2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

# Multi-Robot Task Allocation Under Uncertainty Via Hindsight Optimization

Neel Dhanaraj[1], Jeon Ho Kang[1], Anirban Mukherjee[1], Heramb Nemlekar[1], Stefanos Nikolaidis[1], and Satyandra K. Gupta[1]

***Abstract*— Multi-robot systems are becoming increasingly prevalent in various real-world applications, such as manufacturing and warehouse logistics. These systems face complex challenges in 1) task allocation due to factors like time-extended tasks, and agent specialization, and 2) uncertainties in task execution. Potential task failures can add further contingency tasks to recover from the failure, thereby causing delays. This paper addresses the problem of Multi-Robot Task Allocation under Uncertainty by proposing a hierarchical approach that decouples the problem into two levels. We use a low-level optimization formulation to find the optimal solution for a deterministic multi-robot task allocation problem with known task outcomes. The higher-level search intelligently generates more likely combinations of failures and calls the inner-level search repeatedly to find the optimal task allocation sequence, given the known outcomes. We validate our results in simulation for a manufacturing domain and demonstrate that our method can reduce the effect of potential delays from contingencies. We show that our algorithm is computationally efficient while improving average makespan compared to other baselines.**

## I. Introduction

Multi-robot systems are increasingly being deployed in real-world applications, requiring effective agent task allocation approaches. In manufacturing, teams of manipulators and mobile robots accomplish complex assembly operations in production lines and collaborative high-mix, low-volume cells [1], [2]. In warehouse logistics, mobile robots are tasked with retrieving and carrying different items while dynamically accounting for new requests [3]. For such applications, task allocation approaches must account for time-extended tasks, complex task constraints, spatio-temporal constraints, agent specialization, and availability [4]. In practice, multi-robot task allocation (MRTA) can be difficult due to the uncertainty associated with task execution and robot availability. Robot execution can lead to task failures. These failures may also create new contingency tasks to address the failure [5]. Furthermore, robots may fail tasks early or become delayed in completing tasks. These challenges require planning over time horizons and reasoning over possible alternative futures.

A task allocation problem in multi-robot team settings in the presence of such outcomes requires us to consider the effect of two different combinatorial effects. The first effect comes from many alternative ways of assigning robots to tasks. The second effect comes from many different potential outcomes due to uncertainties in task execution durations and task execution failure possibilities. Conceptually, this can be modeled as a joint problem involving planning under uncertainty and coordinating multi-agent allocations (e.g., a tree search with two different types of branching at each level). While this joint formulation is conceptually easy to understand and visualize, it is computationally intractable when dealing with non-trivial problem sizes and real-world constraints.

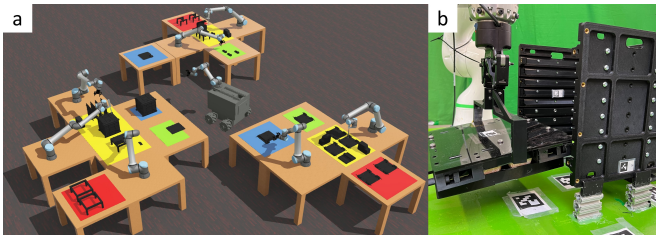

Fig. 1: (*a*) A representative example that motivates our approach. An 8-robot fixtureless assembly cell is used in high-mix, low-volume satellite manufacturing. Robots assemble sub-components such as batteries in parallel. These parts must then be installed in certain sequences. There is further uncertainty in task completion due to the fixtureless nature of the cell and complex insertion tasks.(*b*) An example of a contingency where an insertion task has failed.

Prior work by Choudhury et al., [6] noted that to solve this problem computationally efficiently, one can decouple the problem and consider a two-level hierarchical approach. The lower level reasons about uncertainty in the task completion of an individual agent. The upper level then coordinates task allocation between all agents. This method works well when inefficiencies come from robots wasting time on failed tasks and missing opportunities to complete other tasks (see Section II for details).

However, for a different class of application domains, if coordination does not consider the effect of uncertainty on the team, inefficiencies can stem from possible bottlenecks due to coupled agent constraints, task constraints, and contingencies caused by failed tasks. There exists an alternative way of constructing a two-level hierarchy to address these problems. *1) The lower level coordinates multi-agent decisions, and 2) the upper level reasons about uncertainty in task completion.* Our prior work has shown a preliminary version of such an alternative: a one-step lookahead algorithm with optimistic rollouts generated using mixed integer programs, which we validated on a specialized formulation for human-robot teams [**?**]. These rollouts were augmented by sampling failure contingencies at the higher level and executing more rollouts from these contingency states. This approach enabled us to select task allocations with good options to recover from failures.

Our previous work only considered how to recover from task failures and selected solutions that had low recovery

[1]Viterbi School of Engineering, University of Southern California, CA USA {dhanaraj, jeonhoka, am28141, nemlekar, nikolaid, guptask}@usc.edu

costs. In this paper, we extend the previous approach to utilize the low-level search to consider how to mitigate the impact of task failures and/or prevent the failures during task allocations.

Consider a manipulator performing an insertion task. This task can fail due to high uncertainty in the slot location (see Figure 1). There are three ways to reason about this potential contingency. If the task were to fail, the robot could plan to attempt the task again by waiting for the mobile agent to perform the high-resolution imaging to reduce uncertainty (*recovery*). The robot can also consider doing the task earlier or later when the mobile agent will more likely be available to perform imaging (*mitigation*). Lastly, the task can be assigned to a different robotic station that has close-up imaging capability (*prevention*). Our key insight is that we can quickly find good task allocations by decoupling the joint problem and explicitly reasoning over task failure recovery, mitigation, and prevention.

We propose a new method that takes inspiration from hindsight optimization [7]. We use a low-level optimization formulation to find the optimal solution for a deterministic multi-robot task allocation problem with known task outcomes. The higher-level search intelligently generates more likely combinations of failures and calls the inner-level search repeatedly to find the optimal task allocation sequence, given the outcomes were known. We merge these task allocation sequences for different sampled futures to create a search tree. Our main contributions are: (1) a general formulation for multi-robot task allocation under uncertainty, task constraints, and temporal constraints, (2) a hierarchical algorithm that performs hindsight optimization via MIP solvers and intelligent sampling of potential futures, and (3) numerical studies that demonstrate the success of the proposed method.

## II. Related Works

**Multi-Robot Task Allocation:** Multi-robot task allocation has been studied extensively across different fields, with numerous works surveying problem variants and approaches to solve them [4], [8]–[10]. From the taxonomy of Gerkey et al., [11] the deterministic version of our problem is a *ST-SR-TA (Single-Task Robots, Single-Robot Tasks, Time-Extended Assignment)*. Such problems require constructing a schedule of tasks for each robot, making them NP-Hard. Korsah et al. [12] has extended the previous taxonomy by explicitly considering task constraint dependencies. Our problem can be described as having *CD (Complex Dependencies)*. Informally, for *CD* class of problems, the effective utilization of an agent's schedule depends on other agents' schedules. The uncertainty, coupled with task constraints inherently makes our problem **CD[ST-SR-TA]**. Such problems are strongly NP-Hard and, in general, have motivated research to develop specialized methods to make the problem tractable [13]–[15].

**Task Assignment and Scheduling:** Multi-robot task allocation is closely related to well-studied problems, such as job shop scheduling [16], [17], vehicle routing [18]–[20], and general operations research [21]. Closely related to our work, job shop scheduling models will assign jobs to machines to minimize the overall makespan for a set of durative tasks while often considering resource constraints and task constraints. These communities have driven advancements in mixed integer linear programs and constraint program solvers in order to solve these problems more efficiently. The robotics community is now utilizing the idea that, theoretically, robots can be considered machines or vehicles, and tasks are jobs or nodes [4]. This has enabled researchers to fully exploit mixed-integer linear programs for solving complex robot task assignments and scheduling while considering temporal and spatial constraints [22]–[24] as well as hierarchical task networks and precedence constraints [25]–[30]. This body of work is the backbone for the lower level of our method.

**Multi-Agent Sequential Decision Making:** Typically, multi-agent sequential decision-making is mathematically modeled as a Multi-Agent Markov Decision Process (MMDP) [31]. Our problem is a centralized MMDP with full knowledge and a shared objective. Typical Markov Decision Process approaches are generally infeasible for solving an MMDP due to the exponential joint action space and state space. Research instead has delved toward reinforcement learning techniques to learn the values of different states and actions [32] and online planning approaches to explore the reachable state space from a current state [33]. We build upon recent work that has shown that the challenges of multi-robot task allocation under uncertainty can be decoupled [6]. Specifically, the authors propose to compute individual agent task allocation policies at the lower level and perform multi-agent coordination at the upper level via conflict resolution of agent task allocations. We observe that this method works well for a class of domains where only robot-task assignment coordination is required, and inefficiency stems from spatio-temporal relationships coupled with uncertainty between each agent and its tasks, which, if not effectively considered, results in a robot's wasted time. Our work is new in that we show an alternative method to decouple the problem and its strength on another class of problems.

## III. Multi-Robot Team Task Allocation Under Uncertainty Problem Formulation

We present a general formulation for a multi-robot task allocation under uncertainty (MRTAU). For the MRTA aspect of the problem, there is a set of $n$ agents $\mathcal{A}_i \in \mathbf{N}$ and a set of $m$ tasks to be completed $\tau_j \in \mathcal{T}$. Each agent has a state $s_{\mathcal{A}_i}$ consisting of *idle* or *busy*. There exists a mapping denoting each agent $\mathcal{A}_i$ belongs to an agent type $\mathcal{A}^x$. Each task has a state $s_{\tau_j}$ consisting of *available*, *inprogress*, *completed* or *failed*. Each task must be executed by an agent type, e.g., a screwing task must be completed by a screwing robot. Lastly, an agent executing an assigned task will incur a durative task execution cost $\mathrm{c_e}(\mathcal{A}_i, \tau_j)$.

An MRTAU problem has a set of constraints $c \in \mathcal{C}$. First, we constrain the problem such that each task must be executed by 1 agent, and each agent can only attempt one task at a time. There also exists a set of ordering constraints that dictate task dependencies. Such constraints

may dictate partial task orderings or that specific tasks cannot be performed concurrently. We use a hierarchical task network to specify these constraints as formulated in [30], where the leaf nodes are atomic tasks and all other nodes specify ordering constraints for subassemblies.

For each agent type and task, there is a task model describing the task's uncertainty dynamics. An agent may fail to complete a task during execution, e.g., an insertion robot has broken an assembly part. The task success uncertainty function $P(\mathcal{A}^x, \tau_j)$ is represented as $\text{Prob}(\mathcal{A}^x \text{ completes } \tau_j)$. For each agent type and task, there is also a duration model associated with how the task succeeds or fails, e.g., a screwing task attempted by the screwing robot may either fail early in the process or succeed on time.

Lastly, if a task fails, new contingency tasks with associated constraints may need to be added to $\mathcal{T}$. For example, if a robot breaks a part, then the tasks "replace part" and "retry operation" will be added to $\mathcal{T}$. We represent this as a contingency function that returns a new set of tasks and hierarchical task network: $\mathcal{T}' = \text{T}_{\text{cont}}(s_{\tau_j})$. The contingency function is domain/task-specific and would be engineered beforehand.

The goal for solving this type of MRTAU problem is to minimize the expected overall time or makespan to complete all tasks. However, because the state and action space of the problem will lead to an intractable state space, we formulate an online planning approach where we interleave planning and execution.

We first formally express the problem as a Multi-Agent Markov Decision Process (MMDP), which is defined as a tuple $\mathcal{M} = (N, S, A, T, C)$. We use the state and action representation to form the nodes in the higher-level search tree, as well as to store the transition function.

1) $N = \{\mathcal{A}_1, ..., \mathcal{A}_i\}$.
2) $s \in S$ is the state of the entire system consisting of a factored representation of all task states and the agents states $s = (\{s_{\tau_1} ..., s_{\tau_m}\}, \{s_{\mathcal{A}_1} ..., s_{\mathcal{A}_n}\})$. Furthermore, each state is a decision epoch where a new task becomes available for agents, or an agent finishes a task and is ready for a new task assignment.
3) $a \in A$ is a joint action consisting of a set of agent actions $a = \{a_{\mathcal{A}_1}, ..., a_{\mathcal{A}_n}\}$. Each action is either a task assignment $a_{\mathcal{A}_j} = \mathcal{A}_i \leftrightarrow \tau_j$ or a no-operation action for an agent $a_{\mathcal{A}_i} = No-Op$.
4) $T(s, a, s')$ is the joint transition probability function. Consider the joint action $a = \{a_{\mathcal{A}_1}, a_{\mathcal{A}_2}\}$ where $a_{\mathcal{A}_1} = \mathcal{A}_1 \leftrightarrow \tau_k$ and $a_{\mathcal{A}_2} = \mathcal{A}_2 \leftrightarrow \tau_l$. Then the joint transition probability function is $T(s, a, s') = T(s_{\tau_k}, a_{\mathcal{A}_1}, s'_{\tau_k}) * T(s_{\tau_l}, a_{\mathcal{A}_2}, s'_{\tau_l})$.
5) $C(s_i)$: Is the accumulated time taken to reach $s_i$ from $s_0$.

**Overview of Approach:** We develop a hierarchical approach inspired by hindsight optimization that minimizes the expected makespan for completing a set of tasks. At the low level, we address the challenge of multi-agent coordination by assuming deterministic dynamics and computing good task assignments and schedules. At the high level, we address the challenge of sequential decision-making under uncertainty by sampling likely alternative failure states the system can transition into when executing its task schedule and compute new task allocation sequences that recover, mitigate, and prevent these failures. Lastly, these sequences are merged into a search tree, the cost accumulated at the leaf nodes is backpropagated to the root, and the lowest cost action is executed.

## IV. Low-Level: Deterministic Multi-Robot Task Allocation

To decouple multi-agent coordination from decision-making under uncertainty, we first develop a method to generate a task assignment and schedule from an initial state to a goal state. We do this by considering a simpler *determinized* problem $\mathcal{M}_d$, where task allocation outcomes are deterministic. This allows us to solve a classical MRTA problem. We formulate the low-level optimization as follows: given the initial system state $s_0$, we want to find the allocation schedule $\pi_d$ that minimizes the objective in Equation 1 while assuming deterministic dynamics given by $\mathcal{M}_d$.

$$\begin{aligned} &\underset{\pi_d \in \Pi_d}{\text{minimize}} [C(s_g)] \\ &\text{s.t} \, c \in \mathcal{C} \end{aligned} \tag{1}$$

Essentially, we create $\mathcal{M}_d$ by *determinizing* all task - agent assignment outcomes, e.g., if an agent is assigned to a task $a_{\mathcal{A}_j} = \mathcal{A}_j \leftrightarrow \tau_i$, then $s'_{\tau_j}$ will result in *completed*, i.e., $T(s_{\tau_i}, a_{\mathcal{A}_j}, s'_{\tau_j}) = 1$. We generate $\mathcal{M}_d$ by first *determinizing* each task-agent assignment outcome to have a single outcome (*completed* or *failed*). If an outcome is determinized to be *failed*, then the resulting contingency tasks and constraints are added to the $\mathcal{M}_d$. Lastly, we set the task execution duration by an agent to be the expected task duration for the specific determinized outcome.

Many methods have been proposed to solve deterministic MRTA problems. Recent successes in mixed integer linear programming (MILP) and constraint programming (CP) solvers motivated us to reformulate the problem into a mathematical program. This enables us to use available solvers to return solutions quickly. Specifically, we adopt a constraint programming mathematical formulation based on the flexible job shop problem. The goal of the solver $\mathcal{P}$ is to find the best value for the binary agent-task decision variables $x_{\mathcal{A}_i \tau_j}$ for $i$ agents and $j$ tasks and the integer task start times $t^s_{\tau_j}$ that minimizes the task allocation cost while subject to task and temporal constraints. More details regarding the formulation can be found in the following prior work [34]–[36]. Given an initial start state, the solver will return the best decision variable values which gives us $\pi_d$.

## V. High-Level: Sequential Decision Making Under Uncertainty

At the high level, we sample potential failure states that can occur and evaluate the effect of the failure by reasoning over how the system can recover from, mitigate, and/or prevent the failure. Consider that the low-level deterministic task allocation $\pi_d$ is a path sequence where nodes are states and

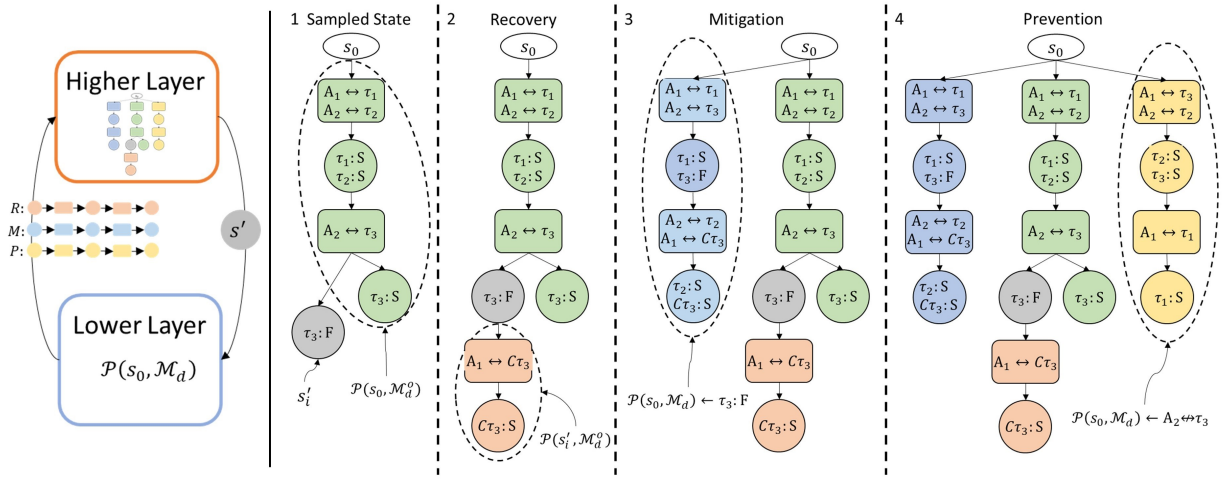


Fig. 2: Here is an illustrative example with two agents $[\mathcal{A}_1, \mathcal{A}_2]$, three tasks $[\tau_1, \tau_2, \tau_3]$, and if $\tau_3$ fails, a contingency task $C\tau_3$. *Sampled State:* The algorithm first generates an initial optimistic state-action sequence (green) connected to $s_0$ and samples alternative states (grey) that could occur from this path. In this example, $\tau_3$, assigned to $\mathcal{A}_2$, fails. *Recovery:* The algorithm first computes a recovery path (red) from the failed task state $s_i'$, where the contingency task is assigned to agent 1. *Mitigation:* Secondly, knowing the task failed in hindsight, the algorithm next creates a new task outcome determinization $\mathcal{M}_d$ and generates a new path from $s_0$ (blue). In this case, knowing $\tau_3$ fails in hindsight, $\tau_3$ is scheduled first so that the contingency task can be addressed earlier. *Prevention:* Lastly, the algorithm passes a constraint into $\mathcal{M}$ preventing $\tau_3$ from being assigned to $\mathcal{A}_2$ and again generates a new path (yellow) that is connected to $s_0$.

actions $s_{0:t_h}, a_{0:t_h-1}$. We then traverse the path and sample other possible states that could occur for each state-action pair $(s_i, a_i)$. For each sampled state $s_i^{'}$, we obtain a new task-agent outcome determinization $\mathcal{M}_d'$, which we input into our deterministic solver $\mathcal{P}$. Given $s_i^{'}$ and $\mathcal{M}_d'$, We propose three mechanisms for reasoning over these potentially occurring alternative states: how to 1) recover from these deviations, 2) mitigate the impact of task failures in hindsight, and 3) prevent agent task failures in hindsight. We illustrate an example of our method in Figure 2.

**Reasoning Over Sampled States**: We propose the following three reasoning mechanisms when considering a sample contingency state:

*Recovery:* Our method first considers the sampled alternative state $s_i'$ as a deviation from the original deterministic trajectory and generates an optimistic sequence from $s_i$ to the horizon.

$$[s_{i:d}, a_{i:d}]_r \leftarrow \mathcal{P}_r(s_i', \mathcal{M}_d^o) \tag{2}$$

*Mitigation:* Secondly, the method reasons over mitigating the impact of task failures in the sampled state by computing a new sequence from $s_0$ given that the task failure outcomes $\tau_i : \textit{failed}$ of the sampled state were known in hindsight.

$$[s_{0:t_h}, a_{0:t_h-1}]_m \leftarrow \mathcal{P}_m(s_i', \mathcal{M}_d') \leftarrow \tau_i : \textit{failed} \tag{3}$$

*Prevention:* Lastly, we consider the effect of preventing the agents from failing tasks by computing a deterministic plan from $s_0$ where the agent that executed the failed task is prevented from that task assignment only $\mathcal{A}_j \nleftrightarrow \tau_i$ if there is another agent alternative.

$$[s_{0:t_h}, a_{0:t_h-1}]_p \leftarrow \mathcal{P}_p(s_i', \mathcal{M}_d') \leftarrow \mathcal{A}_j \nleftrightarrow \tau_i \tag{4}$$

For each sampled state, we call these deterministic planners $[\mathcal{P}_r, \mathcal{P}_m, \mathcal{P}_p]$ to generate three sequences. These sequences are then merged into a search tree.

## VI. Hindsight Optimization Search Tree

We use a method to create an online search tree for multi-agent teams. This involves constructing a search tree using system states $s_i$ and joint actions $a_i$ based on low-level task schedules. An example of this can be seen in Figure 2. Our approach involves three main steps: 1) sampling likely contingency states, 2) generating new state-action node sequences using reasoning mechanisms, and 3) integrating these sequences into the search tree. We then backpropagate the expected cost to the first layer of potential actions once the search is complete. The leaf nodes of the search tree represent makespan values for specific futures. We use Bellman backups, as shown in Equation 5, to backpropagate values from the leaf nodes to the root node. Finally, we return the lowest cost action using $\underset{a_0 \in A_0}{\textit{argmin}} Q(s_0, a_0)$.

$$C(s) = \min_{a \in A} \left[ \frac{\sum_{s'} [T(s,a,s') C(s')]}{\sum_s T(s,a,s')} \right] \tag{5}$$

Algorithm 1 describes our approach. We first instantiate the root of the search tree as the current state $s_0$ (line 5). The algorithm first initializes the search tree by assuming all task outcomes will be successfully *completed*: $\mathcal{M}_d^o$ and generate the first node sequence (line 6). It then merges the sequence to the search tree and samples other possible contingency states along the optimistic sequence that could occur (line 7). By doing this, we begin the search tree optimistically, and as we evaluate contingency states and merge new node sequences, we converge to the expected solution.

In Section V, we discussed how we use three mechanisms to reason over the effect of a contingency on a sampled state. Each mechanism generates a new node sequence, which we then merge into the search tree (Line 13). Figure 2 shows that our algorithm finds alternative good actions to

take at the current state by reasoning over how to prevent and mitigate contingencies. Sampled contingencies from the merged node sequences are stored in the associated action containers (Line 13). Finally, the algorithm selects actions to explore/exploit and evaluates a new contingency from the selected action's container. We repeat this process until the search is terminated, at which point we backpropagate the leaf node cost and return the lowest cost action (Line 14).

**Algorithm 1**

1: $s_0$: Current State
2: $Open$: Set of containers for each action in consideration; each container has sampled contingency states
3: **function** PLAN($s_0, p_{c,min}$)
4: $root \leftarrow s_0$
5: $[s,a] \leftarrow \mathcal{P}_r(s_0)$ ▷ Optimistic
6: $Open \leftarrow$ MERGE($root, [s,a]_r$)
7: **while** Not Interrupted **do**
8: $a_0 \leftarrow$ SELECT-ACTION
9: $s_i \leftarrow open[a_0]$
10: $[s,a]_r \leftarrow \mathcal{P}_r(s_i)$ ▷ Recovery
11: $[s,a]_m \leftarrow \mathcal{P}_m(s_0)$ ▷ Mitigation
12: $[s,a]_p \leftarrow \mathcal{P}_p(s_0)$ ▷ Prevention
13: $Open \leftarrow$ MERGE($root, [s,a]_r, [s,a]_m, [s,a]_p$)
14: BACKPROPOGATE-COST
**return** Lowest Cost Action

## VII. RESULTS

Our proposed approach is primarily evaluated based on the average makespan required to complete a set of assembly tasks. To achieve this, we implement an OR-Tools CP-SAT solver as the lower-level deterministic scheduler. The CP-SAT solver solves a mixed integer program for scheduling, which is similar to our prior work [**?**]. We use a Python implementation on a machine with 32 GB RAM and an 8-core 2.1 GHz CPU to conduct our numerical simulations.

### A. Problem Setup

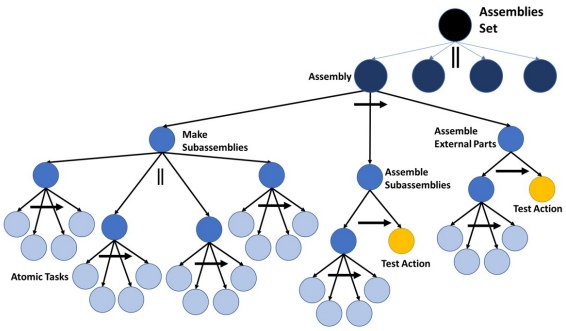


Fig. 3: Hierarchical task network used to represent an example assembly inspired from a satellite assembly domain.

Our evaluation framework is inspired by a high-mix, low-volume satellite assembly domain. Our planner must generate task assignments for a robot cell, completing four identical assemblies. Each assembly begins with four subassemblies that can be completed in parallel. After completing these subassemblies, they are integrated to form a full assembly and undergo a functionality test. If the test is successful, external component parts are added to the assembly and tested again. However, if a failure occurs during the testing phase, additional contingency tasks are added to the HTN.

The contingency task formulation is the following: first, the assembly undergoes a rework process where the assembly is disassembled, and the issue is resolved, and then reassembly tasks are added to the task set. Each task can be completed by 1-2 agent types, including a Testing Agent responsible for evaluating assembly functionality, a Rework Agent in charge of performing the rework task for failed assemblies, three agents with unique task proficiencies, and a human agent that can handle all tasks. The task durations are randomly generated, ranging from 10 to 25 time units. Lastly, we randomly choose 25% of tasks that may fail. We generate success probabilities for those tasks to be between 0.80-0.90 when completed by a robot, else it is set to 0.95 if completed by a human agent. We generate three problem formulation case studies of 30 tasks, 50 tasks, and 100 tasks.

### B. Baselines

We implement and evaluate our method against three baseline approaches.

1) **Constraint Program Scheduler:** A classical mixed integer program to compute and execute a full schedule. When the system state deviates from the schedule, we rerun the scheduler to generate a new task assignment. This is similar to optimistic replanning approaches [37].
2) **1-Step Lookahead with Optimistic MILP Rollouts:** The formulation from our prior work [**?**]. We sample states from possible actions that can be taken at the current state and use MILP to generate state-action trajectory rollouts.
3) **MCTS:** A centralized monte-carlo tree search using the MMDP formulation to expand on states and actions.

All three baselines are online model-based planning approaches. The classical scheduler is a reactive approach that performs sequential decision-making, but does not consider the uncertainty of the problem. The 1-step lookahead method is a more informed approach that considers the uncertainty of the problem by sampling failure states. The MCTS approach plans using the MMDP framework, and can quickly generate estimates through random sampling, however, it can still suffer from the large branching factor.

### C. Scalability of the Lower Level

We first analyze how scalable the lower level is based on the number of tasks in the task set and the level of constraint in the hierarchical task network. To change the problem's level of constraint, we vary the number of subassemblies that have sequential ordering constraints in the HTN. We classify the resulting constraint level categories as low (6 constraint nodes), medium (12 constraint nodes), and high (18 constraint nodes). Our goal is to determine the minimum

computation time required to achieve a solution with a makespan that is 95% of an optimal solution. However, finding a provably optimal solution can be extremely time-consuming. Therefore, we ran the solver for 10 minutes to generate the optimal solution benchmark.

| Constraint: | Low | | Medium | | High | |
|---|---|---|---|---|---|---|
| Warm Start: | No | Yes | No | Yes | No | Yes |
| 40 Tasks | 0.2s | 0.1s | 0.9s | 0.25s | 0.1s | 0.1s |
| 70 Tasks | 3.2s | 0.1s | 10.0s | 2.8s | 1.4s | 0.5s |
| 120 Tasks | 7.2s | 0.9s | 16.3s | 4.1s | 10.8s | 3.2 s |

TABLE I: Computation time required to calculate a schedule that is on average 95% of an optimal solution. The number of tasks chosen is the worst-case number of tasks the lower level has to optimize for the given case studies.

Table I shows the lower-level computation results for our CP-SAT implementation. We clearly see that the required computation time increases exponentially with the number of tasks. Furthermore, the task set that is "medium" constrained required the largest computation times. In contrast the "low" and "high" constrained problems had lower computation times because a highly constrained problem has a smaller solution space to search in, while a lowly constrained problem is more amenable to linear relaxation techniques used by the CP-SAT solver.

A useful feature of modern mathematical programming and constraint programming solvers is their ability to use a previous solution to warm-start future computations. Consider the following, when we sample contingency states from an initial trajectory of states and actions, we are identifying deviations from that trajectory. Since these deviations may not be extremely significant, we can use the decision variable values from the previous solution to warm-start the next search. To test this idea, we introduce a presolve step of 120 seconds for the initial solution. For all identified deviations, we use the presolved solution as a starting point and measure the average time needed to find 95% of the solution. Our results show that warm-starting reduces computation time for the lower level.

### *D. Makespan Evaluation*

We tested our approach against three baselines on the "highly constrained" assembly HTN. For all four approaches, we interleave planning and execution such that the planner is called each time the system is in a new state. To determine the lower-level computation time limit, we referred to the results in Table I. We further specified how many times each lower-level mechanism could be called: 10 times for the mitigation and prevention reasoning mechanisms and 30 times for the recovery reasoning mechanism. This was based on initial experimentation, which showed our algorithm worked well with these parameters. In total, the lower-level deterministic scheduler was called 50 times. We calculated the average planning time of our approach for each case study, and we gave the same amount of planning time for each baseline. The extremely large scenario space makes it difficult to evaluate our approach via random simulation runs. In order to enable a direct comparison of our method to the baselines, we instead used predefined failure scenarios to measure assembly makespans. For each case study, we created 10 failure scenarios where we selected 1-8 tasks that would fail. As stated in the problem setup, multiple contingency tasks are added when a task fails, which causes delays if the contingencies are not proactively managed. Lastly, we also generate a scenario where no tasks fail.

Our approach performed as well as or better than the benchmarks in all scenarios. Table II displays the average makespan results for all four methods, as well as the percentage improvement our approach had in comparison to the baselines. We observed that reactively planning consistently underperformed in comparison to our approach, highlighting the benefit of proactively managing contingencies in order to improve makespan. Based on the characteristics of manufacturing assemblies, we expect improvement on the order of 15% for the makespan, which is a significant improvement in industrial applications. For MCTS, We hypothesize the method did not perform as well due to the long horizon required for reasoning over possible contingency states and the large branching factor. Finally, we see that our proposed approach outperforms our previous work [**?**] in large case studies, showing improvement in makespan when explicitly considering both mitigation and prevention.

| Average Makespan | | | | |
|---|---|---|---|---|
| # Tasks | R-MILP | MCTS | 1-Step | HO |
| 30 | 368s | 352s | 341s | 339s |
| 50 | 461s | 446s | 427s | 394s |
| 100 | 726s | 710s | 670s | 648s |

| Percentage Improvement over Baselines | | | |
|---|---|---|---|
| # Tasks | R-MILP | MCTS | 1-Step |
| 30 | 7.9% | 3.7% | 0.6% |
| 50 | 14.5% | 11.7% | 4.5% |
| 100 | 12.0% | 8.4% | 5.2% |

TABLE II: Average makespan and percentage improvement for baselines in comparison to our hierarchical hindsight optimization-based approach (HO).

## VIII. Conclusions

We present a hierarchical approach for multi-robot task allocation under uncertainty that decouples the task scheduling and decision-making under uncertainty problem, which we then solve using hindsight optimization. Our results demonstrate that using constraint programming solvers to solve a mixed integer program is an effective tool to quickly compute low-level task schedules. Furthermore, we show at a high level that our approach effectively reasons over possible contingencies that may occur during task execution. We effectively accomplish this via a framework for reasoning in hindsight on how the system can recover, mitigate, and prevent contingencies. In practice, our approach shows improvement in executed makespan for a complex assembly domain in comparison to multiple baselines.

## Acknowledgements

This work was supported by the National Science Foundation NRI (# 2024936).

## REFERENCES


[1] A. Ham and M.-J. Park, "Human–robot task allocation and scheduling: Boeing 777 case study," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1256–1263, 2021.

[2] N. Dhanaraj, N. Ganesh, R. Gurav, M. Jeon, O. M. Manyar, S. Narayan, J. Park, Z. Yu, and S. K. Gupta, "A human robot collaboration framework for assembly tasks in high mix manufacturing applications," in *International Manufacturing Science and Engineering Conference*, vol. 87240. American Society of Mechanical Engineers, 2023, p. V002T07A011.

[3] Í. R. da Costa Barros and T. P. Nascimento, "Robotic mobile fulfillment systems: A survey on recent developments and research opportunities," *Robotics and Autonomous Systems*, vol. 137, p. 103729, 2021.

[4] H. Aziz, A. Pal, A. Pourmiri, F. Ramezani, and B. Sims, "Task allocation using a team of robots," *Current Robotics Reports*, pp. 1–12, 2022.

[5] J. H. Kang, N. Dhanaraj, S. Wadaskar, and S. K. Gupta, "Using Large Language Models to Generate and Apply Contingency Handling Procedures in Collaborative Assembly Applications," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024.

[6] S. Choudhury, J. K. Gupta, M. J. Kochenderfer, D. Sadigh, and J. Bohg, "Dynamic multi-robot task allocation under uncertainty and temporal constraints," *Autonomous Robots*, vol. 46, no. 1, pp. 231–247, 2022.

[7] S. W. Yoon, A. Fern, R. Givan, and S. Kambhampati, "Probabilistic planning via determinization in hindsight." in *AAAI*, 2008, pp. 1010–1016.

[8] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[9] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[10] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robotics and Autonomous Systems*, vol. 90, pp. 55–70, 2017.

[11] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[12] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[13] E. G. Jones, M. B. Dias, and A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," *Autonomous robots*, vol. 30, pp. 41–56, 2011.

[14] B. Fu, W. Smith, D. M. Rizzo, M. Castanier, M. Ghaffari, and K. Barton, "Robust task scheduling for heterogeneous robot teams under capability uncertainty," *IEEE Transactions on Robotics*, pp. 1–19, 2022.

[15] H. Wang, W. Chen, and J. Wang, "Coupled task scheduling for heterogeneous multi-robot system of two robot types performing complex-schedule order fulfillment tasks," *Robotics and Autonomous Systems*, vol. 131, p. 103560, 2020.

[16] M. L. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[17] B. Çaliş and S. Bulkan, "A research survey: review of ai solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, pp. 961–973, 2015.

[18] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[19] H. C. Lau, M. Sim, and K. M. Teo, "Vehicle routing problem with time windows and a limited number of vehicles," *European journal of operational research*, vol. 148, no. 3, pp. 559–569, 2003.

[20] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[21] V. C. Wiers, "A review of the applicability of or and ai scheduling techniques in practice," *Omega*, vol. 25, no. 2, pp. 145–153, 1997.

[22] C. Zhang and J. A. Shah, "Co-optimizating multi-agent placement with task assignment and scheduling." in *IJCAI*, 2016, pp. 3308–3314.

[23] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 618–12 624.

[24] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," in *Robotics: Science and Systems Foundation*, 2013.

[25] Y. Cheng, L. Sun, C. Liu, and M. Tomizuka, "Towards efficient human-robot collaboration with robust plan recognition and trajectory prediction," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2602–2609, 2020.

[26] Y. Cheng, L. Sun, and M. Tomizuka, "Human-aware robot task planning based on a hierarchical task model," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1136–1143, 2021.

[27] Y. Cheng and M. Tomizuka, "Long-term trajectory prediction of the human hand and duration estimation of the human action," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 247–254, 2021.

[28] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 5469–5476.

[29] L. Johannsmeier and S. Haddadin, "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 41–48, 2016.

[30] J. Leu, Y. Cheng, M. Tomizuka, and C. Liu, "Robust task planning for assembly lines with human-robot collaboration," in *Proceedings of the International Symposium on Flexible Automation 2022 International Symposium on Flexible Automation*. The Institute of Systems, Control and Information Engineers, 2022, pp. 188–195.

[31] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, *Algorithms for decision making*. MIT press, 2022.

[32] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[33] T. Vodopivec, S. Samothrakis, and B. Ster, "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.

[34] D. Müller, M. G. Müller, D. Kress, and E. Pesch, "An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning," *European Journal of Operational Research*, vol. 302, no. 3, pp. 874–891, 2022.

[35] P. Brucker and R. Schlie, "Job-shop scheduling with multipurpose machines," *Computing*, 1990.

[36] G. Da Col and E. C. Teppan, "Industrial-size job shop scheduling with constraint programming," *Operations Research Perspectives*, vol. 9, p. 100249, 2022.

[37] S. W. Yoon, A. Fern, and R. Givan, "Ff-replan: A baseline for probabilistic planning." in *ICAPS*, vol. 7, 2007, pp. 352–359.