A cell (x, y) with blocks up to height h casts a shadow s at cell (x_0, y_0) if $h(x,y) > |x-x_0| + |y-y_0|$. When a shadow is cast, the size of the shadow s at (x_0, y_0) is given by $h(x,y) - |x-x_0| - |y-y_0|$ [3]. Larger shadows or shadows

Algorithm 1: Calculate Cost of Block Actions

Data: Location, Input Structure S
Result: Cost of Pick-up and Delivery actions at all locations in the workspace

for *each cell* (x_i, y_i) *in the workspace* **do**
  Initialize usefulness factor $u(x_i, y_i) \leftarrow 0$
  for *each tower of height* $h(x_k, y_k)$ *at* (x_k, y_k) *that casts a non-zero shadow* s *at* (x_i, y_i) **do**
    $s \leftarrow h(x_k, y_k) - |x_k - x_i| - |y_k - y_i|$
    $u(x_i, y_i) \leftarrow u(x_i, y_i) + s/h(x_k, y_k)$
  end
end
Cost of Pick-up $\leftarrow \frac{u(x_i,y_i) \ \forall \ i}{\max\{u(x_i,y_i) \ \forall \ i\}}$
Cost of Delivery $\leftarrow$ 1 - $\frac{u(x_i,y_i) \ \forall \ i}{\max\{u(x_i,y_i) \ \forall \ i\}}$

Algorithm 2: Allocate Tasks to N agents

Data: Action Dependency Graph G
Result: List of tasks allocated to each agent

TaskList = Topological Sort of G
Initialize $t \leftarrow 0$, and all agents as "free" at all t
while *TaskList not empty* **do**
  if *any agent is free at time* t **then**
    Pop task from *TaskList*
    For each free agent compute *TaskTime*
    Allocate task to agent A with min. *TaskTime*
    Set A not free for next *TaskTime* steps
  end
  $t \leftarrow t + 1$
end

from multiple locations increase the usefulness of the said block. Blocks with high usefulness are assigned a low cost of placement but a high cost of removal, and vice-versa.

PDDL Formulation: The block actions are represented using Planning Domain Definition Language (PDDL). These actions encapsulate the placement or removal of a block at the action location (x_i, y_i) while standing at a parking location (x_j, y_j). As a reference, the definition of *Deliver* action in PDDL is provided below. The $NeighbourOf$ condition selects a parking location that is a neighbor of the action location. $Scaffolding$ ensures scaffolding blocks exist such that a robot may climb/descend them and reach the parking location of the action. We use the heuristically determined *Cost* from Algorithm 1 to incentivize the choice of useful scaffolding blocks. Finally, we build the framework using PDDL in Julia-1.6.7 and invoke the A Star Forward Search algorithm with a precomputed Fast Forward heuristic[13] to obtain the sequence of actions.

Action Definition of Deliver in PDDL

Parameter : (x_i, y_i)

Preconditions : $\exists (x_j, y_j) : h(x_j, y_j) = h(x_i, y_i)$

$(x_j, y_j) = NeighborOf(x_i, y_i)$

$h(x_j, y_j) = 0 \mid Scaffolding(x_j, y_j)$

Effects : $h(x_i, y_i) = h(x_i, y_i) + 1$

$Cost = Cost + Cost(x_i, y_i)$

B. Action Dependency Graph

We represent action preconditions in the problem formulation by constructing an Action Dependency Graph(ADG)[10] from the sequence of actions obtained from the high-level plan. The final ADG is a Directed Acyclic Graph that may be disjoint. In order to bring a new block into the world, the robot must enter the world from a depot. In our formulation, all cells outside the boundary act as depot locations. Since our ADG only contains pick-up/delivery actions, visits to the depot are added in the next step.

C. Task Allocation

We parallelize actions in the ADG to N agents as shown in Algorithm 2 given the temporal dependencies of actions. We prioritize within free agents based on an estimate of the number of timesteps required to complete the action. This estimate ignores agent-agent interactions.

$$EstimatedTaskTime = PathLength(start, goal)$$

Here, *start* and *goal* correspond to robot's current location and the location of the action to be assigned. In cases where a robot needs to visit a depot before going to the next task, the path from the previous task to a depot and then from a depot to the next task is considered to obtain $EstimatedTaskTime$.

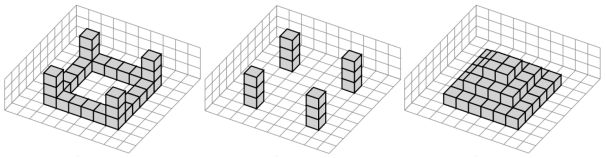
D. Path Planning

We formulate the remaining problem of finding paths for multiple agents simultaneously as a multi-goal problem. For a given pick-up/delivery location, the path found can be to any of its neighbors in the 4-connected grid space. We use the Multi-Label A* Algorithm [14] in a multi-goal problem with temporal constraints on actions: action cannot be completed until all actions it depends on are finished. This step provides each agent with a collision-free path to complete all their tasks, thus constructing the given structure.

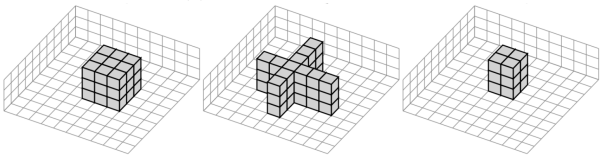
V. Results

A. Experimental Setup

We compare the performance of the hierarchical planner on a set of six test structures in Fig 2 also considered in baselines [1] [6]. The high-level plan is computed using PDDL in Julia-1.6.7. Construction of dependency graph, task allocation, and multi-agent path-finding are all implemented in Python3. All computation is done on a 12th Gen Intel® Core™ i5-1235U x 12 processor with 8GB RAM and 512 GB of disk capacity.



(a) Structures 1, 2 and 3



(b) Structures 4, 5 and 6

Fig. 2: Six test cases considered in baselines [1][6]

TABLE I: Comparison of the sum of costs with baselines

Structure	Tree based [3]	RL based [6]	Exact Approach [1]	Ours
1	1144	3040	173	**181**
2	836	1026	124	**139**
3	1590	3056	-	**336**
4	2120	3252	-	**273**
5	2180	2804	-	**395**
6	836	1276	160	**162**

B. Comparison with Baselines

We compare our approach with two existing sub-optimal approaches and an optimization-based makespan-optimal approach and present the sum-of-costs in Table I. The Distributed Reinforcement Learning approach [6] runs a pre-trained policy for 100 trials using 8 agents for each structure. It takes around nine days of training to converge to the final learned policy. Table I reports the average sum of costs over successful trials. Heuristic-based approach [3] uses a minimum spanning tree-based planning approach. In comparison, our approach offers improvement of an order of magnitude in the sum-of-costs compared to both approaches. Optimization-based Exact Approach [1] uses a Mixed Integer Linear Programming (MILP) based optimization model. The model first minimizes makespan and then sum of costs for the minimum makespan. We present the sum of costs in Table I which is closely comparable to that of our approach for the test structures. Table II presents data on their computation time in comparison with our approach. Computation of the high-level plan takes up more than 90% of this computation time for majority of our experiments. For all cases here, we enforce a run-time limit of 10,000 seconds for each structure for our comparison. The exact approach violates this limit for structures 3-5. Thus, our approach maintains solution quality in terms of sum-of-costs without being computationally expensive. We also achieve comparable makespans but skip the discussion here for the purpose of brevity.

VI. Conclusions

We present a centralized, hierarchical planning approach to the multi-agent collective construction problem. We provide an order-of-magnitude improvement in computation time, and maintain solution quality in the sum of costs as compared to the time-optimal baseline approach. In the future we aim to demonstrate scalability through refined methods. We also aim generalize this solution to challenging variation of the problem that include having heterogeneous block sizes or robust planning that can handle uncertainty in completion times.

TABLE II: Comparison of computation time with Exact Approach

Structure	Exact Approach (in s)	Our Approach (in s)
1	1115	**52**
2	135	**39**
3	-	**147**
4	-	**192**
5	-	**388**
6	1715	**116**

References

[1] Edward Lam, Peter J. Stuckey, Sven Koenig, and T. K. Satish Kumar. 2020. 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. Springer-Verlag, Berlin, Heidelberg, 743–758. https://doi.org/10.1007/978-3-030-58475-7_43.

[2] Justin Werfel, Kirstin Petersen, and Radhika Nagpal. 2014. "Designing Collective Behavior in a Termite-Inspired Robot Construction Team." Science, 343, 6172.

[3] Kumar, T.K.S. & Jung, Sangmook & Koenig, Sven. (2014). A Tree-Based Algorithm for Construction Robots. Proceedings of the International Conference on Automated Planning and Scheduling. 2014. 481-489. 10.1609/icaps.v24i1.13673.

[4] Petersen, Kirstin & Nagpal, Radhika & Werfel, Justin. (2011). TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction. 10.15607/RSS.2011.VII.035.

[5] Werfel, Justin & Nagpal, Radhika. (2006). Extended Stigmergy in Collective Construction. IEEE Intelligent Systems. 21. 20-28. 10.1109/MIS.2006.25.

[6] Sartoretti, G., Wu, Y., Paivine, W., Kumar, T.K., Koenig, S., & Choset, H. (2018). Distributed Reinforcement Learning for Multi-robot Decentralized Collective Construction. International Symposium on Distributed Autonomous Robotic Systems.

[7] Trevor Cai, David Y. Zhang, T.K. Satish Kumar, Sven Koenig, and Nora Ayanian. 2016. Local Search on Trees and a Framework for Automated Construction Using Multiple Identical Robots: (Extended Abstract). In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS '16). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1301–1302.

[8] Panangadan A, Dyer MG. Construction in a Simulated Environment Using Temporal Goal Sequencing and Reinforcement Learning. Adaptive Behavior. 2009;17(1):81-104. doi:10.1177/1059712308101787

[9] Barros dos Santos, Sergio Ronaldo & Givigi, Sidney & Nascimento Jr, Cairo. (2013). Autonomous construction of structures in a dynamic environment using Reinforcement Learning. SysCon 2013 - 7th Annual IEEE International Systems Conference, Proceedings. 452-459. 10.1109/SysCon.2013.6549922.

[10] Hönig, W.; Kumar, S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2017. Summary: Multi-agent path finding with kinematic constraints. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 4869–4873.

[11] Bechon, P., Barbier, M., Infantes, G., Lesire, C., & Vidal, V. (2014). HiPOP: Hierarchical Partial-Order Planning. Starting AI Researchers' Symposium.

[12] Knoblock, C.A. (1994). Automatically Generating Abstractions for Planning. Artif. Intell., 68, 243-302.

[13] J. Hoffmann, "FF: The Fast-Forward Planning System", AIMag, vol. 22, no. 3, p. 57, Sep. 2001.

[14] Grenouilleau, F., Hoeve, W.J., & Hooker, J. (2019). A Multi-Label A* Algorithm for Multi-Agent Pathfinding. International Conference on Automated Planning and Scheduling.