# Proactive Contingency-Aware Task Allocation and Scheduling in Multi-Robot Multi-Human Cells via Hindsight Optimization

Neel Dhanaraj, *Student Member, IEEE*, Heramb Nemlekar, *Member, IEEE*, Stefanos Nikolaidis, *Member, IEEE*, and Satyandra K. Gupta, *Fellow, IEEE*

***Abstract*—Multi-robot systems are becoming more common in various real-world applications, such as manufacturing and warehouse logistics. However, task allocation and scheduling for a multi-agent team face complex challenges due to the need to simultaneously consider time-extended tasks, task constraints, and uncertainties in execution. Potential task failures or contingencies can add additional tasks to recover from the failures, and reactively addressing contingencies can decrease teaming efficiency. To efficiently and proactively consider contingencies, this paper proposes treating the problem as a multi-robot task allocation under uncertainty problem. We suggest a hierarchical approach that divides the problem into two layers. We use mathematical program formulation for the lower layer to find the optimal solution for a deterministic multi-robot task allocation problem with known task outcomes. The higher-layer search intelligently generates more likely combinations of contingency scenarios and calls the inner-level search repeatedly to find the optimal task allocation sequence for the given scenario. We validate our results in simulation for manufacturing applications and demonstrate that our method can reduce the effect of potential delays from contingencies.**

***Note to Practitioners*—Automation engineers interested in deploying robotic cells in low-volume applications need to consider contingency handling. When the occurrence of contingencies can be characterized as probability distributions, it is often useful to consider using a proactive approach for task allocation and scheduling. To implement our algorithm, automation engineers will need to develop a hierarchical task network specified by domain experts that models task constraints and a task-agent duration model, which may be generated from simulation environments. Furthermore, they must identify tasks that can result in contingencies and describe them with a probabilistic model. This model can be generated from historical data and/or real-world experiments. Lastly, for addressing the contingency, the practitioner will need to specify a task procedure to recover from a specific contingency type. To run the algorithm, we found that repeatedly approximating the best proactive task allocation for a fixed computation budget and dispatching the best tasks worked well. The computation budget required to approximate the best task allocation is directly affected by the number of contingency scenarios that can be sampled. Therefore, the practitioner must determine a suitable computational budget empirically based on the number of contingencies that can occur.**

***Index Terms*—Multi-robot systems, flexible manufacturing systems, adaptive scheduling, uncertainty, task analysis, robots, multi-robot task allocation, proactive scheduling, contingency management, hindsight optimization.**

Received 31 August 2024; revised 9 December 2024; accepted 2 February 2025. Date of publication 27 February 2025; date of current version 18 April 2025. This article was recommended for publication by Associate Editor Z. Pei and Editor T. Nishi upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant NRI 2024936. *(Corresponding author: Neel Dhanaraj.)*

Neel Dhanaraj, Stefanos Nikolaidis, and Satyandra K. Gupta are with the Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: dhanaraj@usc.edu; nikolaid@usc.edu; guptask@usc.edu).

Heramb Nemlekar is with the Department of Mechanical Engineering, Virginia Tech, Blacksburg, VA 24060 USA (e-mail: hnemlekar@vt.edu).

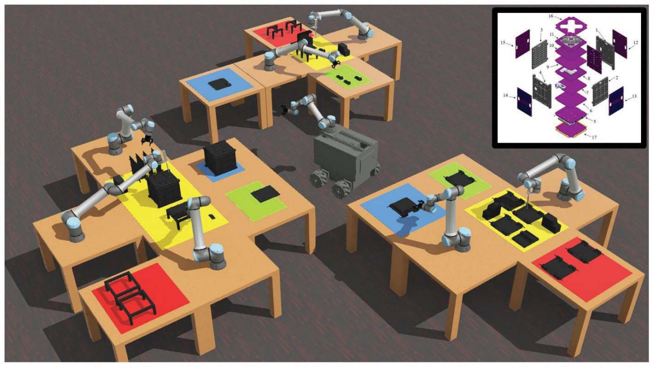
Digital Object Identifier 10.1109/TASE.2025.3546281

## I. Introduction

MULTI-ROBOT systems are increasingly being considered for automating complex assembly operations in high-mix, low-volume (HMLV) applications, which have traditionally been carried out by human operators. Collaborative robot assembly cells, such as those used in the robotic assembly of satellites or ATVs (as shown in Figure 1), can enhance human productivity by allowing robots to perform repetitive tasks while humans handle high-value and fine manipulation tasks [1], [2]. Examples of tasks executed by robots in high-mix environments include material handling, surface preparation, adhesive dispensing, screwing, insertion, and inspection.

Multi-robot task allocation (MRTA) during robot assembly can be challenging due to uncertainties in task execution, the robots themselves, and the surrounding environment [3], [4], [5]. These uncertainties can lead to contingencies that must be addressed and recovered from [6], [7], and [8]. For example:

1) Screw Driving: Vision-guided screw driving may not always be perfect. A robot may occasionally need to make multiple attempts to successfully complete the operation due to alignment issues or variability in the screws or materials.
2) Adhesive Dispensing: Variations in the viscosity of the adhesive can affect the success of dispensing, leading to improper application and the need for correction.
3) Disassembly Operations: In refurbishing and recycling contexts, disassembly tasks may fail due to the unpredictable wear and tear of components, leading to challenges.
4) Sensor Malfunction or Degradation: Even seemingly simple tasks can fail if a sensor malfunctions, such as a camera lens becoming obstructed by dust, resulting in poor image quality and failed operations.
5) Deformation During Force-Application Operations: Tasks involving the application of force, such as sanding, can fail if the part deforms excessively during the


(a) Satellite Assembly Cell

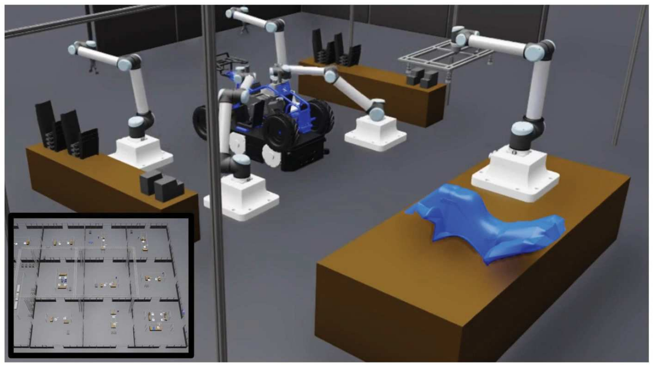

(b) ATV Assembly Factory

Fig. 1. Examples of robotic assembly cells that are used to motivate this work: 1) a simulation of an 8-robot featureless assembly cell intended for high-mix, low-volume satellite manufacturing and 2) a micro-factory with ten robot cells for all-terrain vehicle (ATV) assembly. Different stations have robots that assemble various sub-assemblies, which must be installed in specific sequences. However, the fixtureless nature of the cell and complex insertion tasks create uncertainty in task execution. To minimize the occurrence of failed tasks and wasted time due to bottlenecks, our planning approach must consider the complex task constraints for assembling the satellites/ATVs while managing uncertainty during policy generation.

operation, which may not be detectable until the failure occurs.

In scenarios where failures are more likely, it is not efficient to assign all challenging tasks to humans, nor should contingencies be reactively managed, where the robotic cell must wait for the human to address the issue. Instead, it is preferable to 1) allow robots to attempt these tasks and, if a failure occurs, the system can then take corrective actions; and 2) plan over a time horizon to proactively consider potential contingency scenarios during task allocation and scheduling. This approach motivates the need for effective dynamic task allocation methods for the human-robot team in such applications to maintain teaming efficiency.

A task allocation and scheduling problem for multi-robot team settings in the presence of contingencies requires us to consider the effect of two different combinatorial effects. The first effect comes from many alternative ways of allocating robots to tasks. The second effect comes from many different potential outcomes due to uncertainties in task execution duration and task execution failure. 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.

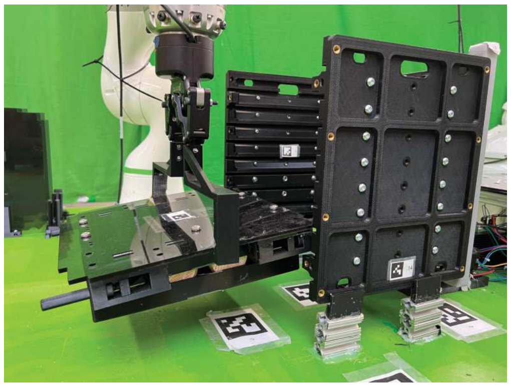


Fig. 2. An example of a contingency occurring during the assembly process. The robot fails to insert the battery sub-assembly into the main assembly, requiring additional contingency tasks to be executed.

Choudhury et al. [9] proposed a computationally efficient solution to address the MRTA under uncertainty problem by decoupling the problem into a two-level hierarchical approach. There are two ways to construct hierarchies: 1) Reason over task uncertainty for each agent separately at the low level, and then coordinate the agents at the high level (as proposed by Choudhury et al. [9]) and 2) Allocate tasks to the agents and then consider the effect of uncertainty on the entire team.

For applications with interlinked task constraints (such as assembly tasks), we identified that 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. Therefore, we consider the second alternative of constructing a two-level hierarchy. *1) The lower layer coordinates multi-agent decisions, and 2) the high-level reasons for uncertainty during team execution.*

As a part of our exploratory work, we considered the following decoupled approach, which is similar in spirit to the previously mentioned hierarchy: a one-step lookahead algorithm with optimistic rollouts generated using mixed integer programs [10]. The method samples failure contingencies at a higher level for each rollout and executes more deterministic rollouts from these contingency states, thereby building a search tree. This approach enabled us to select task allocations with good options to recover from failures. Let us now consider that not only can we reason on how to recover from contingencies, but we can also reason about mitigating and preventing contingencies.

We extended the initial exploratory work to investigate our next insight, which is that, in addition to reasoning on how to recover from contingencies, one can also explicitly reason about how to mitigate and prevent them [11]. Consider a scenario where a robotic manipulator is performing an insertion task. However, this task may fail due to the high uncertainty in the slot location, as shown in Figure 2. To plan for such a contingency, there are three ways to reason about this contingency. Firstly, if the task fails, the robot can plan to attempt the task again by waiting for the mobile agent to perform high-resolution imaging to reduce uncertainty, which we refer to as 'recovery'. Secondly, the robot can consider performing the task earlier or later when the mobile agent is more likely to be available for imaging, which we call 'mitigation'. Lastly, the task can be assigned to a different robotic station that has close-up imaging capability, which we refer to as 'prevention'.

In this work, we present a hierarchical method that uses constraint programming solvers and contingency scenario sampling to generate task schedules using hindsight optimization. This is done by explicitly reasoning how to recover, mitigate, and prevent contingency scenarios in hindsight. This method was first introduced in our second exploratory work [11]. Our manuscript further extends exploratory work, and we present a refined algorithm that includes an intelligent method for sampling contingency scenarios using upper confidence bounds for the higher level, as well as details of the mathematical program used by the constraint programming solver for the lower level. We also provide insights on the complexity of the proposed approach, along with results for two contingency domains and a discussion on the usefulness of proactive contingency management in different assembly domains.

## II. Related Works

### A. Multi-Agent Task Allocation

Multi-agent task allocation has been extensively studied in various fields of which many proposed centralized/exact as well as decentralized/distributed approaches to solving allocation problems that minimize objectives like task makespan, energy consumption, and teaming efficiency under different complex constraints such as spatial, ordering and time window constraints [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. Several works have surveyed different problem variants and approaches to solve them [22], [23], [24], [25]. According to Gerkey and Matarić taxonomy [26], the deterministic version of our problem falls under the category of 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. [27] extended the previous taxonomy by considering explicit task constraint dependencies. Our problem can be described as having CD (Complex Dependencies). In this class of problems, the effective utilization of an agent's schedule depends on other agents' schedules. Due to the uncertainty and task constraints, our problem is CD[ST-SR-TA], which is strongly NP-hard. For similar problems, researchers have needed to develop specialized methods for the studied domains in order to tackle such problems and make them tractable [28], [29], [30].

### B. Task Assignment and Scheduling

Multi-robot task allocation is a problem that is closely related to well-studied problems in the fields of job shop scheduling [31], [32], vehicle routing [33], [34], and general operations research (OR) [35]. The closest problem to ours is flexible job shop scheduling (FJSS), where operations are assigned to machines and scheduled to minimize the overall time taken to complete tasks while taking into account resource and task constraints. Researchers in these communities have made significant advancements in solving these problems using mixed-integer linear programs, which rely on solving continuous linear program relaxation as the foundation of the search and constraint program solvers whose search process relies on advanced constraint propagation techniques. For robotics, robots can be considered as machines or vehicles and tasks as jobs or nodes [22]. This concept has enabled researchers to adapt state-of-the-art Operational research (OR) methods to solve complex robot task assignments and scheduling while considering temporal and spatial constraints [36], [37]. Hierarchical task networks and precedence constraints have also been used to represent complex assembly constraints [38], [39], [40], [41], [42]. OR methods are effective for small to medium-sized scheduling problems; however, they become inefficient when applied to larger problems. In contrast, deep reinforcement learning (DRL) methods have shown promise in efficiently and approximately solving large task allocation challenges. Previous studies have demonstrated the application of DRL to address the flexible job shop scheduling problem [43], [44]. These methods can learn priority dispatching rules, making them particularly beneficial for dynamic scheduling scenarios [45], [46].

### C. Multi-Agent Sequential Decision Making Under Uncertainty

Multi-agent sequential decision-making is usually modeled as a Multi-Agent Markov Decision Process (MMDP). In our case, we are dealing with a centralized MMDP that has full knowledge and a shared objective. However, standard Markov Decision Process approaches are not practical for solving an MMDP due to the exponential joint action space and state space. Consequently, researchers have turned to model-free deep reinforcement learning (DRL) techniques [47], [48], as well as model-based online planning approaches [49]. Deep reinforcement learning has proven effective for managing stochastic processing times and random task arrivals, as it does not require explicit modeling of the environment. However, there are currently no existing DRL approaches that proactively address the impact of failures and contingencies since each type of failure may require a different contingency model. Recent work has shown that the challenges of multi-robot task allocation under uncertainty can be decoupled for certain problems [9]. Specifically, the authors propose computing individual agent task allocation policies at the lower level and performing multi-agent coordination at the high-level via conflict resolution of agent task allocations. 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 can result in a robot's wasted time. Our work is new in that we present an alternative method to decouple the problem and demonstrate its strength on another class of problems.

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

### A. *Task Allocation Problem*

We begin with the problem formulation for the multi-robot task allocation problem. The problem involves a set of $n$ agents $\mathcal{A}_i \in \mathbf{N}$ and a set of $m$ tasks $\tau_j \in \mathcal{T}$. Each agent can be either idle, meaning they are available to be assigned a task, busy, meaning they have been assigned a task: $a_{\mathcal{A}_j} = \mathcal{A}_i \leftrightarrow \tau_j$ (agent $i$ is assigned to task $j$), or unavailable. At any given time, an agent can only be assigned to one task. Each task has a specific subset of agents that can be assigned to it, denoted by $\mathbf{N}_{\tau_\mathbf{j}} \subseteq \mathbf{N}$. A task can either be available, in progress or completed. The completed state is a set grouping that can be further divided into sub-states, such as succeeded, succeeded with delay, or failed. These sub-states are specific to the task's domain. Each task $\tau_j$ must be completed by one agent $\mathcal{A}_k$ and has a process time cost $p_{j,k}$ that is dependent on which agent was assigned to the task.

Our motivation for solving this problem comes from complex assembly tasks that have task ordering constraints. To represent these input constraints, we use a hierarchical task network (HTN) representation [42]. The HTN has a root node that encompasses the task set $\mathcal{T}$ and leaf nodes that represent atomic tasks $\tau_j$. All other nodes are subtask groups that can be further broken down into smaller tasks. The non-leaf nodes further indicate a relationship for their child nodes. We use two types of relationships: sequential nodes ($\rightarrow$) and parallel nodes ($\|$). A sequential node encodes precedence constraints and indicates that its child nodes must be completed sequentially from left to right. Parallel nodes indicate that there are no constraints and that their child nodes can be executed concurrently.

### B. *Task Execution Uncertainty and Contingencies*

Our main focus is to solve the problem of multi-robot task allocation under uncertainty. We consider scenarios where agents may fail their assigned tasks, which requires additional recovery tasks, which we refer to as contingencies. We assume prior knowledge of probability distributions for contingencies to be known beforehand and are domain-dependent. This is a reasonable assumption because the probability distributions can be derived from historical data, expert knowledge, and simulation model-based predictions.

In order to describe the uncertainty dynamics of a task assigned to an agent, we use a task transition function $T(s_{\tau_i}, a_{\mathcal{A}_j}, s'_{\tau_i})$. This function returns the probability that a task with initial state $s_{\tau_i}$ and task assignment $a_{\mathcal{A}_j}$ will transition to a final completed state $s'_{\tau_i}$. This final state can either be a succeeded or a task contingency state. For example, when assembling a product, there is a probability a robot may fail to insert a part and damage it, resulting in a contingency state.

In the event of a task contingency being observed, a separate set of contingency tasks denoted as $\mathcal{T}^{cont}_{\tau_i}$, may need to be performed to rectify the issue. To continue the damaged part example, additional contingency tasks like disabling the part, acquiring a new one, and installing it into the assembly will be included in the task set. We define a contingency function $F(\tau_i, \mathcal{T}, \mathrm{HTN}) \rightarrow (\mathcal{T}', \mathrm{HTN}')$ that gets triggered when a task $\tau_i$ in the task set fails. It returns a new task set $\mathcal{T}' = \mathcal{T} \cup \mathcal{T}^{cont}_{\tau_i}$ and a new hierarchical task network HTN$'$. The new HTN specifies how to execute the contingency tasks to recover from the failure.

### C. *Multi-Agent Markov Decision Process*

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

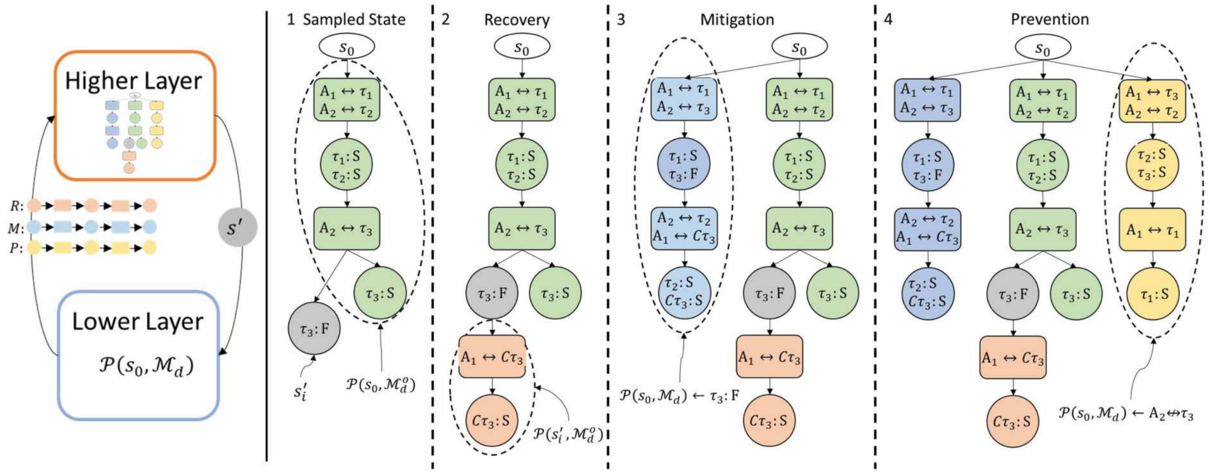
1) $N = \{\mathcal{A}_1, \ldots, \mathcal{A}_i\}$: Set of agents.
2) $S$: is the state space of the entire system where a state $s \in S$ consists of a factored representation of all task states and the agents states $s = (\{s_{\tau_1} \ldots, s_{\tau_m}\}, \{s_{\mathcal{A}_1} \ldots, s_{\mathcal{A}_n}\})$. Furthermore, each state is a decision epoch where a new task becomes available for idle agents or an agent finishes a task and is ready for a new task assignment.
3) $A$: is the action space available to the system at each state where an action $a \in$ is a joint action consisting of a set of agent actions $a = \{a_{\mathcal{A}_1}, \ldots, a_{\mathcal{A}_n}\}$. 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 transition probability function that returns the probability that the state transitions from one decision epoch to another.
5) $C(s_i)$: is the accumulated time taken to reach $s_i$ from $s_0$.
6) $s_g \in S_g$: is a terminating goal state for the MDP where all tasks in the task set have been completed.

### D. *Examples of Contingencies*

We have gained experience in establishing an HMLV robot assembly cell from our previous work [50]. This has helped us identify different contingencies that can occur during operation. Some of these contingencies can only be managed reactively, such as when a part is unavailable. Others can be probabilistically modeled and proactively managed, such as part insertions, screwing tasks, and robot failures. Based on our prior implementation, we propose two broad types of contingencies that we consider for our problem instantiation and motivation. We first identified that an agent can complete a task within a nominal duration or fail early, causing the robot to be incapacitated. The robot can not execute more tasks until a human agent performs a recovery task. We have also identified a second type of contingency where tasks may fail, which is detected later during a quality control testing step. To recover from such failures, additional tasks are added to the HTN and task set. For instance, during testing, the assembly is detected to have been damaged earlier. Additional contingency tasks of disassembling and reworking the parts, as well as redoing the assembly operations, are added to the HTN and task set.

### E. *Problem Statement*

We aim to find a policy that efficiently assigns and schedules tasks that minimize the expected time required to complete all



Higher Layer
R:
M:
P:
$s'$
Lower Layer
$\mathcal{P}(s_0, \mathcal{M}_d)$
1 Sampled State
2 Recovery
3 Mitigation
4 Prevention
$s_0$
$A_1 \leftrightarrow \tau_1$ $A_2 \leftrightarrow \tau_2$
$\tau_1$: S $\tau_2$: S
$A_2 \leftrightarrow \tau_3$
$\tau_3$: F
$\tau_3$: S
$s_i'$
$\mathcal{P}(s_0, \mathcal{M}_d^o)$
$A_1 \leftrightarrow C\tau_3$
$C\tau_3$: S
$\mathcal{P}(s_i', \mathcal{M}_d^o)$
$A_1 \leftrightarrow \tau_1$ $A_2 \leftrightarrow \tau_3$
$\tau_1$: S $\tau_3$: F
$A_2 \leftrightarrow \tau_2$ $A_1 \leftrightarrow C\tau_3$
$\tau_2$: S $C\tau_3$: S
$\mathcal{P}(s_0, \mathcal{M}_d) \leftarrow \tau_3$: F
$A_1 \leftrightarrow \tau_3$ $A_2 \leftrightarrow \tau_2$
$\tau_2$: S $\tau_3$: S
$A_1 \leftrightarrow \tau_1$
$\tau_1$: S
$\mathcal{P}(s_0, \mathcal{M}_d) \leftarrow A_2 \leftrightarrow \tau_3$

Fig. 3. 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 $\mathcal{M}_d$ preventing $\tau_3$ from being assigned to $\mathcal{A}_2$ and again generates a new path (yellow) that is connected to $s_0$. The algorithm repeatedly generates all three paths simultaneously to reason about each possible contingency.

tasks in the task set (makespan) and have the system terminate at goal state $s_g$. It is important to note that relying on a single task allocation that is robust to small task duration deviations, also known as robust scheduling problems, would not be enough for our problem formulation. This work focuses on dealing with contingencies that may cause significant disruptions during the process. Therefore, we must determine a policy $\pi : S \rightarrow A$, that gives efficient recourse at contingency states so that we minimize the expected makespan of reaching a goal state $C(s_g)$ from an initial state $s_0$. Our objective function becomes the following.

$$\underset{\pi \in \Pi}{\operatorname{argmin}} \ \mathrm{E}[C(s_g)|s_0, \pi] \tag{1}$$

## F. Overview of Approach

We have developed a hierarchical approach to generate a search tree, where the root node of the tree is the current state, denoted as $s_0$, the terminating goal states are the leaf nodes, and all other nodes are possible states and best actions to take at the state that minimizes expected makespan. To construct the tree, we first address the challenge of multi-agent coordination at the low level. We assume deterministic dynamics and compute good task assignments and schedules using a constraint programming formulation. This schedule is then converted into a sequence of states and actions based on our MMDP formulation. The low level of our search is discussed in Section IV.

In Section V, we propose the high level of our search, where we address the challenge of sequential decision-making under uncertainty by sampling likely alternative failure states that the system can transition into when executing its task schedule. We then compute new task allocation schedules that recover, mitigate, and prevent these failures. We begin with a schedule solution for the most likely task outcomes scenario. By converting the schedule into a state-action sequence, we then sample other scenarios that may occur. We use the mitigation reasoning mechanism to determine how to schedule tasks given the scenario was known in hindsight. We also use the prevention mechanism to consider how a different agent can complete the task by preventing the original agent from failing the task. Lastly, the recovery reasoning method is used to create a policy tree by generating recourse strategies if a contingency were to occur.

In Section VI, we propose the hindsight optimization algorithm that connects the low-level and high-level formulations. The algorithm is designed to efficiently search through likely and relevant determinized scenarios instead of exploring the joint state and joint action space, which is computationally expensive. We achieve this by using proposed contingency sampling and reasoning mechanisms and solving these scenarios in hindsight using a deterministic solver. The solutions of all the sampled scenarios are then merged into a single policy tree iteratively. Figure 3 provides an illustrative example of the approach, and we describe the details of our methodology in subsequent sections.

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

To decouple multi-agent coordination from decision-making under uncertainty, we create a simpler *determinized* problem $\mathcal{M}_d$, where task allocation outcomes are deterministic. This allows us to use a planner $\mathcal{P}(s_0, \mathcal{M}_d)$ to solve the classical MRTA problem and return a task allocation and schedule $\pi_d$. From the deterministic transition dynamics, the allocation schedule will yield a sequence of states (decision epochs) and actions (task assignments), with the trajectory terminating at a goal state: $[s_{0:d}, a_{0:d-1}] \leftarrow \pi_d$, where $s_d \in S_g$. The low-level optimization problem becomes the following: given the initial system state $s_0$, we want to find the allocation $\pi_d$ that minimizes the objective in Equation 2 while assuming deterministic dynamics given by $\mathcal{M}_d$.

$$\underset{\pi_d \in \Pi_d}{\operatorname{argmin}} \, [C(s_g)|s_o, \pi_d] \tag{2}$$

Essentially, we instantiate $\mathcal{M}_d$ by *determinizing* all task-agent assignment outcomes. For example, in the optimistic case, if a task is assigned to an agent $a_{\mathcal{A}_j} = \mathcal{A}_j \leftrightarrow \tau_i$, then $s'_{\tau_j}$ will result in *succeeded*, i.e., $T(s_{\tau_i}, a_{\mathcal{A}_j}, s'_{\tau_j}) = 1$. We initially generate $\mathcal{M}_d$ by first *determinizing* each task-agent assignment outcome to be its most likely outcome. If a task outcome is determinized as a contingency, then the resulting contingency tasks and constraints are added to the task set and HTN of $\mathcal{M}_d$. Lastly, we set the task process time for an agent to be the expected task duration for the determinized outcome.

## *A. Low-Level Constraint Programming Formulation*

Various methods have been proposed to solve deterministic MRTA problems. The recent advancements in mixed-integer linear programming (MILP) and constraint programming (CP) solvers motivate us to reformulate the problem as a mathematical program. References [51], [52], and [53], and use off the shelf solvers to generate the solutions. Specifically, we chose a constraint programming approach based on a flexible job shop problem formulation. This decision is informed by the recent success of the CP-SAT solver, which quickly returns high-quality solutions and enables us to explicitly model constraints without needing linearization techniques. The aim of solver $\mathcal{P}$ is to find the best value for the binary agent-task assignment decision variables and the integer task start times that minimize the makespan while subject to task constraints. We extract a problem instance from the determinized MDP and current state $s_0$ and present the mathematical program formulation below.

**Sets:**

$\mathcal{T}$: Set of incomplete (unattempted and inprogress) tasks at $s_0$
$\mathcal{T}_{curr}$: Set of inprogress task assignments at $s_0$
**N**: Set of agents
$\mathcal{T}_\mathbf{k}$: Set of tasks that can be completed by agent $k$
$\mathbf{N}_{\tau_i}$: Set of agents that can process task $i$
$P$: Set of task precedence pairs $(i, j)$ specifying $\tau_i$ must precede $\tau_j$

**Parameters:**

$p_{i,k}$: Processing time of task $i$ executed by agent $k$, if the task is inprogress then the processing time is the nominal process time - the elapsed duration

**Decision and Auxiliary Variables**

$$x_{i,k} = \begin{cases} 1 & \text{if task } i \text{is assigned to agent } k, \\ 0 & \text{otherwise.} \end{cases}$$

$t^s_i$: Integer start time of task $i$
$p_i$: Integer duration time of task $i$
$t^e_i$: Integer end time of task $i$
$t^s_{ik}$: Integer start time of task $i$ being completed by agent $k$
$t^e_{ik}$: Integer end time of task $i$ being completed by agent $k$

$$\text{Minimize} \quad t \tag{3}$$
$$\text{subject to} \quad t^e_i = t^s_i + p_i, \quad \forall i \in \mathcal{T} \tag{4}$$
$$x_{i,k} = 1 \rightarrow t^e_{i,k} = t^s_{i,k} + p_{i,k}, \quad \forall i \in \mathcal{T}, \ \forall k \in N \tag{5}$$
$$t^s_j \geq t^e_i, \quad \forall (i,j) \in P \tag{6}$$
$$t \geq t^e_i \quad \forall i \in \mathcal{T} \tag{7}$$
$$(x_{i,k} = 1) \wedge (x_{j,k} = 1) \rightarrow \left[(t^s_i + p_i \leq t^s_j) \vee (t^s_j + p_j \leq t^s_i)\right] \quad \forall i, j \in \mathcal{T}_k, \ i \neq j, \ \forall k \in N \tag{8}$$
$$\sum_{k \in N_{\tau_i}} x_{i,k} = 1 \quad \forall i \in \mathcal{T} \tag{9}$$
$$s_i = 0, \ x_{i,k} = 1 \quad \forall (i,k) \in \mathcal{T}_{\mathbf{curr}} \tag{10}$$
$$x_{i,k} = 1 \rightarrow t^s_i = t^s_{ik}, \ t^e_i = t^e_{ik} \quad \forall i \in \mathcal{T}, \ \forall k \in N \tag{11}$$

The main objective of this constraint program is to determine the task assignments and their corresponding start times, which will result in the minimum time required to complete all tasks. This minimum time, referred to as the makespan, is denoted by the variable $t$ and represents the time at which the last task is completed. Equation (3) formally defines the makespan as the objective to be minimized.

Equation (4) ensures that the end time $t^e_i$ of each task $i$ is equal to its start time $t^s_i$ plus the task's processing time $p_i$. This ensures that the duration of each task is properly accounted for in the schedule. Equation (5) specifies that if task $i$ is assigned to agent $k$ (i.e., $x_{i,k} = 1$), then the end time $t^e_{i,k}$ of the task-agent pair equals the start time $t^s_{i,k}$ plus the processing time $p_{i,k}$. Equation (6) enforces precedence constraints, ensuring that task $j$ cannot start until task $i$ is completed for all pairs of tasks $(i, j) \in P$ with a sequential ordering constraint. Equation (7) defines the makespan variable $t$ as greater than or equal to the end time of every task, ensuring that $t$ captures the time when the last task is completed.

Equation (8) ensures that each agent has non-overlapping schedules by stipulating that if two tasks, $i$, and $j$, are assigned to the same agent, $k$, their execution times must not overlap. In other words, either task $i$ must be completed before task $j$ begins, or task $j$ must finish before task $i$ starts. The use of a CP-SAT approach allows for this disjunctive condition to be defined explicitly. Equation (9) requires that each task is assigned to exactly one agent, ensuring that no task is left unassigned. Equation (10) states that the start times for in-progress task assignments $(i, k)$ equals zero and that task $i$ is assigned to an agent $k$. Finally, Equation 11 defines that if task $i$ is assigned to agent $k$, then the start time, duration, and end time of the task will be equal to the corresponding start time, duration, and end time of the task-agent pair.

## *B. Deterministic MRTA Planner*

Our deterministic planning approach is illustrated in Algorithm 1. The input is a start state $s_0$ and a determinized problem instance $\mathcal{M}_d$. From $\mathcal{M}_d$, we extract the task-agent models, the process times, and the hierarchal task network (Line 4), and from the start state, we extract the set of incomplete tasks, current task assignments and the set of agents (Line 5). Lastly, we extract the precedence constraint pairs from the HTN (Line 6). These sets are inputted in a constraint programming solver, which solves the constraint program formulated in the prior section and returns task start and end times as well as task assignments (Line 7). We then convert the solution into a sequence of states and actions (Line 8). This is necessary for sampling other contingency states from state-action pairs and constructing a search tree.

To extract the state-action trajectory, we first identify the set of time steps (decision epochs) from $t_i^s$ and $t_i^e$, where task(s) finished, resulting in a new state and an opportunity to execute an action. We first start with an empty sequence of actions. We then iterate through each time step and extract the subset of task assignments whose start time equals the time step. This subset of task assignments is an action $a_i$, which is appended to the set of actions. If there is a time step with no associated task assignment start time, we append a no-op action to the sequence of actions. This gives us a sequence of actions $[a_{0:t}]$. Next, we iteratively apply these actions beginning with $s_0$ and get the next deterministic states using the determinized MDP $\mathcal{M}_d$ transition function $s_{i+1} \leftarrow T_d(s_i, a_i)$. We do this until we get the state action trajectory $[s_{0:t+1}, a_{0:t}]$, where the last state will be $s_g$. In Section V, we will discuss how one can sample states from the state-action trajectory.

**Algorithm 1** Deterministic Planner

```
1: $\mathcal{M}_d$: Determinized MDP
2: $s_0$: Initial state
3: function $\mathcal{P}(s_0, \mathcal{M}_d)$
4:     $\mathcal{T}_\mathbf{k}, \mathbf{N}_{\tau_j}, p_{i,k}$, HTN $\leftarrow \mathcal{M}_d$
5:     $\mathcal{T}, \mathcal{T}_\mathbf{curr}, \mathbf{N} \leftarrow s_0$
6:     $P \leftarrow$ HTN
7:     $t_i^s, t_i^e, x_{i,k} \leftarrow$ SolveCP($\mathcal{T}_\mathbf{k}, \mathbf{N}_{\tau_j}, p_{i,k}, \mathcal{T}, \mathcal{T}_\mathbf{curr}, \mathbf{N}, P$)
8:     $[s_{i:t+1}, a_{i:t}] \leftarrow$ ExtractTraj($t_i^s, t_i^e, x_{i,k}, s_0, \mathcal{M}_d$)
9:     return $[s_{i:t+1}, a_{i:t}]$ ▷Return the state-action trajectory
```

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

At the high level, we want to sample other potential scenarios that can occur and evaluate the effect of the contingency by reasoning over how the system can recover from, mitigate, and/or prevent the contingency. Consider that the low-level deterministic task allocation $\pi_d$ is a path sequence where nodes are states and actions $s_{0:t+1}, a_{0:t}$. We then traverse the path and sample other possible states (scenarios) 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 $\mathcal{P}$. Instead of uniformly sampling scenarios from the scenario space, by sampling scenarios from the state-action trajectory, we can compute the path probability of the scenario state occurring $P(s_i'|s_i)$ and use this probability to prioritize the evaluation of scenarios that have a higher likelihood of occurring and are closer in time to the initial state$s_0$

Given $s_i'$ and $\mathcal{M}_d'$, we propose three mechanisms for reasoning over these scenarios: 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 3.

**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.

$$[s_{i:t+1}, a_{i:t}]_r \leftarrow \mathcal{P}_r(s_i', \mathcal{M}_d^r) \tag{12}$$

*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 $\tau_i$ : *failed* of the sampled state were known in hindsight.

$$[s_{0:t+1}, a_{0:t}]_m \leftarrow \mathcal{P}_m(s_i', \mathcal{M}_d^m) \qquad \mathcal{M}_d^m \leftarrow \tau_i : \textit{failed} \tag{13}$$

*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.

$$[s_{0:t+1}, a_{0:t}]_p \leftarrow \mathcal{P}_p(s_i', \mathcal{M}_d^p) \qquad \mathcal{M}_d^p \leftarrow \mathcal{A}_j \nleftrightarrow \tau_i \tag{14}$$

For each sampled state, we call the deterministic planner $\mathcal{P}$ on the three determinized problems $[\mathcal{M}_d^r, \mathcal{M}_d^m, \mathcal{M}_d^p]$ to generate three sequences. These sequences are then merged into a search tree.

## VI. Hindsight Optimization Algorithm

We now propose the following method to create the search tree, which is illustrated in Figure 4. Our method involves constructing a tree where the nodes $v$ are system states $s_i$ and joint actions $a_i$ based on low-level deterministic task schedules. Our approach involves iteratively executing four main steps: 1) selecting likely contingency states, 2) generating new state-action trajectories using reasoning mechanisms, 3) integrating these sequences into the search tree, and 4) backpropagating the goal state node cost in the search tree. The last nodes of the node trajectories are terminating goal states, with the cost of the state being the makespan for that specific scenario. We use bellman backups, as shown in Equation 15, to backpropagate values to the root node. Finally, we return the lowest cost action for a given state $s_i$ to get the task allocation.

$$C(s) = \min_{a \in A} \left[ \frac{\sum_{s'} [T(s,a,s')C(s')]}{\sum_{s'} T(s,a,s')} \right] \tag{15}$$

Our approach is described in Algorithm 2. The first step is to instantiate the root of the policy tree $v_{root}$ as the current state $s_0$ (line 3). Next, the algorithm selects an unexplored state for evaluation (line 5) and generates three determinized problem inputs that reason over how to recover, mitigate, and prevent task failures that occur in the selected state. In practice, when the algorithm first starts, it will select the root node, and we only generate the most-likely outcome

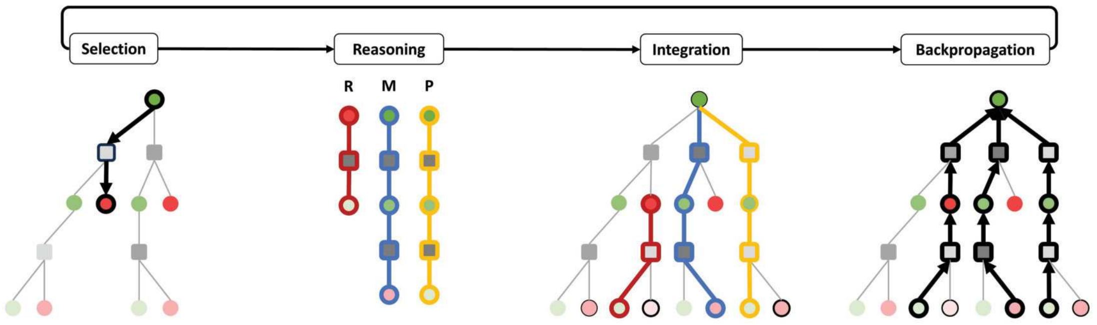


Fig. 4. This diagram illustrates the algorithm. The circular nodes represent states, while the square nodes represent actions. The darker the circle, the higher the probability of that state occurring. Additionally, darker actions indicate more frequently visited actions. The algorithm begins by selecting a contingency state based on less explored actions and higher path probability states. The next step involves generating state-action trajectories using the three reasoning mechanisms as illustrated in Figure 3. These trajectories are then integrated into the search tree, and new contingency states are expanded. Finally, the cost of reaching the goal states is backpropagated to the root node.

**Algorithm 2** HINDSIGHT OPTIMIZATION

```
1:  $s_0$: Current State
2:  function HindsightOptimization($s_0$)
3:      $v_{root} \leftarrow s_0$
4:      while Exploration Computation Budget do
5:          $s_i \leftarrow$ SELECTNODE($v_{root}$)
6:          $[\mathcal{M}_d^r, \mathcal{M}_d^m, \mathcal{M}_d^p] \leftarrow$ DETERMINIZE($s_i$)
7:          $[s,a]_r \leftarrow \mathcal{P}_r(s_i, \mathcal{M}_d^r)$          ▹Recovery
8:          $[s,a]_m \leftarrow \mathcal{P}_m(s_0, \mathcal{M}_d^m)$          ▹Mitigation
9:          $[s,a]_p \leftarrow \mathcal{P}_p(s_0, \mathcal{M}_d^p)$          ▹Prevention
10:         INTEGRATE
11:         BACKPROPOGATE
12:         while Exploitation Computation Budget do
13:         $s_i \leftarrow$ SELECTNODE($v_{root}$)
14:         $[\mathcal{M}_d^r] \leftarrow$ DETERMINIZE($s_i$)
15:         $[s,a]_r \leftarrow \mathcal{P}_r(s_i)$          ▹Recovery
16:         INTEGRATE
17:         BACKPROPOGATE
        return Lowest Cost Action
18: function SELECTNODE($v$)
19:     if $v$ is unselected then
20:         return $v$
21:         else if $v$ is state then
22:             $v = \underset{v' \in \text{children}(v)}{\text{argmin}} \left( C(v) - c\sqrt{\frac{\ln N(v)}{N(v')}} \right)$
23:         return SELECTNODE($v$)
24:     else if $v$ is action then
25:         $v^l = v^l \in \text{leaves}(v)\,\text{argmax}\left(P(v^l)\right)$
26:         return SELECTNODE($v$)
```

determinization $\mathcal{M}_d^r$ from $s_0$ because we cannot reason about task failure mitigation and prevention for already executed tasks. For each determinization, we generate a schedule from the deterministic planner, which is converted into a state-action sequence (lines 7, 8, 9). The state-action sequences are then integrated into the policy tree. During the integration step, if new state-action pairs are added to the tree, we sample other possible contingency states that could occur (line 10). Finally, we backpropagate the value of the terminating goal state node back to the root node (line 11). By executing this algorithm, we start the search tree optimistically and, as we evaluate contingency state scenarios and merge new node sequences, we converge to the expected solution.

We will now explain how we select a contingency state to evaluate, as mentioned in line 17. The algorithm begins with the root node and continues selecting states and actions (nodes) until we reach an unexplored contingency state. If the node is a state, we choose an action using the upper confidence bound heuristic. This helps in exploring new actions that may have been recently added to the state node. If we are at an action node, we select the state node that will lead to the unexplored contingency state with the highest path probability. This node selection procedure enables us to prioritize evaluating contingency states that are more likely to occur.

We also observed that the prevention and mitigation mechanisms work by exploring alternative task allocations, which is essentially exploring the policy space. On the other hand, the recovery mechanism provides coverage to help us converge to the expected value of the policy tree. To make the search more effective, we have divided it into two stages: exploration and exploitation. During the exploration stage (Line 4), all three mechanisms are called to explore different ways of executing task schedules that may prevent and mitigate contingencies. During the exploitation stage, only the recovery mechanism provides sampling coverage for all possible ways of executing task schedules, and all potential paths converge to their expected value.

## A. Insights on Complexity

We now examine the number of times the algorithm calls the recovery, mitigation, and prevention mechanisms to reason over task contingencies by creating scenario determinizations. The mechanisms produce new scenarios for evaluation, i.e., scenarios for if task 1 fails or if task 1 is not executed by agent 1. Therefore, we can determine the number of times the higher level calls the lower level via the reasoning mechanisms by the number of possible scenarios there are to evaluate.

By characterizing the scenario space, we can show that the algorithm terminates and examine how many lower-level calls the algorithm makes if it is allowed to run to completion. Suppose we have $m$ tasks that can result in contingencies, each task can be executed with $n$ agents, and each of those tasks can result in $l$ task states, no matter which agent executes it. We can reason that the total number of times the prevention and mitigation mechanism together is called is the number of possible contingency scenarios to evaluate, which is simply $(nl)^m - 1$ (total number of scenarios minus one because the first scenario is the initial trajectory where everything is optimistically assumed to succeed).

Now let's consider how many times the recovery mechanism would be called. After initializing the search tree with the optimistic schedule, which gives the state-action trajectory, the recovery mechanism would be called for every possible contingency scenario that results from the trajectory. This means that for the initial trajectory, the recovery mechanism would be called for $n^m - 1$ possible contingency scenarios.

The prevention and mitigation mechanisms are to explore alternative ways to allocate and execute tasks and add alternative actions that the system can execute, as illustrated in Figure 3. Essentially, a newly added action is a sub-tree policy root. The recovery mechanism adds state-action trajectory deviations and converges the value of the alternative action (sub-tree root) from an optimistic value to its expected value. The worst-case scenario for how many times the recovery mechanism is called would be if, for every scenario in the scenario space, there was a different unique state-action trajectory to execute. This worst-case would mean the recovery mechanism would be called $(m^n - 1) * (m * o)^n$ times.

Overall, we see that this algorithm will terminate after a finite number of calls, showing that the algorithm is complete. We also see that the number of calls to the lower level is primarily affected by the number of tasks that can result in a contingency. However, from our experiments, we find that for the assembly domain, each lower-level call does not always create a new state-action trajectory, but instead, for many scenarios, the produced state-action trajectory solution is merged into an existing sub-tree. Furthermore, due to the sequential nature of each assembly, a good task allocation to execute at the current state is generally found quickly after a small number of mitigation/prevention calls. In domains where there are many task contingencies that can occur and few constraints, this method may not perform as well.

## VII. Results

Our primary evaluation of our proposed method is the average makespan required to complete a set of assembly tasks. We also examine the scalability of our approach. We evaluate our method via simulations against three other baselines on two contingency task domains: 1) robots can become incapacitated, and 2) robot task failures are detected during the testing stages. We have implemented the low-level constraint programming formulation using the OR-Tools CP-SAT solver. To conduct numerical simulations, we use a Python implementation on a machine with 32 GB RAM and an 8-core 2.1 GHz CPU.

### A. Experimental Setup

*1) Assembly Inspired Domain Problem Setup:* We have constructed a hierarchical task network template for an assembly task inspired by the high-mix, low-volume satellite assembly production domain. The details of the actual tasks as well as contingencies that can occur can be found in [10], [54], and [55]. The assembly begins with four subassemblies that can be completed simultaneously. Once these subassemblies are finished, they are merged to create a complete assembly and then undergo testing. When the assembly passes the functionality test, it is fitted with external parts, and a final test action is performed. We assume the robotic cell can accomplish multiple assemblies, so the final problem input is a multi-assembly HTN, displayed in Figure 5. For the atomic tasks, we randomly generate task durations ranging from 10 to 25 time units and use a team of 5-6 agents to test the system. Furthermore, each task can be completed by 1-2 agents (robot/human or robot and human). For this case study, we assume that robots can perform tasks faster than humans. Therefore, for all human tasks, we scale the duration by 1.5 times. We later provide domain-specific information on how we define the task uncertainty dynamics and the task failure contingency function for both domains.

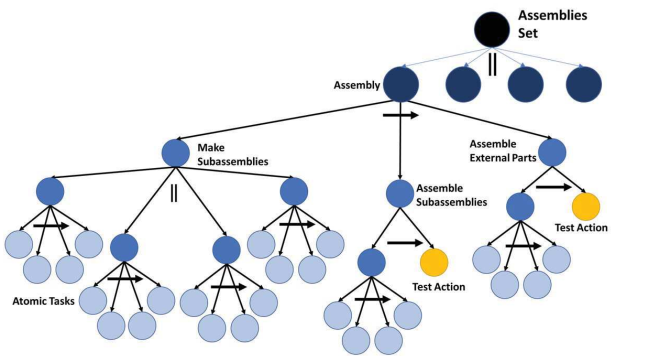


Fig. 5. Hierarchical task network used to represent an example assembly inspired from a satellite assembly domain.

*2) Selection of Contingency Test Problems:* We first identify attributes of problems where proactive contingency management is useful. We conducted experiments on our algorithm using various contingencies that may arise, building on our prior work [10], [11]. Initially, we examined two types of contingencies. The first is when the robot fails to complete a task and must retry it. This contingency can be modeled as a task that has the probability of either succeeding with a nominal duration or with a large delay due to the robot having to recover and try the task again. The second contingency we examined is when a robot fails to complete a task, and a contingency recovery task is added. In this case, the contingency task is that the human must correct the error and complete the task while the robot can be reallocated to other tasks.

We found that proactive contingency management improved makespan when planning for a single assembly, with a 5% improvement in comparison to a reactive approach. This is because assembly tasks are usually sequential in nature, making them more susceptible to bottleneck effects from delays due to contingencies. However, we found for problems with two or more assemblies, the effect of proactive task allocation becomes negligible. In other words, when multiple assemblies are considered concurrently, simply applying reactive replanning when a large task delay occurs, or a task fails yields similar average makespans to a proactive approach. We conclude that proactive task allocation is only useful for contingencies where global replanning, in hindsight, can yield more useful task allocation strategies, and the aforementioned contingencies only require local plan repairs. Therefore, we investigated this approach in two contingency domains: 1) the incapacitated robot domain and 2) the failure testing domain. Both domains have an opportunity for global replanning.

*3) 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 [56].
2) **MCTS:** A centralized Monte-Carlo Tree search using the MMDP formulation to expand on states and actions.
3) **1-Step Lookahead with Optimistic MILP Rollouts:** The formulation from our prior work [10]. We sample states from possible actions that can be taken at the current state and use MILP to generate state-action trajectory rollouts.

All three baselines are online model-based planning approaches. The classical scheduler is a reactive approach that makes decisions one after the other, but it doesn't factor in the uncertainty of the problem. The 1-step lookahead method is a more informed approach that takes into consideration the uncertainty of the problem by sampling failure states. The MCTS approach uses the MMDP framework to plan and can quickly generate estimates by random sampling. However, it may suffer from a large branching factor. We considered including a model-free DRL baseline and adapted the approach proposed in [43] but ultimately excluded it due to its limitations on small-scale problems, where even though the makespan quality underperforms compared to classical approaches like CP-SAT and MILP. Additionally, DRL methods require significant adaptation to handle the model-based uncertainty and dynamic job arrivals in our problem formulation. Since our evaluation focuses on small to medium-sized problem instances, where DRL's scalability advantages are less relevant, we prioritized baselines better aligned with our objectives, such as online model-based planning under uncertainty.

## B. Scalability Analysis

We evaluated the scalability of the lower layer based on the number of tasks in the task set and the level of constraint in the hierarchical task network (HTN). We examined a multi-assembly HTN, consisting of four HTNs as our case study. To adjust the level of constraint, we vary the number of sequential ordering constraints for the lowest subassemblies that have atomic task leaf nodes. The resulting constraint level categories are divided into low (6 total constraint nodes), medium (12 total constraint nodes), and high (18 total constraint nodes). Lastly, we create different task sizes by randomly removing atomic tasks. Our primary objective is to establish a mapping between a problem input's characteristics and the minimum computation time needed to achieve a solution with a makespan that is 95% of an optimal solution. However, obtaining a provably optimal solution can require a significant amount of time. Therefore, we ran the solver for 10 minutes to establish the optimal solution benchmark. We use this approach to create a mapping in our implementation for each domain to determine the computation time for a selected scenario for the low-level evaluations.

TABLE I

This Table Presents the Computation Time Required to Calculate a Schedule That Is, on Average, 95% of an Optimal Solution. The Number of Tasks in the Problem Input Is the Worst-Case Number of Tasks the Lower Level May Have to Optimize for Given Contingencies

| 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 displays the computation results for our implementation of CP-SAT. It is evident that the amount of time required for computation increases exponentially with the number of tasks. Interestingly, the “medium” constrained task set took the longest time to compute. However, the “low” and “high” constrained problems had lower computation times, as 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. This is consistent with job shop scheduling literature, where computational complexity is a function of jobs, machines, and precedence constraints. The scalability of our approach is primarily limited by the complexity of the lower-level solver. Qualitatively, our proposed method is suitable for ‘small’ and ’medium’-sized problems. For example, an instance similar in size to a $10 \times 10$ Flexible Job Shop Scheduling Problem instance, as referenced in [43], would take over an hour to approximate a single solution. Consequently, such instances are not feasible with our current method, as the computation time increases significantly with the number of scenarios.

Modern mathematical programming and constraint programming solvers have the useful feature of using a previous solution to warm-start future computations. For instance, we can use the decision variable values from the previous solution to warm-start the next search when we sample contingency states from an initial trajectory of states and actions. To test this idea, we introduced a presolve step of 120 seconds for the initial solution. We computed the schedules for all next state transitions from $s_0$, with the presolved solution as a starting point, and measured the time needed to find 95% of the solution. Our results show that warm-starting reduces computation time for the lower level.

## C. Results in Incapacitated Robot Scenario

We now consider our first contingency domain. We have designed the domain for incapacitated robots, which becomes

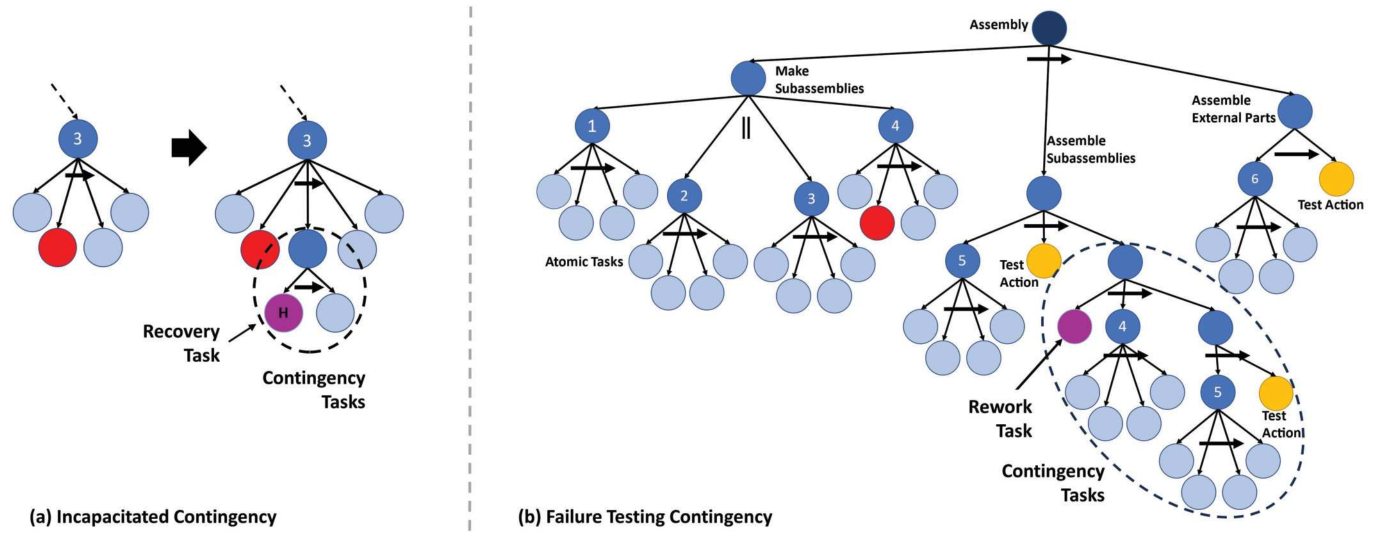


Fig. 6. This figure shows examples of how the hierarchical task network is modified when a contingency occurs for two different scenarios.

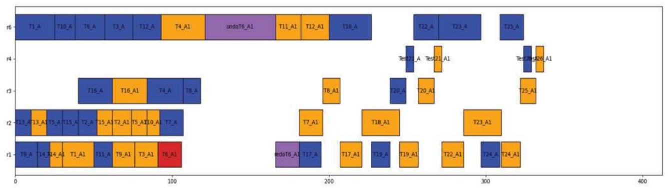


Fig. 7. Robot 1 fails a task (marked in red) and becomes incapacitated. Two contingency tasks (marked in purple) are added to the task set. It can be seen that reactively addressing the contingency creates a bottleneck as other agents can not perform tasks due to sequential dependencies.

TABLE II

AVERAGE MAKESPAN AND STANDARD ERROR, AS WELL AS PERCENTAGE IMPROVEMENT FOR BASELINES IN COMPARISON TO OUR HIERARCHICAL HINDSIGHT OPTIMIZATION-BASED APPROACH (HO) FOR THE INCAPACITATED DOMAIN

| Average Makespan (seconds) & Standard Error | | | | |
|---|---|---|---|---|
| # Tasks | R-MILP | MCTS | 1-Step | HO |
| 26 | (264, 7.98) | (245, 3.77) | (237, 3.56) | (231, 2.27) |
| 52 | (350, 11.09) | (343, 8.43 | (332,7.59) | (314, 5.68) |

| Percentage Improvement over Baselines | | | |
|---|---|---|---|
| # Tasks | R-MILP | MCTS | 1-Step |
| 26 | 13.4% | 5.7% | 2.5% |
| 52 | 10.2% | 8.4% | 5.4% |

unavailable if a robot fails a task. This domain is based on real assembly scenarios where a robot needs to be reset by a human if it fails a task. To handle such a situation, we have defined the contingency function where a recovery task is added to the HTN and task set, which is to be done by the human agent in case of a robot failure. Additionally, the robot can only perform other tasks once the recovery task is completed. A copy of the failed task is also added to the HTN, and the task is set to be redone. Thus, two contingency tasks are added for a task failure. The HTN modification procedure in Figure 6a. We hypothesize that reactively considering this contingency can lead to bottlenecks, requiring a proactive approach. Naively allocating the tasks to the human to prevent task failures can result in an overload of tasks, leading to a loss in efficiency. Without proactively considering the contingency, the schedule can have bottlenecks where no tasks are completed during the recovery period, as shown in Figure 7. Finally, we have defined a team of five agents for this domain: four robots and one human. We have designated four tasks in a single assembly that can only be executed by humans. The remaining tasks are selected to be done by both humans and robots or specialized by the robot. Lastly, we selected four tasks in the assembly that can fail with a task failure probability of 0.10-0.20.

In our proposed assembly domain, we tested our approach against three other methods. We performed interleaved planning and execution by calling the planner each time the system entered a new state. To determine the time limit for lower-level computations, we referred to the results mentioned in Table I. Moreover, we specified the number of times each lower-level mechanism could be called. For instance, the mitigation and prevention reasoning mechanisms were called 20 times, while the recovery reasoning mechanism could be called 40 times. These values were determined empirically, and our algorithm performed well with these parameters.

We conducted an experiment where we increased the number of assemblies being considered. This resulted in an increase in the total number of tasks and contingency tasks to be considered. We performed 30 numerical simulations to estimate the average makespan and standard error, which are reported in Table II. Our findings showed that we were able to achieve a maximum improvement of around 10-13% in average makespan for up to two assemblies (56 tasks). We conducted an Analysis of Variance (ANOVA) test, which confirmed that these improvements were statistically significant, with p = 0.0034. However, when the scheduler considered three assemblies, we found that the opportunity for proactive contingency management became negligible; that is, the average makespan for the reactive and proactive approaches

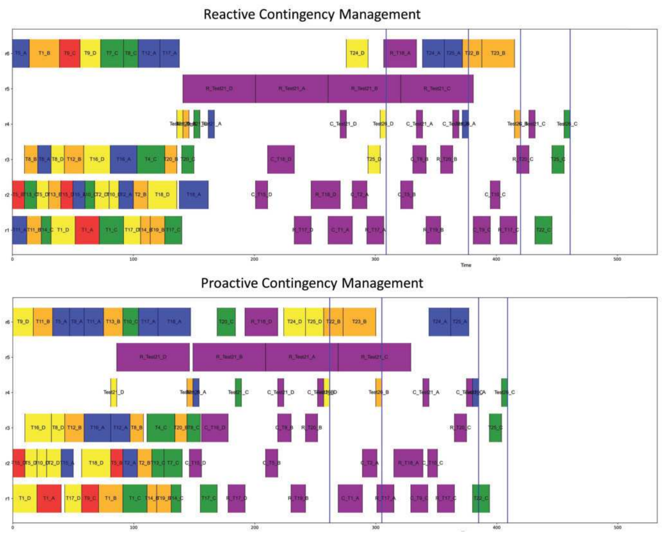


Fig. 8. We show an example scheduling scenario with four assemblies (blue, orange, yellow, green), and the blue line is where an assembly finished. We show four tasks that have failed (marked in red). This requires contingency tasks (marked in purple) to be executed. Reactively managing contingencies results in a bottleneck being formed and causing unnecessary delays. Proactively considering contingencies results reduces the effect of the bottleneck.

was the same. This was due to the bottlenecks being negated because more tasks were available to execute. In this domain, for the two problem instances, we observed that reactive planning consistently underperformed in comparison to our approach, showing that proactive contingency management is useful. Overall, our approach performed as well or better than the benchmarks. The generated schedules showed that the best policy balanced allocating tasks to the human to prevent task failures while not overloading them with all tasks.

## D. Results in Task Failure Testing Scenario

We are also interested in evaluating performance in a domain where it is important to mitigate contingencies. We design this assembly setup domain in the following way. After completing the subassemblies, they are integrated to form a full assembly and undergo a functionality test. If the test is successful, external parts are added to the assembly and undergo a second functionality test. However, if a failure is detected during the testing phase, additional contingency tasks are added to the HTN. The contingency task function is the following: First, the assembly undergoes a rework process where the assembly is disassembled, and the issue is resolved. A rework agent must complete this rework task. Finally, additional reassembly tasks are added to the task set. This contingency function is visualized in Figure 6b. Without proactively managing the contingency, the rework tasks become a major bottleneck in Figure 8.

In this domain, 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. For this domain, we create a 4-assembly HTN (refer to Figure 5) and randomly remove atomic task nodes to get the following 3 case studies of 30 tasks, 50 tasks, and 100 tasks. Lastly, we increase the number of contingency tasks and 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.

TABLE III

Examining Task Size: Average Makespan and Percentage Improvement for Baselines in Comparison to Our Hierarchical Hindsight Optimization-Based Approach (HO) for the Task Testing Domain

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

| 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% |

Empirically, we found the following search parameters to work well: 10 calls for the mitigation and prevention reasoning mechanisms and 30 calls for the recovery reasoning mechanism based on initial experimentation. In total, the lower-level deterministic scheduler was called 50 times. The large scenario space for this domain made 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.

In all scenarios, our method performed as well as or better than the benchmarks. Table III shows the average makespan results for all four methods, including the percentage improvement our approach had compared to the baselines. We observed that reactively planning consistently underperformed compared to our approach. Furthermore, Figure 8 shows a comparison between a reactive and proactive approach for a specific scenario. We observed a simple behavior that emerged where testing actions are completed as soon as possible so that if a contingency is detected, agents can continue to work on other assemblies while the contingency is resolved. This highlights the benefit of proactively managing contingencies to improve makespan.

## E. Role of Recovery, Mitigation, and Prevention Components

We performed an ablation study to investigate the impact of various components of our search algorithm. The results showed that removing the recovery mechanism significantly diminished the algorithm's performance. This outcome is expected, as the recovery mechanism plays a role in covering the possible scenario space when constructing the search tree and ensuring that the cost of a task allocation action node converges to the expected cost. Consequently, we examine the impact of using just the mitigation mechanism or the prevention mechanism with the recovery mechanisms.

We first examine the 26-task problem of the incapacitated robot scenario. When we used only the mitigation reasoning mechanism, the average makespan quality was reduced by 5.6%, while with only the prevention mechanism, the makespan quality was reduced by only 1.9%. We next examined the 30 task problems for the task failure testing domain. When only the mitigation reasoning mechanism was used, the average makespan quality decreased by 1.8%. However, when only the prevention mechanism was used, the makespan quality decreased by 6.8%.

This indicates that the performance of the search components is problem-domain dependent. We observe this in the incapacitated robot domain, where the prevention mechanism is more dominant for managing contingencies. In contrast, the mitigation mechanism is more relevant in the task testing domain. This is because, in the incapacitated domain, it is more beneficial to prevent the incapacitation from happening, but if it doesn't happen, there are not many tasks added to fix the contingency, so there is less opportunity to mitigate the effect of contingency tasks. In sharp contrast, there are many more contingency tasks added in the testing domain, meaning there are more opportunities for mitigating its effect.

## VIII. Conclusion

We have developed a new approach for multi-robot task allocation that deals with uncertainty by decoupling the task scheduling and decision-making under uncertainty problems. Our results show that using constraint programming solvers to solve a mixed integer program is an effective candidate for creating low-level task schedules. Our proposed framework is useful for addressing contingencies where contingency tasks can cause significant delays. By reasoning in hindsight, we can generate schedules that proactively recover, mitigate and prevent contingencies. We tested this framework in two domains and found that a proactive approach can significantly improve makespan by 8-15%, which is significant for industrial applications.

*Limitations and Future Work:* There are two main issues that could limit the effectiveness of this approach. The first limitation is that while we find constraint programming methods suitable for solving the task allocation and scheduling problem, this approach will become a computational bottleneck as we consider very large task and agent sizes. We can improve the performance of the lower-level solver by incorporating domain-specific heuristics or priority dispatching rules to compute the deterministic task allocation. Future work will consider integrating deep reinforcement learning approaches for approximating the lower-level solutions for large problems. Incorporating deep reinforcement learning directly or hybridizing it with exact solving methods for the lower-level solver will make this approach scalable to larger problem sizes. The second limitation is as the number of tasks that can fail increases, leading to more possible scenarios, the number of contingencies that need to be sampled grows exponentially, leading to a second computational bottleneck. In the future, one could prioritize the evaluation of relevant contingencies by learning a contingency selection heuristic, such as evaluating the contingencies that may have a larger delay effect first. Lastly, we also make simplifying assumptions by ignoring task switch, setup time, and resource costs. Due to these assumptions, we found that when we tested large task sizes when a contingency happened, the makespan could be unaffected by any bottlenecks due to the ability to immediately switch to another assembly. However, the cost of addressing the contingency is much higher due to the aforementioned costs required to switch to a different assembly. Modeling these costs can help further match the real-world scenario.

## References

[1] A. Peterson, "High-mix/low-volume manufacturers are a sweet spot for collaborative robots," NIST, Gaithersburg, MD, USA, Tech. Rep., Jul. 2020. [Online]. Available: https://www.nist.gov/blogs/manufacturing-innovation-blog/high-mixlow-volume-manufacturers-are-sweet-spot-collaborative

[2] J. H. Kang, N. Dhanaraj, O. M. Manyar, S. Wadaskar, and S. K. Gupta, "A task allocation and scheduling framework to facilitate efficient human–robot collaboration in high-mix assembly applications," in *Proc. Int. Manuf. Sci. Eng. Conf.*, vol. 88117, 2024, pp. 1–12.

[3] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *Proc. IEEE 18th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2022, pp. 2110–2117.

[4] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 3559–3564.

[5] S. Chen, Q. Nguyen, and S. K. Gupta, "A computationally efficient approach to account for stochastic delays in multi-robot task allocation in a proactive manner," in *Proc. IEEE 20th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2024, pp. 3095–3102.

[6] S. Shriyam and S. K. Gupta, "Incorporation of contingency tasks in task allocation for multirobot teams," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 2, pp. 809–822, Apr. 2020.

[7] S. Shriyam and S. K. Gupta, "Incorporating potential contingency tasks in multi-robot mission planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 3709–3715.

[8] S. Al-Hussaini, J. M. Gregory, and S. K. Gupta, "Generating task reallocation suggestions to handle contingencies in human-supervised multi-robot missions," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 1, pp. 367–381, Jan. 2023.

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

[10] N. Dhanaraj, S. V. Narayan, S. Nikolaidis, and S. K. Gupta, "Contingency-aware task assignment and scheduling for human–robot teams," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 5765–5771.

[11] N. Dhanaraj, J. H. Kang, A. Mukherjee, H. Nemlekar, S. Nikolaidis, and S. K. Gupta, "Multi-robot task allocation under uncertainty via hindsight optimization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 16574–16580.

[12] M. Gombolay et al., "Human-machine collaborative optimization via apprenticeship scheduling," *J. Artif. Intell. Res.*, vol. 63, pp. 1–49, Sep. 2018.

[13] G. Michalos, J. Spiliotopoulos, S. Makris, and G. Chryssolouris, "A method for planning human robot shared tasks," *CIRP J. Manuf. Sci. Technol.*, vol. 22, pp. 76–90, Aug. 2018.

[14] A. Roncone, O. Mangin, and B. Scassellati, "Transparent role assignment and task allocation in human robot collaboration," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1014–1021.

[15] W. Wang, R. Li, Z. M. Diekel, and Y. Jia, "Robot action planning by online optimization in human–robot collaborative tasks," *Int. J. Intell. Robot. Appl.*, vol. 2, no. 2, pp. 161–179, Jun. 2018.

[16] S. Wang, Y. Liu, Y. Qiu, S. Li, and J. Zhou, "An efficient distributed task allocation method for maximizing task allocations of multirobot systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 3, pp. 1–15, Jul. 2024.

[17] D.-H. Lee, S. A. Zaheer, and J.-H. Kim, "A resource-oriented, decentralized auction algorithm for multirobot task allocation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 4, pp. 1469–1481, Oct. 2015.

[18] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 876–888, Jul. 2015.

[19] M. Aggravi, G. Sirignano, P. R. Giordano, and C. Pacchierotti, "Decentralized control of a heterogeneous human–robot team for exploration and patrolling," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3109–3125, Oct. 2022.

[20] X. Bai, A. Fielbaum, M. Kronmüller, L. Knoedler, and J. Alonso-Mora, "Group-based distributed auction algorithms for multi-robot task assignment," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 2, pp. 1292–1303, Apr. 2023.

[21] L. Zhang, J. Zhao, E. Lamon, Y. Wang, and X. Hong, "Energy efficient multi-robot task allocation constrained by time window and precedence," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–12, 2024.

[22] H. Aziz, A. Pal, A. Pourmiri, F. Ramezani, and B. Sims, "Task allocation using a team of robots," *Current Robot. Rep.*, vol. 3, no. 4, pp. 227–238, Aug. 2022.

[23] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–31, Apr. 2019.

[24] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Int. J. Adv. Robot. Syst.*, vol. 10, no. 12, p. 399, Dec. 2013.

[25] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robot. Auto. Syst.*, vol. 90, pp. 55–70, Apr. 2017.

[26] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.

[27] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *Int. J. Robot. Res.*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.

[28] E. G. Jones, M. B. Dias, and A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," *Auto. Robots*, vol. 30, no. 1, pp. 41–56, Jan. 2011.

[29] 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 Trans. Robot.*, vol. 39, no. 2, pp. 1087–1105, Apr. 2023.

[30] 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," *Robot. Auto. Syst.*, vol. 131, Sep. 2020, Art. no. 103560.

[31] M. L. Pinedo, *Scheduling*, vol. 29. Cham, Switzerland: Springer, 2012.

[32] B. Çaliş and S. Bulkan, "A research survey: Review of AI solution strategies of job shop scheduling problem," *J. Intell. Manuf.*, vol. 26, no. 5, pp. 961–973, Oct. 2015.

[33] H. C. Lau, M. Sim, and K. M. Teo, "Vehicle routing problem with time windows and a limited number of vehicles," *Eur. J. Oper. Res.*, vol. 148, no. 3, pp. 559–569, Aug. 2003.

[34] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 3, pp. 462–467, Jan. 2017.

[35] V. C. S. Wiers, "A review of the applicability of OR and AI scheduling techniques in practice," *Omega*, vol. 25, no. 2, pp. 145–153, Apr. 1997.

[36] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 12618–12624.

[37] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast scheduling of robot teams performing tasks with temporospatial constraints," *IEEE Trans. Robot.*, vol. 34, no. 1, pp. 220–239, 2018.

[38] Y. Cheng, L. Sun, C. Liu, and M. Tomizuka, "Towards efficient human–robot collaboration with robust plan recognition and trajectory prediction," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2602–2609, Apr. 2020.

[39] Y. Cheng, L. Sun, and M. Tomizuka, "Human-aware robot task planning based on a hierarchical task model," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1136–1143, Apr. 2021.

[40] Y. Cheng and M. Tomizuka, "Long-term trajectory prediction of the human hand and duration estimation of the human action," *IEEE Robot. Autom. Lett.*, vol. 7, no. 1, pp. 247–254, Jan. 2022.

[41] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human–robot collaboration," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 5469–5476.

[42] J. Leu, Y. Cheng, C. Liu, and M. Tomizuka, "Robust task planning for assembly lines with human–robot collaboration," 2022, *arXiv:2204.07936*.

[43] K. Lei et al., "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 205, Nov. 2022, Art. no. 117796.

[44] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1600–1610, Feb. 2023.

[45] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Y. Meng, and L. M. Qian, "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 20, no. 1, pp. 1007–1018, May 2023.

[46] R. Liu, R. Piplani, and C. Toro, "Deep reinforcement learning for dynamic scheduling of a flexible job shop," *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4049–4069, 2022.

[47] S. Ma, J. Ruan, Y. Du, R. Bucknall, and Y. Liu, "An end-to-end deep reinforcement learning based modular task allocation framework for autonomous mobile systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 1519–1533, 2025.

[48] R. Bezerra et al., "Heterogeneous multi-robot task allocation for garment transformable production using deep reinforcement learning," in *Proc. IEEE 19th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2023, pp. 1–8.

[49] T. Vodopivec, S. Samothrakis, and B. Ster, "On Monte Carlo tree search and reinforcement learning," *J. Artif. Intell. Res.*, vol. 60, pp. 881–936, Dec. 2017.

[50] N. Dhanaraj, R. Malhan, H. Nemlekar, S. Nikolaidis, and S. K. Gupta, "Human-guided goal assignment to effectively manage workload for a smart robotic assistant," in *Proc. 31st IEEE Int. Conf. Robot Human Interact. Commun. (RO-MAN)*, Aug. 2022, pp. 1305–1312.

[51] L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv, "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem," *Comput. Ind. Eng.*, vol. 142, Apr. 2020, Art. no. 106347.

[52] A. Fekih, H. Hadda, I. Kacem, and A. B. Hadj-Alouane, "Mixed-integer programming and constraint programming models for the flexible job shop scheduling problem," in *Proc. Int. Conf. Artif. Intell. Ind. Appl.* Cham, Switzerland: Springer, Jan. 2023, pp. 110–122.

[53] 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," *Eur. J. Oper. Res.*, vol. 302, no. 3, pp. 874–891, Nov. 2022.

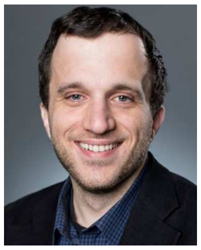
[54] N. Dhanaraj et al., "A human robot collaboration framework for assembly tasks in high mix manufacturing applications," in *Proc. Int. Manuf. Sci. Eng. Conf.*, vol. 87240, 2023, pp. 1–17.

[55] 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 *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 15585–15592.

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

**Neel Dhanaraj** (Student Member, IEEE) received the B.S. and M.S. degrees in mechanical engineering from Worcester Polytechnic Institute, Worcester, MA, USA. He is currently pursuing the Ph.D. degree with the Viterbi School of Engineering, University of Southern California, Los Angeles, CA, USA, under the guidance of Prof. S. K. Gupta. His research interests include task allocation/scheduling and motion planning for human-robot teams.

**Heramb Nemlekar** (Member, IEEE) received the M.S. degree in robotics engineering from Worcester Polytechnic Institute, Worcester, MA, USA, in 2019, and the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2023. He is currently a Post-Doctoral Associate with the Department of Mechanical Engineering, Virginia Tech, Blacksburg, VA, USA. His research interests include transfer learning of human preferences, representation learning, and human-robot co-adaptation.



**Stefanos Nikolaidis** (Member, IEEE) is currently an Associate Professor of Computer Science and the Fluor Early Career Chair of Engineering at the University of Southern California, Los Angeles, CA, USA. His current research interests include robotics, human-robot interaction, quality diversity optimization, and machine learning.

**Satyandra K. Gupta** (Fellow, IEEE) holds Smith International Professorship at the Viterbi School of Engineering, University of Southern California, and serves as the Director of the Center for Advanced Manufacturing. He has published more than 400 technical articles in journals, conference proceedings, and edited books. His research interests include physics-informed artificial intelligence, computational foundations for decision-making, and human-centered automation. He is a fellow of American Society of Mechanical Engineers (ASME), the Solid Modeling Association (SMA), and the Society of Manufacturing Engineers (SME).