Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

APEX-MR: Multi-Robot Asynchronous Planning and Execution for Cooperative Assembly

Philip Huang*,[1], Ruixuan Liu*,[1], Shobhit Aggarwal[1], Changliu Liu[1] and Jiaoyang Li[1]
*Equal Contribution, [1]Carnegie Mellon University

***Abstract*—Compared to a single-robot workstation, a multi-robot system offers several advantages: 1) it expands the system's workspace, 2) improves task efficiency, and, more importantly, 3) enables robots to achieve significantly more complex and dexterous tasks, such as cooperative assembly. However, coordinating the tasks and motions of multiple robots is challenging due to issues, *e.g.*, system uncertainty, task efficiency, algorithm scalability, and safety concerns. To address these challenges, this paper studies multi-robot coordination and proposes APEX-MR, an asynchronous planning and execution framework designed to safely and efficiently coordinate multiple robots to achieve cooperative assembly, *e.g.*, LEGO assembly. In particular, APEX-MR provides a systematic approach to post-process multi-robot tasks and motion plans to enable robust asynchronous execution under uncertainty. Experimental results demonstrate that APEX-MR can significantly speed up the execution time of many long-horizon LEGO assembly tasks by 48% compared to sequential planning and 36% compared to synchronous planning on average. To further demonstrate performance, we deploy APEX-MR in a dual-arm system to perform physical LEGO assembly. To our knowledge, this is the *first* robotic system capable of performing customized LEGO assembly using commercial LEGO bricks. The experimental results demonstrate that the dual-arm system, with APEX-MR, can safely coordinate robot motions, efficiently collaborate, and construct complex LEGO structures. Our project website is available at https://intelligent-control-lab.github.io/APEX-MR/.**

I. Introduction

Multi-robot manipulation is critical in robotic applications, such as industrial assembly [37, 3], material handling [50], and object arrangement [12, 13], etc. Compared to a single-robot setup, a multi-robot arm system can easily expand the system's overall reachable area. Besides, with a team of robot arms, a task can be accomplished more efficiently [49] by having each robot execute individual tasks in parallel. In addition to improving task efficiency, multi-robot systems can achieve significantly greater dexterity and are *necessary* in many applications that require collaborations, *e.g.*, cooperative assembly, since certain tasks cannot be done with only one arm.

LEGO assembly is an example of cooperative assembly. In a LEGO structure, the bricks are assembled by forcing the top knobs of a brick into the bottom cavities of another brick. The bricks are held together passively, *i.e.*, by the friction in the knob-to-cavity connections. Thus, the connections are not rigid and subsequent manipulation, if performed inappropriately, can easily break the existing structure. Due to the nature of the passive connection, a multi-robot system is necessary for such cooperative assembly. Fig. 1 demonstrates example assembly

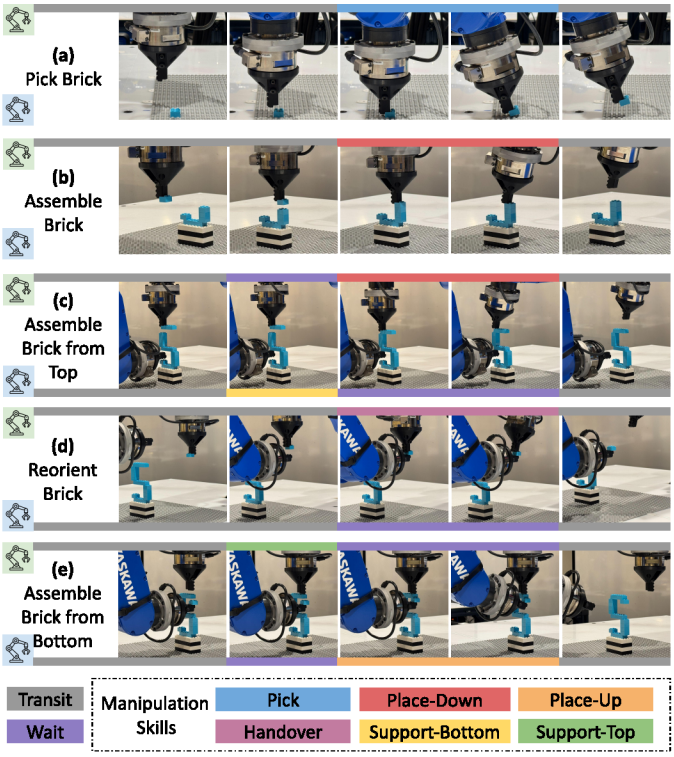


Fig. 1: Illustrations of bimanual cooperative assembly. Manipulation skills denote contact-rich operations for assembly.

operations in the construction of a LEGO structure. Fig. 1 (a) shows the robot picking up a brick by disassembling it from the LEGO plate (*i.e.*, pick). Fig. 1 (b) illustrates the robot assembling a brick by placing it at the desired location and forcing a solid connection (*i.e.*, place-down). One robot is sufficient for these two tasks since the manipulated object is fully supported and the operation would not collapse the existing structure. On the other hand, Fig. 1 (c)-(e) showcase operations that require multi-arm collaboration. In Fig. 1 (c), the upper robot is assembling a LEGO brick on top of the character 'S'. To establish a solid connection, the robot needs to press down and force the knob insertion due to the passive connection nature. However, since the existing connections are non-rigid, the place-down operation would break the existing overhanging structure. Therefore, the lower robot is necessary to support and stabilize the structure from below (*i.e.*, support-bottom). Similarly, in Fig. 1 (e), the lower robot assembles a brick from the bottom onto the character 'S' by pushing it up and forcing the connection (*i.e.*, place-

up). The place-up operation would break the existing structure and therefore the upper robot is required to press down (*i.e.*, support-top) in order to stabilize the structure. In addition to cooperative assembly, multi-robot collaboration also enables more dexterous object manipulation, *e.g.*, reorienting bricks in hand. As shown in Fig. 1 (d), the upper robot initially grabs a brick from its top. To have the robot grab the brick from its bottom, we can have the upper robot handover the brick to the lower robot. With the ability to reorient objects in-hand, the robot can subsequently accomplish the assembly from the bottom as illustrated in Fig. 1 (e). Despite being a toy brand, LEGO has been widely used in entertainment, education, prototyping, etc. It is ideal for assembly benchmarking because it is a low-cost, standardized, and highly customizable assembly platform. Meanwhile, constructing LEGO structures is a challenging contact-rich manipulation problem due to the high-precision requirement and non-rigid connections. Thus, we use LEGO as our cooperative assembly benchmarking platform, and the remaining discussion is in the context of LEGO assembly.

Coordinating the tasks and motions of multiple robots to accomplish cooperative assembly is challenging for several reasons. First, a system with more than a single robot introduces overhead and algorithmic challenges in their coordination. As shown in Fig. 1, certain operations involve contacts with objects (*i.e.*, pick, place-down, place-up, handover, support-bottom, and support-top) while some do not (*i.e.*, transit and wait). There exist delays in controlling the robot and receiving sensor feedback, or even contingencies due to unexpected events when performing contact-rich operations, which would cause delays or stops to one or more robots. The uncertainty makes it challenging to safely coordinate the robots throughout the assembly process. Second, due to the need for collaboration, robots often need to operate closely to each other as illustrated in Fig. 1 (c)-(e). Even when executing under uncertainty in a confined shared workspace, robots must always avoid collisions. Third, it is desired that the task can be accomplished more efficiently by a multi-robot system. Thus, robots must have a comprehensive understanding of the cooperative assembly task and plan optimized motions to reduce the completion time, instead of frequently stopping to avoid collisions. Lastly, the multi-robot system should scale as the complexity of assembly design increases. It is necessary for the algorithm to scale to larger and more complex tasks.

Many recent works have proposed methods for planning multiple robot arms for assembly [22, 5], rearrangement [12, 14], or general manipulation tasks [45]. However, these planners often assume a sequential execution order and synchronously moving robots, where all robots start a task at the same time and wait for other robots to finish. We take a different lens and propose APEX-MR, an *asynchronous* planning and execution framework to coordinate multiple robots robustly, efficiently, and safely. Given an assembly task sequence, APEX-MR leverages integer-linear programming (ILP) to distribute the tasks to robots and generate a sequential robot plan. Most importantly, APEX-MR extends the temporal plan graph (TPG) [19] to multi-robot-arm systems and post-processes the sequential robot plan for asynchronous execution. Specifically, the TPG captures all dependencies between different robots' tasks and motions and generates a partial-order graph, which can significantly reduce unnecessary wait time and is robust to execution delay and uncertainty. To highlight the applicability of our proposed algorithm within a full multi-robot assembly pipeline, we deploy the proposed APEX-MR to a dual industrial arm system to construct complex customized LEGO structures up to 258 objects in simulation and up to 47 objects in the real world. In summary, our contributions are as follows:

- We extend the TPG execution framework to multiple robot arms and show that it enables robust and safe multi-robot asynchronous execution under uncertainty.
- We propose a sequential multi-robot task and motion framework that is complementary to the TPG execution.
- We demonstrate algorithm deployment and system integration for bimanual LEGO assembly. To our knowledge, this is the *first* robotic system that can accomplish flexible (*i.e.*, customized complex designs instead of simple pick and stack) assembly using commercial LEGO bricks.

The rest of this paper is organized as follows: Sec. II discusses relevant works. Sec. III deliberates the input that APEX-MR consumes and the assumptions we have. Sec. IV introduces our proposed APEX-MR, an asynchronous planning and execution framework for multi-robot cooperative assembly. Sec. V shows the experimental results and demonstrates the deployment to a bimanual system for LEGO assembly. Sec. VI discusses limitations and future works. Sec. VII concludes the paper.

II. Related Works

Multi-Agent Path Finding and Execution Multi-agent path finding (MAPF) [54] studies how to coordinate a large team of mobile robots in a discretized world environment, often in 2D grid worlds representing warehouse settings [33] or predefined roadmaps [21]. State-of-the-art MAPF algorithms can plan near-optimal collision-free paths for hundreds of robots in seconds [26].

However, these MAPF algorithms achieve such impressive efficiency at the cost of neglecting robot kinematics and execution uncertainty. As a result, there is growing research within the MAPF community focused on the efficient and robust execution of (imperfect) MAPF plans on real robots. One of the most widely adopted frameworks is the temporal plan graph (TPG) [19], originally proposed to post-process MAPF plans to meet robot kinematics by enforcing passing orders at locations visited by multiple robots. TPG has since been extended to MAPF execution frameworks under uncertainty [32] and mobile robot coordination in warehouses [20, 59]. Recent advances introduce bidirectional TPG [56] and switchable TPG [1, 23], which further enhance TPG by allowing flexible passing orders at certain locations.

Given the success of TPG in coordinating mobile robots, we aim to explore its applicability in coordinating robot arms. A key distinction of traditional TPG versus our use case is

that the robot kinematics is more complex and may change over time as the robot arm picks up different objects, which is a key focus in this paper.

Multi-Robot Arm Motion Planning Motivated by the rise of bimanual manipulation systems, many early works in the field study the problem of dual arm motion planning [51]. One naive approach that scales single-robot motion planning methods to two or more robots is to plan in the composite joint space, with methods such as RRT-Connect [25], BIT* [11], or graph of convex sets [35]. However, due to the curse of dimensionality, these methods struggle to find high-quality solutions as the number of robots increases. Other common strategies include building a composite roadmap from the Cartesian product of individual roadmaps [15], coordinating the speed of individually planned motions [2], or using prioritized planning [8]. More recently, more specialized multi-robot motion planners have been proposed that are based on roadmaps (dRRT [53], dRRT* [48], and CBS-MP [52]) or using MAPF techniques [44, 45]. Some approaches also use online planning or control techniques to generate motions in real-time. Zhang and Pecora [61] propose an online motion coordination technique, in which they plan all robots' paths offline and a pairwise collision matrix between robots. Then, the speed of each robot can be efficiently planned online to avoid collisions, even in the presence of execution delays. However, their method cannot always find a feasible motion and relies on a task reallocation process to avoid deadlock. Other techniques such as a distributed model predictive controller [10] or a dynamical system approach [38] can also be used to generate multi-robot arm motions in real-time, but are not tailored towards long-horizon planning tasks.

Multi-Robot Task and Motion Planning Beyond motion planning, multi-robot arm task and motion planning (MR-TAMP) have been studied since the 1990s, *e.g.*, object pick and place [24], and many of which are designed for a dual arm setting [17, 12, 13, 51]. Similarly to our method, these methods take a two-stage approach in which they first generate a task plan with robot assignment and grasp poses, then search for corresponding motion plans with composite state space or prioritized planning. The authors of dRRT* have proposed extending dRRT* to multi-modal roadmaps with given possible pick-up and hand-off configurations and searching for a task and motion plan directly [46, 47]. Given that motion planning calls tend to be costly, another approach is to generate promising task plans first in a lazy manner and subsequently find corresponding motion plans and backtrack when required. This approach can be implemented with a greedy method [18], with a mixed-integer linear program [49, 5, 31] or through a satisfiability modulo theories solver [42]. However, a major drawback is that task planning often assumes motions to be synchronous, which can be suboptimal in practice. Our TPG framework is designed to be complementary to these synchronous tasks and motion planners, as it can relax the synchronicity assumption with post-processing and shorten the makespan of the overall plan. Another significant concern is that many MR-TAMP methods are designed for simple environments such as object pick and place, and their planned robot paths are executed in open loop or only in simulation. Our framework scales to more complex environments and is designed to integrate with feedback controllers and manipulation skills.

Multi-Robot Arm Motion Execution Most existing work executes their multi-robot task and motion plan on real robots in a synchronized way. Since the popular motion planning framework *MoveIt* [7] does not natively support moving multiple robots asynchronously, some recent work has sought to address this and enable asynchronous execution. Meehan et al. [36] adds a path reservation component and treats the entire path of a moving robot arm as a static obstacle when planning another arm's motion, which can be too conservative and cause deadlock. Stoop et al. [55] uses a central scheduler to check if a new path collides with previously scheduled paths and executes it asynchronously if there is no collision. Otherwise, the new path waits for the conflicting previous path to finish before it can start. However, their methods do not account for execution delays, may wait longer than necessary, and directly modify the *MoveIt* software stack. In contrast, our TPG formulation is robust to arbitrary delays by design, minimizes robot wait time, and can be implemented without modification to *MoveIt*.

Robotic LEGO Assembly Automating LEGO assembly using robots is challenging due to the high-precision requirement, tiny sizes of LEGO bricks, and non-rigid connections in the structure. Most of the existing works address the LEGO assembly problem in simulation [43, 39], which cannot be generalized to physical assembly due to the lack of simulators to simulate the connections between LEGO bricks. Recent works [9, 16, 27, 29] assume that the structure is fully supported and only consider placing bricks on top of others, which are limited to assembling simple structures. [34] considers assembly using customized brick toys, which does not apply to LEGO assembly. In this paper, we apply APEX-MR to a bimanual system to construct complex customized LEGO structures beyond simple stacking.

III. Preliminaries

To coordinate a multi-robot system to perform cooperative assembly tasks, we assume that three inputs are provided.

Environment Setup We assume that the environment setup, including (a) geometries of all robots, (b) poses of objects to be manipulated $\mathcal{B} = [b_1, b_2, \ldots, b_{N_b}]$, and (c) states of all obstacles, is known as shown in the input section in Fig. 2. We assume that the system consists of N robots and N_b is the number of objects that can be used for the assembly. An object b_k is semi-static, meaning it can be grasped, attached, and moved by robots. Note that since duplicate objects are common in assemblies, b_k and $b_{k'}$ can be identical objects, *e.g.*, identical 2×2 bricks in the character 'S' shown in Fig. 2.

Assembly Plan Given an assembly design with N_a objects, we assume that the assembly plan $A = [a_1, a_2, \ldots, a_{N_a}]$ is provided as shown in the input section in Fig. 2. Each step a_j refers to an object, such as a 1×2 brick. The assembly plan

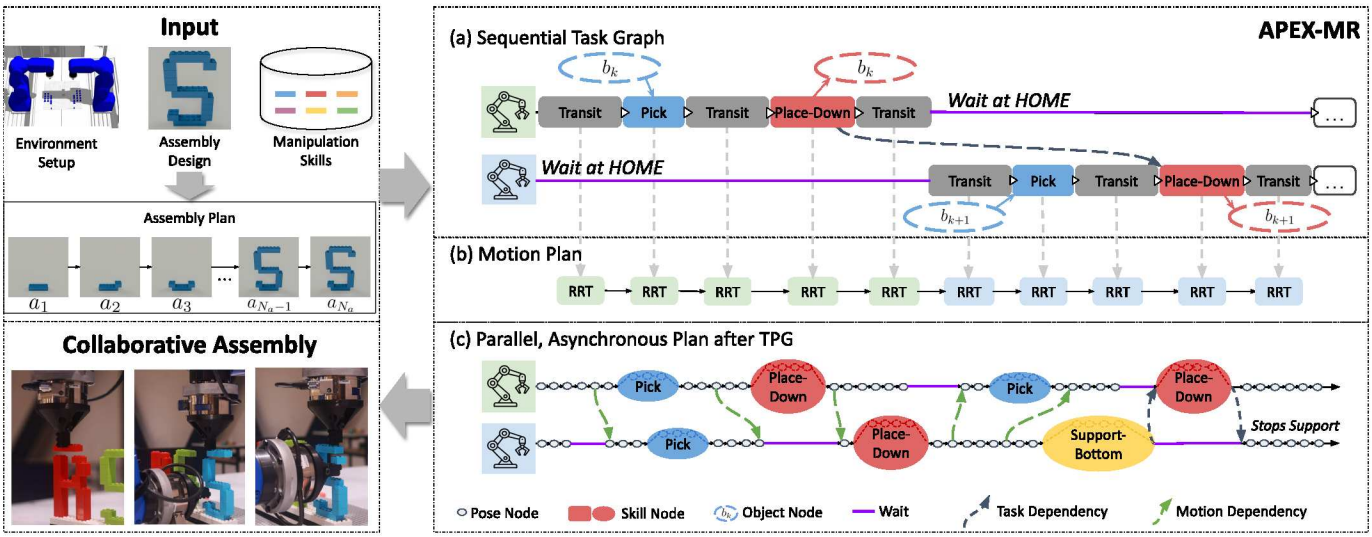


Fig. 2: An overview of APEX-MR. On a high level, APEX-MR builds a sequential task plan given an assembly sequence, plans the motion of each task with RRT-Connect, and converts the solution to a parallel, asynchronous plan for execution with a TPG. Specifically, (a) shows the example of a task graph, (b) illustrates generating robot motion from the task plan, and (c) shows the example of a multi-modal TPG.

specifies the order in which each object should be assembled. Note that due to duplicate objects in $\mathcal{B}$, there are multiple candidates $b_i, \ldots, b_k$ that can be used to accomplish an assembly step a_j in A. Furthermore, we require the assembly plan A to be physically valid [58, 30], ensuring that each task can be performed in reality. Specifically, the partially assembled structure after each step a_j must be physically stable, and there exists at least one feasible grasp pose for each step. The assembly sequence also specifies whether each step a_j requires two robots for cooperative assembly and the specific type of cooperation needed (*i.e.*, support or reorientation). Lastly, we assume that $\mathcal{B}$ is sufficient so that each object required for a_j can be found in the environment. More details on generating the assembly plan A is included in Sec. V and appendix A.

Manipulation Skills We assume that the robot manipulation skills are predefined and known. We denote the skill set as $\mathcal{S} = [s_1, s_2, \ldots, s_{N_s}]$, where N_s is the number of skills an individual robot has. For instance, a skill can be, inserting a pin, fastening a screw, picking up an object, etc. In our case of LEGO assembly, each skill is a composite of multiple motions parametrized by the pose of the manipulated object, the robot end-effector, and parameters learned from demonstrations. We assume that an algorithm, such as interpolation or RRT-Connect, can generate a reference robot path for the purpose of computing a collision-free coordination schedule. During execution, each skill is executed with a feedback controller, so that its exact execution time is unpredictable. Examples of manipulation skills are shown in Fig. 1, and more details are discussed in Sec. V and appendix C.

In addition to these inputs, we assume the existence of a collision-free HOME pose for each robot that never blocks other moving robots from executing their tasks. Robots can also plan paths to move between different poses, or wait and hold their current pose in place, as shown in Fig. 1.

IV. APEX-MR: Asynchronous Planning and Execution for Multi-Robot System

In this section, we introduce APEX-MR, a framework for asynchronously coordinating the task plan, motion, and execution of a multi-robot system to accomplish cooperative assembly tasks. Fig. 2 provides an overview of the three stages of APEX-MR. The first task planning stage is discussed in Sec. IV-A. Given the input discussed in Sec. III, a sequential task plan and the corresponding task graph, as shown in Fig. 2 (a), are generated from the assembly using ILP. In the next stage, explained in Sec. IV-B, the motion plan for each task is generated sequentially with a single-robot motion planner (see Fig. 2 (b)). Sec. IV-C presents the last and most crucial stage, which converts the sequential plan to a multi-modal TPG (e.g. Fig. 2 (c)) that can be executed safely, asynchronously, and efficiently in real robots.

For notation, we use $i \in [1, N]$ to index robots, $j \in [1, N_a]$ to index assembly steps in A, m to index tasks for each robot, and $k \in [1, N_b]$ to index moveable objects in $\mathcal{B}$.

A. Task Planning

Given the assembly sequence A, task planning aims to construct a sequential task plan that includes robot assignment, robot target pose assignment, and object assignment. Specifically, for each step a_j in the assembly sequence, the task planner must assign one responsible robot, a support robot if required, and an object b_k of the correct type. In addition, the task planner must select the feasible grasp pose and a feasible support pose if required to perform the necessary manipulation skills for each a_j. Fig. 3 provides an overview of all the decisions made. These decisions directly affect the feasibility

of motion planning, as task planning determines whether there are collision-free and deadlock-free paths.

Task Definition We define a task $\mathcal{T}^i$ as a piece of work that requires a robot i to either transit to a goal pose or to perform some manipulation skill s to achieve a goal constraint. For example, a task may involve a robot arm picking up an object, supporting a structure, receiving an object handed from another robot, retracting to its HOME pose, etc. In the context of LEGO assembly, each assembly task a_j is broken down into a sequence of tasks. As shown in Fig. 1 (a) and (b), a robot must first transit to the initial location of an object, pick the object, transit the object to the location for assembly, place down the object on the target structure, and finally transit back to the robot's HOME pose. For an assembly step that requires collaborative assembly, the support robot is also assigned a sequence of tasks, such as transiting to the structure and supporting it in Fig. 1 (c) and (e). If a reorientation is required, as illustrated in Fig. 1 (d), the support robot is assigned a transit task and a pick task to collect the object, and a handover task to the other robot so that the object can be placed up to the structure.

Task Graph A task graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a direct acyclic graph that represents the ordered set of tasks for all robots to complete the final assembly. The task graph also implicitly represents the movement of manipulated objects, the evolving environment and planning scene, as well as the kinematics switches of the robots. A node is either a task node or an object node. A task node represents a task $\mathcal{T}_m^i \in \mathcal{V}$ for robot i, where m is the index. Each task node also contains a robot target pose for this task. An object node represents an object b_k and its pose. An edge from one task node to another in the graph, $\mathcal{T}_m^i \rightarrow \mathcal{T}_{m'}^{i'}$, represents a precedence constraint that states a dependency where $\mathcal{T}_m^i$ must finish before $\mathcal{T}_{m'}^{i'}$ can start. An edge from an object node to a task node, $b_k \rightarrow \mathcal{T}_m^i$, means that the object b_k is kinematically attached to the robot i at the beginning of this task. In contrast, an edge from a task node to an object node, $\mathcal{T}_m^i \rightarrow b_k$, represents that an object would be detached from robot i after this task ends. Each robot i has M^i tasks. A task graph itself does not limit whether the tasks must be executed sequentially, synchronously, or asynchronously.

Approach The main idea of our approach is to find a sequential turn-based task plan according to the assembly sequence A, which is itself sequential. Only one robot is actively moving or executing skills at any time, while the other robot waits. Each robot returns to its HOME pose at the end of completing an assembly step.

Algorithmically, an ILP jointly optimizes robot assignment, object assignment, and target robot poses. A set of binary decision variables assigns a robot i to an assembly step a_j using the object b_k and a feasible grasp pose. Another set of binary decision variables denotes the assignment of the support robot and support poses. We precompute feasible robot poses for grasping all available objects at initial and assembled positions, and support poses if necessary. The cost of assigning a robot i, an object b_k, and a corresponding grasp pose to an assembly step a_j is estimated by the sum of the transit distances necessary for this step. The ILP finds the best set of assignments that minimizes the sum of costs to complete the assembly and an auxiliary term for load balancing while ensuring that the object type and support robot requirements are met. More details of the ILP formulation are discussed in appendix B.

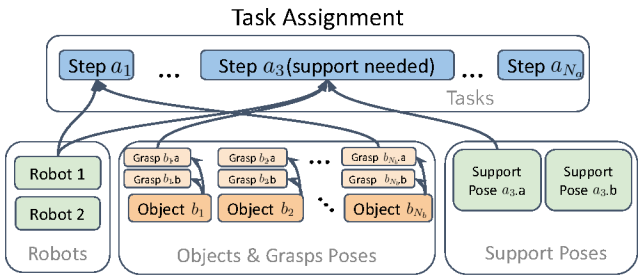


Fig. 3: An overview of the task planning in APEX-MR. Given the assembly sequence A, each step a_j is assigned a robot, an object, a feasible object grasp pose, and a supporting pose if necessary.

Combined with the precomputed robot poses, the optimized assignment gives a complete set of robot, object, and grasp pose assignments in each step. We then construct a corresponding sequential task plan and task graph (*e.g.*, Fig. 2 (a)) with all inter-robot task dependencies and object relationships. A sequence of tasks for the assigned robot is added for each assembly step, and two object nodes are added to the task graph and connected to the pick task and place task nodes to indicate their attachment and placement, respectively. If a collaborative assembly is required, a sequence of support tasks is added to the support robot to be completed first. Inter-robot task dependencies are added to the task graph to constrain that the support task must precede any place task, and the following task after support can only start after the place task finishes. If two consecutive assembly steps a_j and a_{j+1} are assigned to different robots, a task dependency is added to ensure that the place task for a_{j+1} must wait for the place task for a_j.

Compared to other MR-TAMP and assembly methods such as [13, 18, 42, 5], APEX-MR first generates a sequential plan. This has two advantages: (1) It is easy to reason about inter-robot collision for scheduling tasks and avoids expensive feasibility checks needed for parallel task execution, thereby making the ILP easier to solve; (2) The complexity of motion planning is significantly reduced since each task becomes a single-robot planning problem, which avoids solving a challenging and time-consuming multi-robot arm motion planning problem. It is worth noting that, while the sequential plan might seem inefficient for a multi-robot system, our TPG execution framework will post-process this plan to enable efficient parallel execution.

B. Motion Planning

Once the sequential task plan for each robot is determined, motion planning becomes straightforward. As illustrated in Fig. 2 (b), APEX-MR iterates over each task and uses a single-robot RRT-Connect algorithm to plan the path for this task

when other robots are waiting. This is feasible because all other robots would be at a nonblocking stationary pose, *i.e.*, HOME[1]. A reference path is also generated for tasks executed by specific manipulation skills based on the grasp/support pose.

The motion planner generates a planned path for every robot and every task. Each path τ_m^i for task $\mathcal{T}_m^i$ of robot i is represented with a sequence of uniformly timestamped poses, $\{(C_n^i, t_n^i)\}, n \in [N_{start,m}^i, N_{end,m}^i]$ with step size Δt. $N_{start,m}^i$ and $N_{end,m}^i$ are the first and last index of the path τ_m^i, $N_{start,1}^i = 1$, and $N_{start,m+1}^i = N_{end,m}^i + 1$ for $m = 1, \dots M-1$. For each task, the planner first generates a sequence of poses C_n^i for the robot i, then determines the corresponding timesteps. To ensure that each robot takes turns to complete its task according to the task sequences, the initial timestep of each task is equal to the last timestep of the previous task in the sequential plan. Then, the timestep t_n^i for each pose of the same task is set based on the maximum robot velocity, *i.e.*, $t_n^i = t_{n-1}^i + d(C_n^i, C_{n-1}^i)/v_{max}$. Since RRT-Connect may produce jerky and long paths, we use a randomized shortcutting algorithm to smooth suboptimal paths. The output of motion planning should be a sequence of paths, one for each task, *i.e.*, τ_m^i, $\forall i \in \{1, \dots, N\}$, $\forall m \in \{1, \dots, M^i\}$.

C. Asynchronous Execution

From the sequential task and motion plan, APEX-MR converts it to a TPG to improve the quality of the plan and support asynchronous execution. Importantly, the process to construct a TPG is not limited to a sequential plan and also applies to synchronous plans commonly seen in MR-TAMP works such as [42, 47, 49].

TPG Definition APEX-MR uses a multi-modal temporal plan graph (TPG) to represent an execution schedule for the team of robot arms. As illustrated in Fig. 2 (c), multi-modal TPG is a directed acyclic graph $G = (V, E)$ with two types of nodes. A pose node v_n^i corresponds to a configuration C_n^i on the path of a transit task for robot i. A skill node $\tilde{v}_n^i$ represents a manipulation skill (*e.g.*, pick, handover, support, etc.) that will be executed by some robot controller or policy. The skill node also contains a reference path generated in motion planning. A type-1 edge $(v_n^i \to v_{n+1}^i)$[2] connects two nodes from the same robot i and indicates the order between two nodes. A type-2 edge $(v_n^i \to v_{n'}^{i'})$ represents an inter-robot precedence order that constrains robot i' to wait for robot i to reach v_n^i (*i.e.*, reach pose C_n^i or complete the manipulation skill) before moving to $v_{n'}^{i'}$. A type-2 edge can be added for both task dependencies and motion dependencies. For every pair of TPG nodes that could lead to collisions, this is a motion dependency, and a type-2 edge is required. TPG ensures a deadlock-free execution schedule if and only if there are no cycles in the TPG. In contrast to the TPG defined in [19], a multi-modal TPG combines a TPG with a task graph and assigns a corresponding task $\mathcal{T}_m^i$ to each node v_n^i. Since there are kinematics switches and changes to the planning scene, each node also contains the robot kinematics (*i.e.*, attached object), which will be important for the TPG construction process.

[1]We also assume that any robot waiting at a support pose is nonblocking.

[2]Type-1 edge connects skill nodes too, *i.e.*, $(\tilde{v}_n^i \to v_{n+1}^i)$ or $(v_n^i \to \tilde{v}_{n+1}^i)$

Algorithm 1 Multi-Modal TPG Construction

▷ *Input: A task graph $\mathcal{G}$ and all robot paths τ_m^i* ◁
for robot $i = 1, \dots, N$ **do**
  for task $\mathcal{T}_m^i = \mathcal{T}_1^i, \dots, \mathcal{T}_{M^i}^i$ **do**
    if task $\mathcal{T}_m^i$ is a transit to a goal pose **then**
      Add a sequence of nodes $\{v_n^i\}$ from τ_m^i
      Add type-1 edges for consecutive nodes
    else Add a skill node $\tilde{v}_n^i$
    Add type-1 edge from $v_{end,m-1}^i$ to $v_{start,m}^i$
for task dependency $(\mathcal{T}_m^i \to \mathcal{T}_{m'}^{i'}) \in \mathcal{E}$ such that $i \neq i'$ **do**
  Add type-2 edge from $v_{start,m+1}^i$ to $v_{start,m'}^{i'}$
for robots $(i, i') = \{1, \cdots, N\} \otimes \{1, \cdots, N\}$ **do**
  if $i == i'$ **then** continue
  ▷ *(Optional) parallelize the following* ◁
  for $n = 1, \dots, N_{end,M^i}^i$ **do**
    Update robot i kinematics if needed
    for $n' = 1, \dots, N_{end,M^{i'}}^{i'}$ **do**
      Update robot i' kinematics if needed
      if $t_{n'}^{i'} \geq t_n^i$ or $v_{n'}^{i'}$ precedes v_n^i in G **then**
        continue
      if $v_{n'}^{i'}$ collides with v_n^i **then**
        Add type-2 edge from $v_{n'+1}^{i'}$ to v_n^i
Simplify TPG with transitive reduction

Building a multi-modal TPG The process of constructing a TPG from a multi-robot motion plan can be interpreted as converting from the sequential robot paths to temporally dependent robot schedules. On a high level, inter-robot task dependencies are copied as type-2 edges for TPG, and type-2 edges for motion dependencies are constructed by scanning for collisions between all pairs of TPG nodes. A key benefit of this partial-order representation is that TPG does not specify a fixed time between two consecutive nodes, and thus allows execution delays. We outline the detailed procedure below.

The construction process begins with creating the nodes and type-1 edges. For each transit task $\mathcal{T}_m^i$, a pose node v_n^i is constructed for each configuration C_n^i of the path τ_m^i. A skill node $\tilde{v}_n^i$ is created for every manipulation task. The timestamp in the input path t_n^i for each node v_n^i is also recorded. Each node v_n^i is then connected to its successor v_{n+1}^i by a type-1 edge, and the end node of a task $v_{end,m}^i$ is connected to the start node of the next task $v_{start,m+1}^i$. When a robot waits at a node v_m^i, that is $C_m^i = C_{m-1}^i$, the node v_m^i is removed because removing the wait action does not change the robot path or any temporal dependency in the TPG.

Next, all the type-2 edges are identified in the TPG. First, all inter-robot task dependencies $(\mathcal{T}_m^i \to \mathcal{T}_{m'}^{i'}) \in \mathcal{E}$ from the task graph $\mathcal{G}$ are added as type-2 edges to the TPG. Specifically, for every edge in the task graph $\mathcal{T}_m^i \to \mathcal{T}_{m'}^{i'}$, a type-2 edge is added from the beginning of task $\mathcal{T}_{m+1}^i$, $v_{start,m+1}^i$, to the

beginning of task $\mathcal{T}^{i'}_{m'}$, $v^{i'}_{start,m'}$. This type-2 edge ensures that robot i only starts $\mathcal{T}^{i'}_{m'}$ after robot i' finishes task $\mathcal{T}^{i}_{m}$.

Then, motion-level dependencies are identified by iterating over all pairs of nodes in the TPG with a double loop iterating over (i, i'). Robot kinematics can be incrementally changed if node v^i_n is the start of a new task that attaches or detaches an object. For each node v^i_n, it is checked against all other nodes $v^{i'}_{n'}$ from a different robot ($i \neq i'$) and has an earlier timestamp $t^{i'}_{n'} \leq t^i_n$. Collision checking for pose nodes means that the robot links and attached objects at the corresponding poses C^i_n and $C^{i'}_{n'}$ are checked. For skill nodes, all robot poses on the reference path have to be collision-free simultaneously. If there is any collision between $v^{i'}_{n'}$ and v^i_n, a type-2 edge is added from the earlier node's successor, $v^{i'}_{n'+1}$, to the later node, v^i_n. Setting the direction of type-2 edge based on timestamps in the sequential motion plan ensures that the multi-modal TPG has no cycles and thus is deadlock-free. This way, if $v^{i'}_{n'}$ has a smaller timestamp than v^i_n in the input sequential plan, *i.e.*, $t^{i'}_{n'} < t^i_n$, $v^{i'}_{n'}$ must still be executed before v^i_n can start, avoiding any potential collisions. When iterating n', it is unnecessary to check collisions for any node $v^{i'}_{n'}$ that has a larger timestamp than v^i_n, *i.e.*, $t^{i'}_{n'} \geq t^i_n$, because those would be checked when the iterated robot pair $(i,\ i')$ are swapped. This ensures the eventual multi-modal TPG is cycle-free and hence deadlock-free, since the Also, if the current $v^{i'}_{n'}$ is a predecessor of v^i_n in the graph, v^i_n already waits for $v^{i'}_{n'}$ and avoid collisions, so it becomes unnecessary to check them again.

The total number of collision checks needed depends on the number of robots, discretized steps, and type-2 edges added from the task graph. Once every node has been checked against potentially colliding nodes, a transition reduction algorithm is applied, similar to [32] to simplify the TPG. A type-2 edge $v^i_n \rightarrow v^{i'}_{n'}$ is redundant if node v^i_n is still a predecessor of node $v^{i'}_{n'}$ after the edge is removed. We remove all such redundant edges to reduce the total number of scheduling constraints and communication overheads during execution.

The primary bottleneck of the TPG construction process is the number of collision checks, which scales quadratically with respect to the number of robots and the number of nodes. To alleviate this, collision checking can be parallelized across many CPU threads, reducing its runtime. A pseudocode of the entire construction process is provided in Algo. 1.

Further Optimization An optional step to further reduce the execution makespan and smooth the path is to skip the intermediate transition to HOME after every assembly step. We use the following shortcutting algorithm, similar to the strategy implemented in [41], to achieve that while maintaining a collision- and deadlock-free plan.

The anytime algorithm works by randomly sampling two nodes of the TPG $(v^i_n, v^i_{n'})$ from the same transit task $\mathcal{T}^i_m$, checking whether connecting them in a shortcut is feasible. Consecutive transit tasks passing through the robot's HOME pose are merged as a single task. This allows the HOME pose to be skipped. A shortcut path directly interpolates between

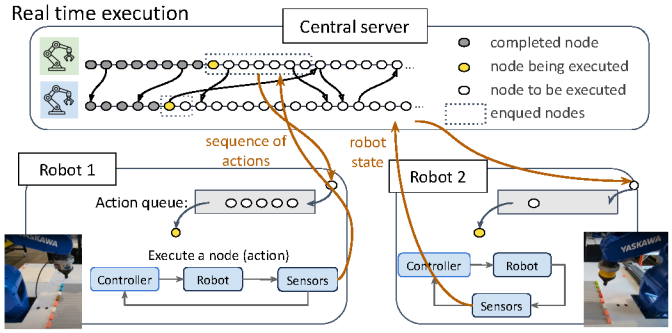


Fig. 4: Illustration of the execution setup. TPG maintains and controls the execution schedule of all robots on a central server, gradually sending new path segments that can be safely executed. Each robot maintains a controller-sensing loop independently while updating its state with the central server.

C^i_n and $C^{i'}_{n'}$ to generate a sequence of poses with the same step size Δt. The shortcut must be collision-checked against any independent nodes (*i.e.*, nodes that are not predecessors to v^i_n or successors to $v^{i'}_{n'}$ in the TPG). On a multi-modal TPG, the collision checking must include any attached objects to the robot, as well as any independent object nodes on a task graph (*i.e.*, object nodes that are not predecessors or successors of the current task). Once a valid shortcut is found, the original nodes between v^i_n and $v^i_{n'}$ are replaced by new nodes corresponding to the shortcut. If adding a valid shortcut from v^i_n to $v^i_{n'}$ skips any outgoing edges between these two nodes, the start nodes of these outgoing edges are moved to v^i_n. If any incoming edges are skipped, then the end nodes of these incoming edges are moved to $v^i_{n'}$. These two steps ensure that any dependencies before adding a shortcut still exist after, and the TPG remains collision-free. Since each shortcut is collision-free, no new dependencies in the TPG are introduced, and TPG remains deadlock-free. The shortcutting algorithm keeps identifying valid shortcuts until a user-defined time limit is reached, removing redundant transitions to HOME poses in the process.

D. TPG Execution

Executing a motion plan on robot arms often requires a position controller for movement and other specific controllers for manipulation skills. These controllers may have delays or uncertainties that affect the real-robot execution time. The TPG formulation provides an easy way to execute a multi-robot plan. Here, we present a semi-centralized mechanism to coordinate multiple robot arms.

As shown in Fig. 4, the TPG is hosted on a central server that communicates with each robot's execution thread. Each pose node in the TPG corresponds to an action that moves the robot's position to the node's configuration. Each skill node corresponds to an action that executes the predefined robot skill. An action can be executed safely if there are no incoming edges from any nodes that are not executed. Although each TPG node is associated with a timestamp, it is only used for

(a) (b) (c)

Fig. 5: Experiment setup for bimanual LEGO assembly. (a) Simulation environment. (b) Real setup. (c) Illustration of the EOAT, *i.e.*, LT-V2, for the robot to construct LEGO structures.

TPG construction and is ignored during execution.

If an action is safe to execute based on the TPG, the central server sends it to the robot's action queue. Each robot maintains its own controller-sensing loop and actuates the robot according to upcoming commands and its state estimation. The state estimation is also shared with the TPG, which then updates the TPG when a node is being executed or completed. Newly completed nodes can enable the central server to enqueue new nodes if their outgoing edges were previously preventing unsafe actions. During execution, TPG can be interpreted as a control law that maintains the safe scheduling of individual robot actions.

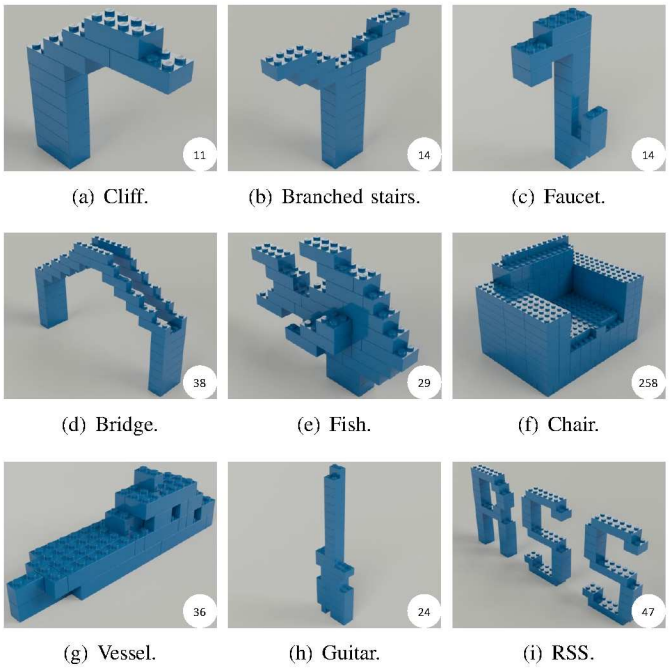
V. Results

To evaluate performance, we apply the proposed APEX-MR to bimanual LEGO assembly tasks. Given a customized LEGO design as shown in Fig. 6, APEX-MR coordinates the robots to construct the desired structure as shown in Fig. 9 using available LEGO bricks. We deliberate the inputs (introduced in Sec. III) to APEX-MR below.

Environment Setup Figure. 5(a) and 5(b) illustrate the simulation environment and the real setup, which includes two Yaskawa GP4 robots. Following the task convention in [29], we consider building LEGO structures on a baseplate, which is calibrated[3] and placed between the two robots, using commercial standard LEGO bricks initially stored on the baseplate. Each robot is equipped with an ATI Gamma force-torque sensor (FTS), and the end-of-arm tool (EOAT) is mounted on the FTS. The simulation consists of the entire workspace, which includes robots, FTS, EOAT, LEGOs, nearby workstations, etc. The complete digital environment provides rich and accurate information for APEX-MR to safely coordinate robot collaboration.

Assembly Plan Given a LEGO structure, we employ the physics-aware assembly planning in [30] with customized LEGO physics reasoning [28] to generate a physically valid assembly sequence. Specifically, a physically valid assembly sequence enforces that for each step after assembling a brick, the structure is stable and does not collapse. More details

[3]We calibrate the transformation from the robots to the baseplate by teleoperating the robot to touch the plate. Note that we only measure the translation (X, Y, Z) and yaw angle while assuming no roll and pitch offsets.



(a) Cliff. (b) Branched stairs. (c) Faucet.

(d) Bridge. (e) Fish. (f) Chair.

(g) Vessel. (h) Guitar. (i) RSS.

Fig. 6: Customized LEGO designs for evaluating APEX-MR. The number in each figure indicates the number of objects required to assemble the LEGO structure.

on generating the physically valid assembly sequence are provided in appendix A. Note that the definition of a physically valid assembly sequence can be different for other cooperative assembly tasks. For other applications, the assembly sequence can be obtained via planners, *e.g.*, [58], and the proposed APEX-MR is also applicable downstream.

Manipulation Skills Manipulating LEGO bricks is a non-trivial contact-rich manipulation problem beyond simple pick and stack. A robot EOAT and manipulation policy (*i.e.*, insert-and-twist) were presented in [29], which enable a robot to manipulate commercial standard LEGO bricks, *i.e.*, pick, and place-down in Fig. 1. However, a robot can only use it to manipulate a LEGO brick from its top, which limits the system from constructing complex structures. To enhance the system capability, we present a new LEGO tool (LT-V2), as shown in Fig. 5(c). In particular, LT-V2 has LEGO studs added to the side of the tooltip. The new design enables the robot to manipulate a brick from its bottom as shown in Fig. 1, *i.e.*, handover and place-up. With LT-V2, we define the manipulation skills as shown in Fig. 1, including 1) goal reaching with force feedback (*i.e.*, support-bottom and support-top), and 2) learned force policy (*i.e.*, pick, place-down, place-up, handover). More details on LEGO manipulation skills are discussed in appendix C.

Implementation We implement the TPG algorithm and manipulation skills in C++ with ROS-Noetic and *MoveIt* [7]. The ILP in task planning is solved with the pulp Python package. The RRT-Connect motion planning uses *MoveIt*'s OMPL [57] plug-in. Paths are discretized using $\Delta t = 0.05$ seconds when the maximum L_1 joint velocity is 1 rad/s. Δt is adjusted linearly based on the maximum velocity to ensure the

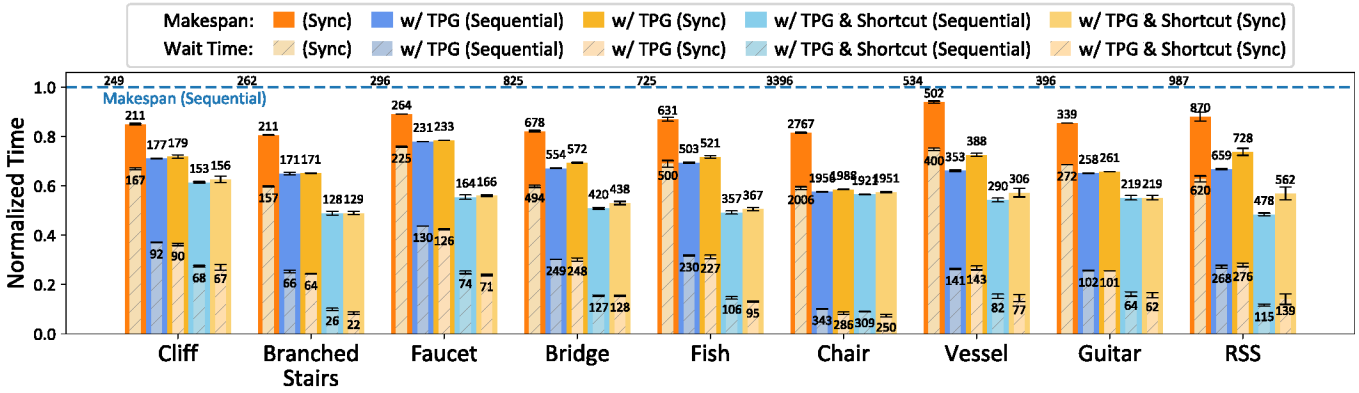


Fig. 7: Normalized makespan and wait time of APEX-MR (Sequential) versus the synchronized planner across example evaluation environment. The results are normalized by the makespan of the sequential motion plan before TPG and averaged over 4 random seeds. The unnormalized makespan and wait time in seconds are labeled for each entry, and the dashed horizontal line corresponds to the makespan of the sequential motion plan before TPG. Shortcut refers to the anytime shortcutting algorithm on TPG in Sec. IV-C

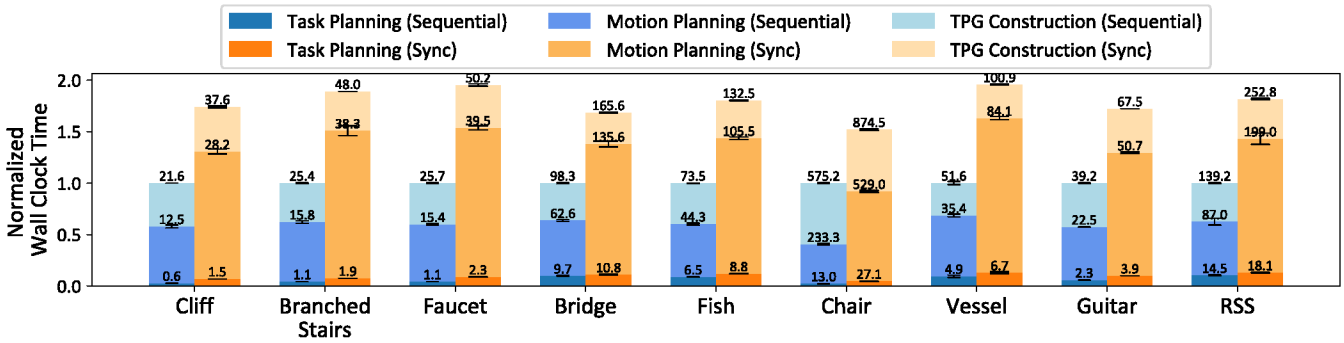


Fig. 8: Breakdown of wall clock time of APEX-MR (Sequential) and the synchronized planner baseline across evaluation environments. The results are normalized by the total of APEX-MR planning time for each task (excluding shortcutting) and averaged over 4 random seeds. The running sum of unnormalized wall clock time in seconds is labeled for each component.

same density. All simulation experiments are conducted on an AMD 7840HS laptop. Algo. 1 parallelizes collision checking with 16 threads.

Experiment Objective While APEX-MR itself is a full pipeline for multi-robot tasks and motion planning, the key innovation that enables asynchronous collaboration is the TPG execution framework. Thus, we are interested in the following questions when evaluating APEX-MR:

- (Q1) How significant is the benefit of asynchronous execution, enabled by the TPG execution framework?
- (Q2) How is the quality of plans produced by APEX-MR and what are the computational costs?
- (Q3) How well does APEX-MR perform in physical LEGO assembly, and can it safely execute planned paths despite uncertainties?

Q1 and Q2 will be closely examined in simulation, whereas Q3 will be the focus of our real robot experiments.

A. *Simulation Performance*

We first conduct experiments in simulation. To our knowledge, no existing simulator can reliably simulate the connections between LEGO bricks. Thus, all robot skills are reduced to deterministic operations when evaluated in simulation, and any variations are due to the stochasticity of planning in APEX-MR.

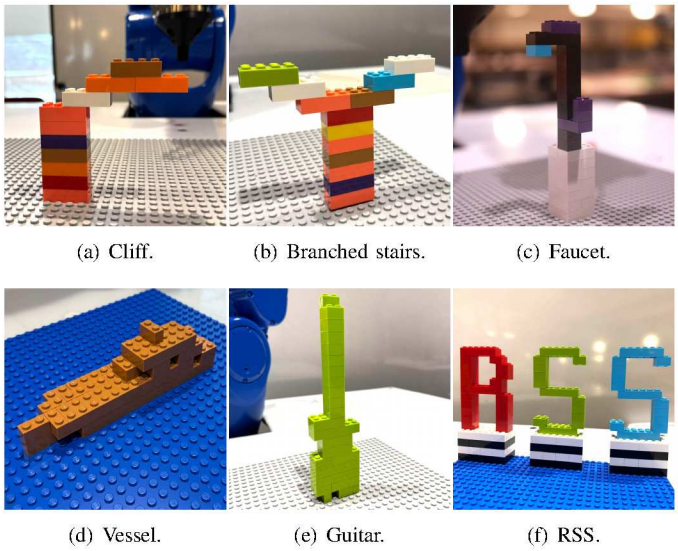
Dataset We evaluate the performance of APEX-MR on a suite of nine LEGO assembly tasks as shown in Fig. 6. The complexity of these tasks varies significantly in terms of the number of objects in the assembly plan, stability, orientation, and manipulation skills required for physical assembly. The Chair shown in Fig. 6(f) has 258 objects, but the structure is solid and stable, and thus, no collaborative skills are needed. On the other hand, many of the bricks along the span of the Bridge (Fig. 6(d)) require a robot to support them when assembling from the top, whereas building the Cliff (Fig. 6(a)) and Faucet (Fig. 6(c)) requires an object reorientation and collaborative assembly from bottom.

Metrics We use execution makespan and wait time as our

evaluation metrics for plan quality. Since our output is a TPG, we first roll out the asynchronous path from the TPG, assuming no controller delay. The rollout path converts each pose node back to a configuration. Actions in the skill nodes are executed based on the reference path, and force feedback is switched off in the simulation. The timestamp for each configuration in the rollout path is the earliest possible time to reach this node based on incoming type-2 edges. Execution makespan is the maximum time taken among all robots to execute their rollout paths, *i.e.*, $\max_{i=1}^{N} t^i_{N^i_{end}}$. Wait time is defined as the total amount of time any robot spends waiting in the rollout asynchronous path from TPG, or the original sequential or synchronous path. In particular, we are interested in whether TPG processing can successfully reduce wait time when initialized with a sequential task and motion plan.

Baseline We also design a baseline for synchronized task and motion planning as a comparison to APEX-MR. Although APEX-MR uses a sequential planner for simplicity and efficiency, our TPG can also improve synchronized motion plans, common in MR-TAMP. Thus, we evaluate the performance improvement of TPG on synchronized plans. In this synchronous planner, robots can execute tasks in parallel but must wait for all robots to finish their current task before proceeding to the next set of tasks. We use an algorithm to convert the sequential task plan from APEX-MR to a synchronous task plan, as shown by Fig. 12 in the Appendix. The main idea is to execute the sequential task plan in parallel if executing the next task does not violate inter-robot task dependencies or block tasks scheduled at an earlier time. This process is similar to building a TPG on tasks instead of motions for parallelization. For every robot i and its task $\mathcal{T}^i_m$, the algorithm checks if the intermediate goal pose $C^i_{N^i_{end,m}}$ collides with any other intermediate goal pose of robot i' and task $\mathcal{T}^{i'}_{m'}$ that satisfies $m' < m$. If there exists a collision, then robot i must wait for robot i' to complete task $\mathcal{T}^{i'}_{m'}$ before starting task $\mathcal{T}^i_m$. A synchronous task graph can then be generated by combining these calculated dependencies with existing inter-robot dependencies. Then, composite RRT-Connect is used as the multi-robot motion planner. Synchronous paths for tasks executing in parallel are generated by planning all degrees of freedom as a single robot.

Performance Fig. 7 shows the quality of the solution of APEX-MR on a variety of tasks. First, the TPG post-processing and applying and shortcut, as described in Sec. IV-C, significantly reduces makespan by 48% and wait time by 85% on average, compared to the initial sequential motion plan on the horizontal dashed line. Compared to the synchronized motion plan, our asynchronous plans from APEX-MR are consistently shorter and have lower wait time. When applied to the synchronous plan, TPG also significantly reduces the makespan by 36% and wait time by 77% on average. Note that the post-processed sequential plan from APEX-MR still slightly outperforms the post-processed synchronized motion plan by 3% in terms of makespan. This is due to the path produced by a multi-robot motion plan being suboptimal compared to sequential motion planning. Still, the wait time for the synchronized plan after TPG post-processing is minimal.



(a) Cliff. (b) Branched stairs. (c) Faucet.

(d) Vessel. (e) Guitar. (f) RSS.

Fig. 9: Example LEGO structures constructed in real by the dual-arm system.

Runtime Fig. 8 and Table II in the appendix shows the wall clock time for APEX-MR. On average, the TPG construction time is always lower than the task motion planning time except for the Chair, which has a very long assembly sequence. On the other hand, running a synchronized planner can be much more expensive than the simple sequential planner used in APEX-MR, which requires more careful coordination in task planning and multi-robot motion planning with more degrees of freedom. By combining a simple sequential task and motion planner with TPG post-processing, APEX-MR produces higher-quality multi-robot plans with 26% lower computational overhead on average than a synchronized multi-robot task and motion planner alone. One concern is that the number of collision checks when building TPG scales quadratically with the number of robots. Thus, we perform additional experiments in Appendix D with three and four robot arms to show that the runtime of TPG construction remains reasonable.

B. System Deployment

We deploy the proposed APEX-MR to a real bimanual setup for cooperative LEGO assembly. Fig. 5(b) illustrates the environment setup of the dual-arm system. Note that despite the environment being pre-calibrated, errors ($\sim$ 1mm) still exist since the calibration is imperfect and the structure could be tilted due to the passive connection nature. Thus, we integrate real-time force feedback using the FTS to improve the manipulation robustness. For operation skills (*i.e.*, pick, place-down, place-up, handover), we use the force feedback to detect successful insertion and update the manipulation policy accordingly. For supporting skills (*i.e.*, support-bottom and support-top), we use force feedback to sense a slight touch

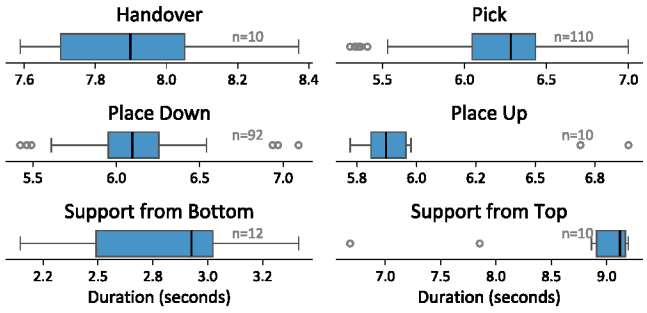


Fig. 10: Distribution of executing various FTS-feedback-controlled LEGO manipulation policy on real robot. The execution time is collected from assembling the Cliff structure in Fig. 9(a) repeatedly. This variation adds to the uncertainty in real-time execution.

with the structure to avoid either over or under-supporting. Fig. 9 showcases example LEGO structures accomplished by the bimanual system with APEX-MR. The robots can safely collaborate and efficiently build customized and complex LEGO objects, including fragile overhanging structures.

Note that the key difference between assembling in real and in simulation is with and without force feedback. Integrating force feedback into the manipulation skills improves the system robustness, but also brings uncertainty with respect to execution time. In particular, we are interested in whether the TPG execution framework ensures safe execution and avoids collisions despite uncertainties of manipulation skills. Fig. 10 depicts the distribution of execution times of six manipulation skills, *i.e.*, pick, place-down, place-up, handover, support-bottom, and support-top (see Fig. 1 for illustrations). All of these skills are designed to execute with force feedback to ensure proper contact between the EOAT and the object being manipulated and minimize the effect of imperfect calibration or a tilted structure. As a consequence of these mm-level adjustments with force feedback, the execution time can vary as much as 2 seconds, or as much as 23% from the median. Nevertheless, our TPG execution framework can reliably adjust the robot schedule if any delay could cause a collision or require another robot to wait longer until a skill is completed.

In practice, APEX-MR also allows the two robot arms to operate in close proximity asynchronously thanks to the use of TPG. For example, if one robot is stopped due to a controller issue or because the user presses emergency stop, the other robot would automatically stop if it is unsafe to continue its execution. Although each robot's feedback controller operates independently, only those actions that are deemed safe are passed to the robot's action queue. Thus, APEX-MR can significantly reduce the risk of unsafe action in real multi-robot execution.

VI. Limitations and Future Work

Although the proposed APEX-MR pipeline enables efficient and safe execution for multiple robot arms, it still has several algorithmic limitations which we discuss below.

Offline Computation Currently, both the TPG processing and motion planning in APEX-MR are performed offline before real execution. This can be a drawback in real assembly tasks, where new tasks are continuously assigned once the robots finish an existing assembly on an assembly line. Another limitation of offline computation is that the APEX-MR cannot easily adapt to changes in the collision environment or assembly steps. A principled framework to address these lifelong planning techniques is to use a windowed multi-robot planner and only convert the first n steps of the robot plan to a TPG, similar to how Varambally et al. [59] address mobile robot coordination in automated warehousing. Taking a reduced-horizon approach will significantly reduce planning time and allow plans to be continuously updated concurrently with execution.

Planning for Robot Dynamics While APEX-MR are reliable and safe on real robots, APEX-MR requires a good position or force controller for the robot because the generated plan does not consider robot dynamics, such as acceleration and jerk constraints. This is challenging because the planner must generate continuous velocity and acceleration profiles while also avoiding inter-robot collisions at all times. We plan to incorporate dynamics as part of the TPG post-processing step, such as solving a linear program on top of TPG-imposed constraints and velocity constraints, as suggested in Hoenig et al. [19]. Another interesting problem with robot arms is that speed constraints may be imposed on the task space, due to the attached object at the end-effector or even closed kinematics chains formed by concurrent manipulation [60].

Manipulation Policy While the proposed APEX-MR enables efficient and safe dual-arm cooperative LEGO assembly, the current system assembles each object based on predefined skills. With the additional force feedback, each single operation can be performed robustly. However, due to the passive connection nature of LEGO structures, *i.e.*, established connections could be gradually loosened and the structure can be tilted or even collapse due to subsequent operations, the long-horizon assembly could still fail. Therefore, the dual-arm system, at its current stage, is not robust enough to construct large-scale LEGO structures that have multiple fragile overhanging geometries. To further improve the robustness from a system perspective, our aim is to investigate methods for failure detection [4] and recovery, *e.g.*, reinforcing the connections that are loosened due to later operations. Failure detection and recovery can also be integrated as part of an online replanning framework that dynamically reschedules the robot's tasks if a failure occurs and intervention becomes necessary.

Other Cooperative Assemblies Building on the APEX-MR pipeline, we plan to extend its application to a broader range of cooperative tasks, such as the NIST Box Assembly [40], and other industrial assembly scenarios. In doing so, we aim to address the unique challenges posed by real-world manufacturing environments, gaining deeper insights into how cooperative systems can be optimized for complex, large-scale production tasks. While the discussion in this paper

is based on LEGO assembly, the components in APEX-MR, especially TPG post-processing, can be applied to other multi-robot assembly tasks. A concrete step would be to investigate how to integrate manipulation policies that are more complex than those used for LEGO assembly, *e.g.*, diffusion policy [6], to the TPG framework.

VII. Conclusion

For many robotic manipulation tasks, a team of cooperative robot arms is often necessary and beneficial because cooperation can improve dexterity, flexibility, and versatility. A reliable framework for coordinating robot arms should possess several key qualities: efficiency to maximize throughput, scalability to long-horizon and complex tasks, and safety during real execution.

With these criteria in mind, we have proposed APEX-MR, a pipeline for multi-robot asynchronous planning and execution. Our proposed pipeline combines a sequential task and motion planner with a TPG to post-process the plan for asynchronous execution. Specifically, TPG post-processing can significantly speed up the execution of otherwise sequential and synchronous multi-robot task plans by 48% and 36% on our simulated assembly tasks. Because coordination is easier, sequential motion planning is far more efficient than planning for synchronous execution.

We demonstrated that our proposed algorithm can be successfully deployed and integrated for a real bimanual cooperative task. LEGO structures, as an example, presented a challenging manipulation task due to the need for high precision and the non-rigid nature of their connections. We presented a set of manipulation skills for complex cooperative assembly, including supported placement and object handover, based on the end-effector design LT-V2. The dual-arm system successfully performs customized LEGO assembly and is the *first* robotic system to do so with commercial LEGO bricks. Additionally, we showed that the integration with the TPG execution framework is robust in the presence of uncertain execution time. In the end, we hope that this framework can advance and bring closer to more real use of multi-robot arm collaboration algorithms.

Acknowledgments

This work is in part supported by the National Science Foundation (NSF) under grant number 2328671 and the Manufacturing Futures Institute, Carnegie Mellon University, through a grant from the Richard King Mellon Foundation, as well as a gift from Amazon. The authors also thank Yifan Sun for helping design LT-V2.

References

[1] Alexander Berndt, Niels Van Duijkeren, Luigi Palmieri, and Tamas Keviczky. A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph. *arXiv 2010.05254*, October 2020.

[2] Z Bien and J Lee. A minimum-time trajectory planning method for two robots. *IEEE Transactions on Robotics and Automation*, 8(3):414–418, June 1992.

[3] Andrea Brunello, Giuliano Fabris, Alessandro Gasparetto, Angelo Montanari, Nicola Saccomanno, and Lorenzo Scalera. A survey on recent trends in robotics and artificial intelligence in the furniture industry. *Robotics and Computer-Integrated Manufacturing*, 93: 102920, 2025.

[4] Hongyi Chen, Yunchao Yao, Ruixuan Liu, Changliu Liu, and Jeffrey Ichnowski. Automating robot failure recovery using vision-language models with optimized prompts. *arXiv:2409.03966*, 2024.

[5] 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. *arXiv:2203.02475*, 2022.

[6] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[7] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a *MoveIt* case study. *arXiv:1404.3785*, 2014.

[8] Michael Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(1-4):477–521, November 1987.

[9] Yongxiang Fan, Jieliang Luo, and Masayoshi Tomizuka. A learning framework for high precision industrial assembly. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 811–817, 2019.

[10] Nigora Gafur, Gajanan Kanagalingam, Achim Wagner, and Martin Ruskowski. Dynamic collision and deadlock avoidance for multiple robotic manipulators. *IEEE Access*, 10:55766–55781, 2022.

[11] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. Batch informed trees (BIT*): Informed asymptotically optimal anytime search. *International Journal of Robotics Research*, 39(5):543–567, April 2020.

[12] Kai Gao and Jingjin Yu. Toward efficient task planning for dual-arm tabletop object rearrangement. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10425–10431, 2022.

[13] Kai Gao, Zihe Ye, Duo Zhang, Baichuan Huang, and Jingjin Yu. Toward holistic planning and control optimization for dual-arm rearrangement. *aXiv 2404.06758*, 2024.

[14] Kai Gao, Zhaxizhuoma, Yan Ding, Shiqi Zhang, and Jingjin Yu. Orla*: Mobile manipulator-based object rearrangement with lazy a star. *arXiv 2309.13707*, 2024.

[15] Mokhtar Gharbi, Juan Cortés, and Thierry Siméon.

Roadmap composition for multi-arm systems path planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2471–2476, October 2009.

[16] Kieran Gilday, Josie Hughes, and Fumiya Iida. Achieving flexible assembly using autonomous robotic systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[17] Kensuke Harada, Tokuo Tsuji, and Jean-Paul Laumond. A manipulation motion planner for dual-arm industrial manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 928–934, May 2014.

[18] Valentin N. Hartmann, Andreas Orthey, Danny Driess, Ozgur S. Oguz, and Marc Toussaint. Long-horizon multi-robot rearrangement planning for construction assembly. *IEEE Transactions on Robotics*, 39(1):239–252, 2023.

[19] Wolfgang Hoenig, T. K. Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 26, pages 477–485, Mar. 2016.

[20] Wolfgang Honig, Scott Kiesel, Andrew Tinka, Joseph W Durham, and Nora Ayanian. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters*, 4(2):1125–1131, April 2019.

[21] Wolfgang Hönig, James A Preiss, T K Satish Kumar, Gaurav S Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, August 2018.

[22] Daqi Jiang, Hong Wang, and Yanzheng Lu. Mastering the complex assembly task with a dual-arm robot: A novel reinforcement learning method. *IEEE Robotics and Automation Magazine*, 30(2):57–66, June 2023.

[23] He Jiang, Muhan Lin, and Jiaoyang Li. Speedup techniques for switchable temporal plan graph optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[24] Y Koga and J-C Latombe. On multi-arm manipulation planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 945–952 vol.2, 1994.

[25] James J. Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 995–1001, 2000.

[26] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. EECBS: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 12353–12362, 2021.

[27] Ruixuan Liu, Alan Chen, Xusheng Luo, and Changliu Liu. Simulation-aided learning from demonstration for robotic lego construction. *arXiv:2309.11010*, 2023.

[28] Ruixuan Liu, Kangle Deng, Ziwei Wang, and Changliu Liu. Stablelego: Stability analysis of block stacking assembly. *IEEE Robotics and Automation Letters*, 9(11): 9383–9390, 2024.

[29] Ruixuan Liu, Yifan Sun, and Changliu Liu. A lightweight and transferable design for robust lego manipulation. In *International Symposium on Flexible Automation*, page V001T07A004, 07 2024.

[30] Ruixuan Liu, Alan Chen, Weiye Zhao, and Changliu Liu. Physics-aware combinatorial assembly sequence planning using data-free action masking. *IEEE Robotics and Automation Letters*, 10(5):4882–4889, 2025.

[31] Xusheng Luo, Shaojun Xu, Ruixuan Liu, and Changliu Liu. Decomposition-based hierarchical task allocation and planning for multi-robots under hierarchical temporal logic specifications. *IEEE Robotics and Automation Letters*, 9(8):7182–7189, 2024.

[32] Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, page 3605–3612, 2017.

[33] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 837–845, 2017.

[34] Yusuke Maeda, Ojiro Nakano, Takashi Maekawa, and Shoji Maruo. From CAD models to toy brick sculptures: A 3D block printer. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2167–2172, 2016.

[35] Tobia Marcucci, Jack Umenberger, Pablo A. Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[36] Charles A Meehan, Mark Roberts, and Laura M Hiatt. Asynchronous motion planning and execution for a dual-arm robot. In *Workshop on Planning and Robotics held in conjunction with the International Conference on Automated Planning and Scheduling (ICAPS)*, Virtual, June 2022.

[37] G. Michalos, S. Makris, N. Papakostas, D. Mourtzis, and G. Chryssolouris. Automotive assembly technologies review: challenges and outlook for a flexible and adaptive approach. *CIRP Journal of Manufacturing Science and Technology*, 2(2):81–91, 2010.

[38] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for coordinated multi-arm motion planning. *International Journal of Robotics Research*, 37(10):1205–1232, September 2018.

[39] Ludwig Nägele, Alwin Hoffmann, Andreas Schierl, and Wolfgang Reif. Legobot: Automated planning for coordinated multi-robot assembly of lego structures. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9088–9095, 2020.

[40] National Institute of Standards and Technology. STEP

File Viewer - Box assembly, 2024. URL https://pages.nist.gov/step-file-viewer.html

[41] Keisuke Okumura and Xavier Défago. Quick multi-robot motion planning by combining sampling and search. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

[42] Tianyang Pan, Andrew M Wells, Rahul Shome, and Lydia E Kavraki. A general task and motion planning framework for multiple manipulators. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3168–3174, September 2021.

[43] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv 1704.03073*, 2017.

[44] Yorai Shaoul, Itamar Mishani, Maxim Likhachev, and Jiaoyang Li. Accelerating search-based planning for multi-robot manipulation by leveraging online-generated experiences. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2024.

[45] Yorai Shaoul, Rishi Veerapaneni, Maxim Likhachev, and Jiaoyang Li. Unconstraining multi-robot manipulation: Enabling arbitrary constraints in ecbs with bounded sub-optimality. In *Proceedings of International Symposium on Combinatorial Search (SoCS)*, 2024.

[46] Rahul Shome and Kostas E. Bekris. Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs. In *Proceedings of International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 37–43, 2019.

[47] Rahul Shome and Kostas E. Bekris. Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints. In *Algorithmic Foundations of Robotics XIV*, pages 243–260, Cham, 2021. Springer International Publishing.

[48] Rahul Shome, Kiril Solovey, Andrew Dobson, Dan Halperin, and Kostas E Bekris. dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning. *Autononomus Robots*, 44(3):443–467, March 2020.

[49] Rahul Shome, Kiril Solovey, Jingjin Yu, Kostas Bekris, and Dan Halperin. Fast, high-quality two-arm rearrangement in synchronous, monotone tabletop setups. *IEEE Transactions on Automation Science and Engineering*, 18 (3):888–901, 2021.

[50] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V. Dimarogonas, and Danica Kragic. Dual arm manipulation – A survey. *Robotics and Autonomous Systems*, 60(10):1340–1353, 2012.

[51] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V Dimarogonas, and Danica Kragic. Dual arm manipulation—a survey. *Robotics and Autonomous Systems*, 60(10):1340–1353, October 2012.

[52] Irving Solis, James Motes, Read Sandström, and Nancy M Amato. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics and Automation Letters*, 6(3):4608–4615, July 2021.

[53] Kiril Solovey, Oren Salzman, and Dan Halperin. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *International Journal of Robotics Research*, 35 (5):501–513, April 2016.

[54] 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. In *Proceedings of the Twelfth Annual Symposium on Combinatorial Search (SoCS)*, pages 151–159, 2019.

[55] Pascal Stoop, Tharaka Ratnayake, and Giovanni Toffetti. A method for multi-robot asynchronous trajectory execution in *MoveIt2*. *arXiv 2310.08597*, 2023.

[56] Yifan Su, Rishi Veerapaneni, and Jiaoyang Li. Bidirectional temporal plan graph: enabling switchable passing orders for more efficient multi-agent path finding plan execution. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

[57] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. The open motion planning library. *IEEE Robotics and Automation Magazine*, 19(4):72–82, 2012.

[58] Yunsheng Tian, Jie Xu, Yichen Li, Jieliang Luo, Shinjiro Sueda, Hui Li, Karl D. D. Willis, and Wojciech Matusik. Assemble them all: Physics-based planning for generalizable assembly by disassembly. *ACM Transactions on Graphics*, 41(6):278:1–278:11, 2022.

[59] Sumanth Varambally, Jiaoyang Li, and Sven Koenig. Which MAPF model works best for automated warehousing? In *Proceedings of International Symposium on Combinatorial Search (SoCS)*, volume 15, pages 190–198, July 2022.

[60] Zhou Xian, Puttichai Lertkultanon, and Quang-Cuong Pham. Closed-chain manipulation of large objects by multi-arm robotic systems. *IEEE Robotics and Automation Letters*, 2(4):1832–1839, October 2017.

[61] Shiyu Zhang and Federico Pecora. Online and scalable motion coordination for multiple robot manipulators in shared workspaces. *IEEE Transactions on Automation Science and Engineering*, PP(99):1–20, 2024.