2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

# Contingency-Aware Task Assignment and Scheduling for Human-Robot Teams

Neel Dhanaraj[1], Santosh V. Narayan[1], Stefanos Nikolaidis[1], and Satyandra K. Gupta[1]

Abstract— We consider the problem of task assignment and scheduling for human-robot teams to enable the efficient completion of complex problems, such as satellite assembly. In high-mix, low volume settings, we must enable the human-robot team to handle uncertainty due to changing task requirements, potential failures, and delays to maintain task completion efficiency. We make two contributions: (1) we account for the complex interaction of uncertainty that stems from the tasks and the agents using a multi-agent concurrent MDP framework, and (2) we use Mixed Integer Linear Programs and contingency sampling to approximate action values for task assignment. Our results show that our online algorithm is computationally efficient while making optimal task assignments compared to a value iteration baseline. We evaluate our method on a 24-task representative assembly and a real-world 60-task satellite assembly, and we show that we can find an assignment that results in a near-optimal makespan.

## I. Introduction

New advances in multi-robotic systems are helping realize intelligent automation of very complex assembly problems, such as a collaborative robotic cell for satellite assembly [1] and the Boeing 777 fuselage upright autonomous build process [2]. However, these robotic cells have not reached the technology readiness level where all tasks can be automated; tasks will still need a human operator in the loop [3]. Intuitively this is more than acceptable, considering that humans and robots have complementary strengths. Robots can accomplish supportive and repeatable tasks, while humans can execute tasks that require fine manipulation or dexterity [4]. In high-mix, low-volume situations where the human-robot team must react to changing task requirements, an intelligent system needs to be able to assign tasks to appropriate agents in the team and then schedule these tasks.

Organizing a human-robot team efficiently to accomplish tasks for challenging applications in an uncontrolled environment is difficult because there is compounding uncertainty in the system, tasks, and environment. Unexpected events can cause delays or even failures in the system [5]–[8]. Contingencies, such as the robot irrecoverably failing a task, will require the human to intervene to recover the system and execute the task themselves. Delays in the tasks executed by the human can decrease the teaming efficacy between the human and the robot.

Many different approaches for task assignment and scheduling of human-robot teams have been proposed with different considerations. A trending approach that is being explored in the planning community is solving human-robot task assignments and scheduling problems as a Mixed Integer Linear Program (MILP). It is advantageous to formulate such problems as MILPs because there are numerous openly available integer and constraint programming solvers that can solve a given program and return a solution very quickly. The quick solution return also allows the ability to quickly replan when the system ends up in an unforeseen state. Though there are approaches to incorporate uncertainty directly into the MILP formulation [9], [10], such approaches take longer to solve and can not effectively account for problem settings with multiple sources of uncertainty. In most prior work, contingencies are not explicitly considered. If the system does not explicitly consider contingencies when generating a policy, there will be bottlenecks during task execution due to these events, leading to inefficiency.

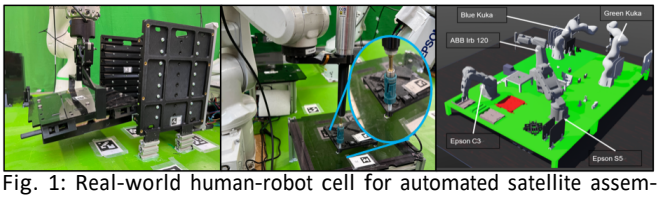

Fig. 1: Real-world human-robot cell for automated satellite assembly and potential contingencies that could occur during execution, i.e., a battery module has gotten stuck, or a screwing operation has failed.

Instead, we can formulate this problem as a Multi-Agent Concurrent Markov Decision Process with system states and task assignment actions and compute an optimal policy. This allows the system to formally incorporate different sources of uncertainty and compute possible state transitions. However, for any meaningful-sized set of tasks, it becomes infeasible for the system to generate a complete state transition diagram with all possible contingencies and then use value iteration techniques to find the optimal policy. Our key insight is that we can quickly populate MILP solutions in the form of state action paths and then sample contingencies that may occur along these sequences. New state action paths can be generated from the MILP solution for these contingencies. Effectively, we use the strength of a MILP solver and the CoMDP formulation to approximate the state transition diagram efficiently. We can then perform a value backup and generate a policy.

In summary, our contributions are: (1) A Multi-Agent Concurrent MDP Framework with durative tasks that accounts for task failure and stochastic durations. (2) A MDP to MILP formulation that takes in an MDP model and state and gives state action paths. (3) online Action Selection that integrates the benefits of the MDP state transition computation and the MILP solution return to evaluate contingencies and approximate a policy. (4) Evaluation on a real assembly that

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

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160806

shows how this method would be implemented on a real-world satellite assembly robotic cell.

## II. Related Works

There is extensive literature for planning, task assignment, and scheduling for human-robot teams [11]–[20]. These approaches have used hybrid MILP, constraint programming, multi-level optimization, and auction-based methods. Recently, prior work has shown that formulating the problem as a Mixed Integer Linear Program is a promising approach for solving schedules with complex constraints such as temporal and spatial constraints [21]–[23] and hierarchical task network ordering and precedence constraints [10], [24]–[27]. The strength of using a MILP formulation is the availability to use many powerful open source solvers that can solve the problem and return the solution quickly [9].

Though most methods use deterministic assumptions, research has shown that uncertainty originating from task failures and human task execution can be directly incorporated into the MILP formulation using 2-stage stochastic programming and chance constraints [28], [29]. Cheng et al. [9] account for stochastic task durations for human actions using a chance-constrained MILP formulation. By explicitly accounting for the human uncertainty, the author's results show a more robust task assignment for different agents and an improved makespan. In contrast, considering more complex uncertainty interaction is very difficult for MILP, which requires more specialized solutions.

Researchers have explored generalized probabilistic planning approaches [30] for complex stochastic domains using Concurrent, and Multi-agent MDP formulations [31], [32]. These approaches formally capture the uncertainty dynamics in multi-agent teams; however, finding a policy for these MDPs is generally intractable. Therefore multiple approaches have been proposed to approximate the policies. Weld et al. [33], [34] proposes a concurrent MDP (CoMDP) formulation for task planning that captures concurrency in action execution, probabilistic task success outcomes, and stochastic durations. They show that a system can approximate a policy using real-time dynamic programming (RTDP) approaches. Other approaches such as multi-agent reinforcement learning and online planning have also shown that they can be used to solve sequential decision-making for multi-agent teams [35]–[38].

## III. Problem Formulation

We focus on the planning problem for task sets consisting of a set of atomic tasks completed using a multi-human-multi-robot team. Robots are agents that have different task specializations. We model robots as agents that complete a task in a deterministic nominal time. However, we must also consider the probability that the robot may irrecoverably fail a task at some time step between the start and end times. The human must recover the failed task in such states.

The human is an agent who can complete all tasks in the task set; however, there is more variability in task durations due to correcting mistakes and fatigue [10], [39]. We model the delays that may occur in humans using a stochastic durative model. We consider the stochastic nature inherent in the multi-human-multi-robot team planning problem by accounting for task failures and stochastic durations.

Task Set State $x = [x_1, x_2, ...x_n]$: The state of the task set is defined by the completion status of each atomic task $x_i$. The state variable of an atomic task can either be notAttempted, currentlyAttempting, succeeded or failed. The state space with n tasks is therefore represented using factored state space representation, i.e. $X = (dom(X_1) \times ... \times dom(X_n))$.

Hierarchical Task Network (HTN): As formulated in [24], we represent a task set with atomic tasks using a sequential/parallel HTN. The root node represents the set of tasks, and the leaf nodes represent the tasks. All other nodes represent sub-tasks. Each sub-task node can be categorized into three types: Sequential, Parallel, and Independent. The type indicates the relationship between the sub-task and the child nodes.

Tasks that are children to a sequential sub-task node must be individually executed, from left to right, and this node type captures ordering constraints found in assemblies. For independent sub-tasks, tasks must be individually executed; however, there is no ordering constraint. Lastly, tasks of a parallel sub-task can be executed at the same time by different agents.

Duration Model: In this work, we approximate discrete distributions from continuous distributions to model the temporal duration of a task. All tasks have an associated nominal time and max completion time that a task will succeed by. We define a max completion time for each task in order to put an upper bound on the distribution. Also, tasks will have probability distributions for what time steps the agent may fail the task.

This work simplifies the human duration model because we assume humans can not fail a task. The stochastic duration of human task execution is modeled using a lognormal distribution $\text{Lognormal}(\mu, \sigma^2)$, where $\mu$ is the nominal completion time and $\sigma$ accounts for the delay that can occur. This model is justified in [25].

For the robot duration model, the max time is equal to the nominal time if the robot succeeds in task completion. For a robot failing a task, we used a skewed normal distribution $\text{Skew}(\mu, \sigma, \alpha)$ to describe when the task may fail.

Multi-Agent Concurrent MDP $(N, S, A, T, C, \gamma)$ : The problem of finding the best task assignment can be modeled as a Multi-Agent CoMDP where N is a set of available agents $A_i$, S is the state space of the entire system, A(s) is the joint task assignment combination space of a particular state, $T(s, a, s')$ : is the transition function, C : is the cost of taking an action at a state and $\gamma$ is the discount factor.

State $s = (x, y, z), s \in S$: The state of the entire system is described by three variables:

1) Task Set State $x = [x_1, x_2, ...x_n]$
2) A set of tuples consisting of the current tasks be executed and the current time-step $\delta_{x_i}$ in the action duration i.e. $y = \{(x_i, \delta_{x_i})..(x_n, \delta_{x_n})\}$
3) A set of tuples consisting of current agent assignments $z = \{(A_i, x_j)..(A_m, x_n)\}$

All states between the start and goal state are decision epochs where there are available tasks and unassigned agents, and an agent assignment decision can be made.

Action $a \in A$: An agent-task assignment at a state is a task assignment decision taken at a decision epoch, i.e., $(A_i, x_j)$. There can be multiple task assignments that can be simultaneously taken. We refer to the set of task-agent assignments taken in a state as an action a. The set of all possible task-agent assignment combinations is a function of the HTN model as well as the agent's specialization model.

State Transition $T(s, a, s')$: When the system takes an action at an initial state, the state transitions to the next decision epoch, where the earliest terminating task has finished, an agent is available, and the system has the opportunity to make a new decision. Specifically, the MDP transition captures the effect of two discrete transitions $s \rightarrow s^{l} \rightarrow s'$. First, given an action and an initial state, the system assigns agents to available tasks. The initial state's current task and agent state get updated with the new assignment, and s transitions to an intermediary state $s^{l}$. Next, an environment transitions T transitions the state to $s'$ where the task that would have finished earliest gets completed, and an agent becomes available with the possibility to be assigned to another task.

Cost model $C(s, a) \equiv \delta_{next}$: The cost of taking an action is the duration $\delta_{next}$ that it takes to transition from the current state to the next decision epoch.

Overview of Approach: Given an HTN, task failure, and duration model, the goal is to compute a policy that minimizes the action value function $Q^{\pi}$. Such a policy would find task-agent assignments while also accounting for any contingencies to minimize the discounted expected makespan of the task. Our approach is an online method for estimating a value function for all possible actions at a state and returning the lowest cost action. We intelligently sample from states where a task fails, which we refer to as contingencies. The method to compute these possible next states is presented in Section IV. For contingencies that have a higher probability of occurring, we use a MILP solver to return an optimal task assignment schedule while assuming deterministic outcomes and durations, which we present in Section V. Lastly, we evaluate the optimal action to take by building a state transition diagram of state-action sequences; this algorithm is presented Section VI.

## IV. Computing the Candidate State Transitions

In order to explain how to find candidate state transitions after the system takes an action, we first explain the candidate state transition computation by assuming a system whose tasks either succeed or fail after a deterministic duration. Let us consider a state where the system executes a set of tasks y. In order to find the next decision epoch, we find the earliest time step that a subset of tasks in y will terminate, causing the state to change. The candidate state transitions then become the set of all possible combinations of success or failure of the tasks. Lastly, for the tasks that did not terminate, the current time step in the action duration variable increases by the elapsed duration. A four-state system is illustrated in

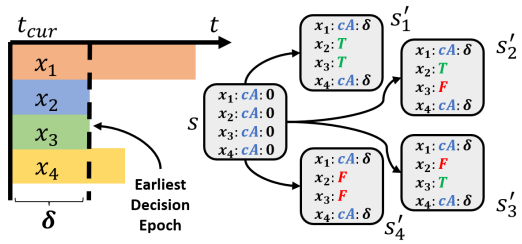


Fig. 2: Consider an example state where four tasks are currently being executed. The earliest next decision epoch will be when $x_2$ and $x_3$ terminate. At this decision epoch, we find there are four candidate state transitions variations that could occur. The tasks that do not terminate in the decision epoch have their current time-step updated by the elapsed time $\delta$ (cA is currently attempting, T is succeeded, and F is failed).

Figure 2.

So far, we have assumed deterministic durations, which do not reflect real-world scenarios where task durations are more variable. When considering stochastic durations, whose distributions depend on the success or failure of the task, the transition state computation becomes complex. We need to consider all time steps and all possible variations of decision epochs that could occur. By explicitly considering stochastic durations coupled with the task outcome distributions, the system can take advantage of transition states where a task could have failed prematurely or taken longer than the nominal time when computing the value function.

**Algorithm 1** Expected Value Candidate Transitions

```
s: Initial state
a: Applied action at state
p_t = P(x'|Xsub_t, t) : prob that Xsub_t terminates after t resulting in x'
---
function COMPUTE-TRANSITION-FUNC(s=(x,y,z),a)
    y^l ← y ∪ {(x_i, 0) | x_i ∈ a}
    z^l ← z ∪ a
    mintime ← min_{(x,δ)∈y^l} {min remaining times for x_i}
    maxtime ← min_{(x,δ)∈y^l} {max remaining times for x_i}
    X_t ← Set of tasks that could terminate in between mintime and maxtime at state s
    for non-empty subsets Xsub_t ⊆ X_t do
        W ← {(x', p_x) | x' is the task set state; p_x is the probability that Xsub_t terminates yielding x'}
        for (x', p_x) ∈ W do
            p_int = Σ_{t∈[mintime,maxtime]} p_t
            t_exp = Σ_{t∈[mintime,maxtime]} (p_t ∗ t)/p_int
            y' ← {(x_i, δ + t_exp) | (x_i, δ) ∈ y^l, x_i ∈ Xsub_t}
            z' ← {(A, x_i) | (A, x_i) ∈ z^l, x_i ∈ Xsub_t}
            p_s ← p_t ∗ p_x
            insert ((x', y', z'), p_s) in output
    return output
```

Finally, we adapt our algorithm for candidate state transition computation from [33], which presents a method for

single-agent CoMDPs with stochastic durations. Algorithm 1 describes the computation. For a state and an active input, the algorithm updates the state's current task and agent assignments; we call this updated state an intermediary state (Line 6,7). We next determine the earliest time that a task could end (mintime) and the earliest time a task will definitely end (maxtime), and then compute the set of tasks that could terminate in between the mintime and maxtime (Line 8,9,10). The algorithm has to consider all possible subset combinations of tasks $Xsub_t$ that may terminate within the considered time interval, the resulting next states, and the associated transition probabilities. Given our state space representation, the exact computation of the transition function would generate all possible state outcomes for each time step in the considered time interval. For an intermediary state where e actions could terminate in an interval with o time-steps, the resulting number of transitions could easily reach the upper limit of $2^e - o - 1$ states leading to a huge branching factor. Instead, we compute the expected value of the task duration for each state transition when computing how much forward to move in time for the next transition.

The algorithm first calculates the probability that $Xsub_t$ completes at time-step t resulting in the next state $x'$ for each time-step integer. The algorithm then sums the probabilities, giving the probability $p_{int}$ that $Xsub_t$ will terminate at some time step resulting in $x'$ within the entire interval (Line 14). Lastly, we calculate the expected duration value for that transition and update the elapsed time for the intermediary state. The algorithm returns a list of the possible successor states and corresponding probabilities that the state will transition to $x'$ at some time-step within the interval. The computed output captures the effect of stochastic durations and task failure probabilities without considering every possible state transition for every time step.

## V. Using a MILP to Insert Paths in State Transition Diagrams

We use a MILP to generate optimal plans while assuming deterministic outcomes and expected durations. This paper adapts the MILP from a standard time-index model. The exact formulation is given in [40]. The goal of the MILP is to find the best agent-task assignments $x_{A_i,j}$ for i agents and j tasks, as well as task start times $t_{start,j}$ in order to minimize the makespan variable t. Furthermore, we want to reduce unnecessary idle time and make decisions at the earliest opportunity, so we also must minimize the start time variables. Therefore, we define our cost function in Equation 1 as the weighted sum of the makespan value t and the summation of start times s. Lastly, the solution will be subject to ordering and precedence constraints for the tasks that are derived from the HTN.

We must now modify the initial formulation to consider in-between states where some tasks are completed, or agents may currently be executing a task. The constraints in Equation 2 capture information from the set of currently executing tasks y by assigning the corresponding agent-task decision variable to be 1. The task $t_{start}$ variable is set to be zero, and the $t_{nom,A_i}$ duration is set to equal the original task nominal duration subtracted by the current time-step in the task duration. Lastly, if the task is completed ($x_i \in X_{complete}$), then the $t_{nom,A_i}$ duration value is set to zero, effectively removing the task decision variable from the computation of the objective.

$$\min At + Bs \tag{1}$$

$$(x_{A_{i,j}} = 1,\ t_{start,j} = 0,\ t_{nom,A_{i,j}} = t^{original}_{nom,A_j} - \delta_{x_j}) : x_{A_{i,j}} \in y \tag{2}$$

$$t_{nom,A_{i,j}} = 0 : x_i \in X_{complete} \tag{3}$$

## VI. Contingency-Aware Action Selection

The key idea for evaluating contingency states is to consider the probability that a state sequence, whose leaf node is the contingency state, occurs given the current state. This probability is the compound probability $p^c$ that the group of states in the sequence occurs. We show an example state transition diagram in Figure 3 where each node is a state, and each edge is an action and possible transition edge. The green nodes are states where tasks are successfully completed, while the red nodes are contingency states where the task failed. The nodes highlighted in yellow are states where the MILP solver evaluated and generated a new sequence that was connected to the goal state.

Let us first conceptually explain the method for evaluating contingencies through an example shown in Figure 3. Consider that in this system, when an action is taken at each state, there is a 0.9 probability that a task will successfully complete and a 0.1 probability that it will fail, resulting in a contingency state. First, given that the system takes $a_1$ at the start state, we find the resulting transition states and corresponding compound probability and then add the tuple $(p^c, s')$ to a queue. We then pop the next state with the highest $p^c$ from the queue. In this example, the next state is state 1, and $p^c = 0.9$, and we then run our MILP solver to generate a state sequence from state 1. This sequence is then connected from state 1 to the goal state. Lastly, we traverse the state-actions in the sequence and find new contingency states and associated $p^c$ values, which we then add to the queue. The method continuously keeps evaluating contingency states until the queue is empty. In our example, the next contingency state that would be evaluated in state 2 with $p^c = 0.1$, and a new state sequence would be generated and connected between state 2 and the goal state.

Without a bound on which contingency states should be evaluated, our method would build a complete state transition diagram and generate a complete solution at the cost of computation time. Instead, we introduce a parameter $p^c_{min}$, which defines the smallest compound probability for a state that should be evaluated. Intuitively, this parameter dictates that the algorithm evaluate contingency states that are likelier to occur than $p^c_{min}$. In the previous example, the $p^c_{min} = .05$; therefore, contingency states with a compound probability above .05 are evaluated by the MILP solver. Once all qualifying states are evaluated, and the state transition diagram is constructed, we perform a value backup, and the algorithm returns the state action pair with the lowest cost.

**Algorithm 2** Policy Approximation algorithm

```
1:  s: Current State
2:  LowestActionQVal: Ordered list actions to be evaluated
3:  QTable: Q Table for all encountered state action pairs
4:  function PLAN(s)
5:      A ← get-all-actions(s)
6:      for a ∈ A do
7:          (Q_est.Traj) ← MILP-ESTIMATE(s,a)
8:          Insert (Q_est, s, a) in LowestActionQVal
9:      while Not Interrupted do
10:         for (Q_est, s, a) in LowestActionQVal do
11:             if Best_Q < Q_est then
12:                 Remove s,a from LowestActionQVal
13:             QVals ← EVAL-CONTINGENCIES(s, p^c_min)
14:             Value Backup of Q(s,a) with QVals
15:             if Best_Q > Q(s,a) then
16:                 Best_Q = Q(s,a)
17:                 Best_Action = a
18:             Incrementally reduce p^c_min
        return Best Action
```

We now formally describe our policy generation method shown in Algorithm 2. The algorithm first gets all possible actions that can be taken at the current state (Line 4). For each action and resulting intermediary state, we run the MILP-Estimate, which returns a state sequence and estimated Q value. This output is then stored in an ordered action list (Line 7,8). The algorithm first evaluates contingencies for potentially good actions by ordering their estimated Q value. The algorithm can then prune out bad actions whose optimistic estimate is worse than the best-expected Q value seen so far (Line 11,12).

The algorithm then gets the state-action pair with the lowest initial estimated Q value and evaluates all encountered contingencies whose compound probability is above $p^c_{min}$. This returns all estimated Q values for the encountered state-action pairs in the evaluated contingency trajectories. The algorithm then performs a value backup of the Q table from the estimated Q values (Line 13,14). Lastly, the algorithm records the action with the lowest Q value. It is important to note that we do not set a static $p^c_{min}$ value. Instead, we incrementally reduce $p^c_{min}$ and loop through possible actions and evaluate any new contingencies whose $p^c$ is higher than the current $p^c_{min}$. Inherently, we are iteratively expanding the state transition diagram for each possible action and performing value backups at each iteration so that the best action will always be returned when output is needed.

## VII. Results

We evaluate our approach in simulation for 2 assemblies (1) a 24-task assembly generated to represent an assembly process with three subassembly groups and (2) a 60-task assembly that reflects the assembly of a mock CubeSat. Both assemblies are executed using a 1-human-5-robot team. We use the 24-task assembly to show how different aspects of the problem and algorithm parameters affect the online policy generation process. Our objective for experiments conducted with the 60 task assembly experiments is to use a real robot assembly cell to justify the decisions we make for the problem formulation and demonstrate our approach to a realistic assembly process.

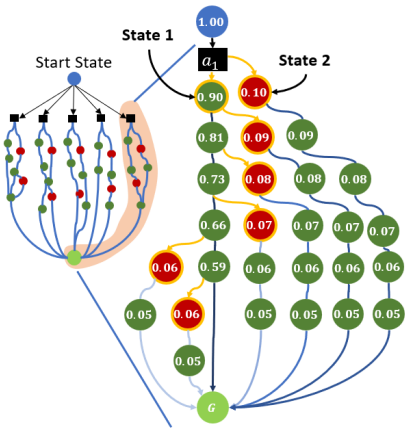


Fig. 3: A approximation of state trajectories for a single state action pair root node

In our implementation, we solved our MILP using Google OR-Tools CP-SAT solver [41]. For a 24-task and 60-task problem, the solver, on average, returned a solution in 30ms and 90ms, respectively. We create a lognormal distribution for both task problems to model the delay that may occur when the human executes a task. For the distribution, the mean delay is 25 percent of the total task duration, and the standard deviation is set so that there is a 95% probability that the human will finish the task by 1.5 times the nominal duration. For all tasks, we model that the robot is 80% faster in completing tasks than a human.

For the 24-task problem, we randomly selected half the tasks to fail prematurely when being executed by the robot. We create a skew-normal distribution such that the mean is 1/2 the nominal duration, and we arbitrarily set the standard deviation so that there is a realistic variance.

### A. Examining the compound probability $p^c$ parameter

For our approach, the $p^c_{min}$ parameter drives the sampling intensity; therefore, it is important to examine how this parameter affects the search depth and the efficacy of our methods. It is important to note that the smaller the $p^c$ value, the more sampling that is being done from the state transition diagram. We are also interested in how these results change for different levels of stochasticity in the problem domain. We, therefore, test against multiple stochastic variations of the 24 tasks assembly. Specifically, we vary the probability P(x) that each task will succeed, e.g. by setting P(x) = .7, we are setting all tasks to have a 70% probability of succeeding.

Figure 4a shows the relationship between the $p^c$ parameter and the number of contingencies sampled for different task success probabilities. The number of explored contingencies directly affects our method's computational speed, as shown in Table I. Though for an extremely small value of $p^c$ = .001 we find a difference in the number of contingencies explored for a task with P(x) = .9 vs P(x) = .6, in general, the number of contingencies explored by our approach increases exponentially at the same rate, regardless for how stochastic the problem setting is, as we decrease the $p^c$ parameter.

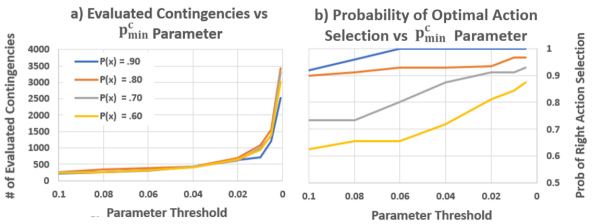


Fig. 4: The effect of setting the $p^c$ parameter on the number of evaluated contingencies and the probability P(x) the system returns the optimal action

| # Nodes | 2848 | 1233 | 915 | 748 | 397 | 270 |
|---|---|---|---|---|---|---|
| Planning (s) | 116.8 | 40.3 | 29.7 | 24.7 | 14.0 | 9.7 |

TABLE I: Number of nodes evaluated vs planning time required before best action was returned (P(x) = .9)

Figure 4b shows how the $p^c$ parameter affects the probability that our approach returned the optimal action. We see that for task problems with low stochasticity, we have to evaluate fewer contingencies (by setting a larger $p^c_{min}$ parameter), and the algorithm will still always return the optimal solution. We see that our method needs to sample a much larger number of contingencies when the task success probability is significantly lower. We use this result to set the lower bound for our examination of task success probability to be P(x) = .7 and above.

### B. 24 Task Simulation Results

We test our approach compared to other planning approaches on different variations of task success probabilities for the 24-task problem. Specifically, we test against three approaches: classical MILP scheduling, real-time dynamic programming, and value iteration. For the classical scheduling approach, we use our MILP formulation from Section V to generate an optimal task assignment and simulate the solution. If an unforeseen event occurs, we rerun the MILP solver and execute the new solution. We also implement a real-time dynamic programming approach (RTDP) as proposed in [33]. The authors developed a heuristic probabilistic planning approach to solve single-agent CoMDPs with stochastic durations. Lastly, we use a value iteration approach to get the exact optimal solution for the 24-task problem. We first used random action-taking to generate the reachable state space. Then we ran value iteration for a substantial amount of time until it converged to an optimal policy.

Our generated results are shown in Figure 5. We find that our approach can generate solutions that lead to the same makespan as value iteration. This result indicates that for the 24-task problem, our approach does yield the optimal policy. Our approach also outperforms classical scheduling and RTDP, especially in situations with higher uncertainty in the task. It is also interesting to note that our approach has a lower variance in resulting makespans compared to RTDP and MILP, which is another strength of the approach.

### C. Implementation on a robotic cell

We also implemented our approach on a 1-human-5-robot cell that is designed to assemble a CubeSat. The entire assembly requires 150 operations, which we simplify into a 60-task HTN. We aim to convey the implementation of our online action selection formulation on this system and present results that motivated our initial decisions when we formulated the problem.

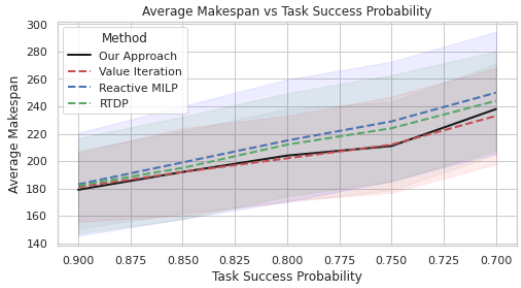


Fig. 5: Graph shows average makespan for our approach, RTDP, optimistic MILP, and value iteration

We first examined how certain tasks failed so that we could create a task failure model. Based on qualitative discussions with the assembly team and experimentation, we showcase two types of tasks: heavy part insertion and screw driving task, shown in Figure 1 that had a non-negligible probability of failing, requiring the human to intervene. Further experimentation showed an average of 71% success rate for the part insertion and 83% success rate for the screwing tasks.

We also found from experimentation that screwing tasks either failed early in the task duration or succeeded by the nominal duration. We conducted screwing for a specific part of the CubeSat and found that it reliably failed within 30% of the task time; else, the task finished successfully by the nominal duration for the rest of the time. We found that slight misalignment for two parts can cause the system to fail, and the human operator had to remove the screw, realign the part, and fasten the screw. This result motivates our decision to formulate the general problem as having different duration distributions for whether the task succeeds or fails.

We ran our approach on the system in simulation, and we report a makespan of 683 seconds and a standard deviation of 52 seconds. An interesting observation is that the algorithm did not assign the human to other concurrently available tasks during the screw-driving tasks showing our system took advantage of the stochastic duration model. Lastly, we found that the human was always assigned to the battery insertion task due to the lower probability of success.

## VIII. Conclusions

This paper presents a task assignment and scheduling framework for a human-robot team to complete complex tasks efficiently. Our approach illustrates the value of the following two ideas: (1) we can capture and account for the complex interaction of uncertainty that stems from the task and agents using a multi-agent concurrent MDP framework, and (2) we can use Mixed Integer Linear Programs and contingency sampling to approximate action values in a computationally efficient manner. Simulation and real-world implementations demonstrate that our approach can successfully return task assignments that minimize the task completion makespan.

## Acknowledgements

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

## REFERENCES


[1] 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. American Society of Mechanical Engineers, 2023.

[2] 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.

[3] S. Shriyam and S. K. Gupta, "Modeling and analysis of subsystem interactions in robotic assembly," in International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 59179. American Society of Mechanical Engineers, 2019, p. V001T02A066.

[4] 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 2022 31st IEEE International Conference on Robot & Human Interactive Communication (RO-MAN). IEEE, 2022.

[5] S. Shriyam and S. K. Gupta, "Incorporating potential contingency tasks in multi-robot mission planning," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 3709–3715.

[6] ——, "Task assignment and scheduling for mobile robot teams," in International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 51807. American Society of Mechanical Engineers, 2018.

[7] ——, "Incorporation of contingency tasks in task allocation for multirobot teams," IEEE Transactions on Automation Science and Engineering, vol. 17, no. 2, pp. 809–822, 2019.

[8] ——, "Modeling and verification of contingency resolution strategies for multi-robot missions using temporal logic," International Journal of Advanced Robotic Systems, vol. 16, no. 6, p. 1729881419885697, 2019.

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

[10] 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.

[11] M. Gombolay, R. Jensen, J. Stigile, T. Golen, N. Shah, S.-H. Son, and J. Shah, "Human-machine collaborative optimization via apprenticeship scheduling," Journal of Artificial Intelligence Research, vol. 63, pp. 1–49, 2018.

[12] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, and B. Williams, "Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty," in Proceedings of the International Conference on Automated Planning and Scheduling, vol. 26, 2016, pp. 267–275.

[13] G. Vázquez, R. Calinescu, and J. Cámara, "Scheduling multi-robot missions with joint tasks and heterogeneous robot teams," in Annual Conference Towards Autonomous Robotic Systems. Springer, 2021, pp. 354–359.

[14] M. Pearce, B. Mutlu, J. Shah, and R. Radwin, "Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes," IEEE Transactions on Automation Science and Engineering, vol. 15, no. 4, pp. 1772–1784, Oct 2018.

[15] G. Michalos, J. Spiliotopoulos, S. Makris, and G. Chryssolouris, "A method for planning human robot shared tasks," CIRP journal of manufacturing science and technology, vol. 22, pp. 76–90, 2018.

[16] A. Roncone, O. Mangin, and B. Scassellati, "Transparent role assignment and task allocation in human robot collaboration," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 1014–1021.

[17] M. Cirillo, L. Karlsson, and A. Saffiotti, "Human-aware task planning: an application to mobile robots," ACM Transactions on Intelligent Systems and Technology (TIST), vol. 1, no. 2, pp. 1–26, 2010.

[18] W. Wang, R. Li, Z. M. Diekel, and Y. Jia, "Robot action planning by online optimization in human–robot collaborative tasks," International Journal of Intelligent Robotics and Applications, vol. 2, no. 2, pp. 161–179, 2018.

[19] H. Nemlekar, J. Modi, S. K. Gupta, and S. Nikolaidis, "Two-stage clustering of human preferences for action prediction in assembly tasks," in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 3487–3494.

[20] H. Zhang, S.-H. Chan, J. Zhong, J. Li, S. Koenig, and S. Nikolaidis, "A mip-based approach for multi-robot geometric task-and-motion planning," in IEEE International Conference on Automation Science and Engineering (CASE), 2022.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] S. Hong, S. U. Lee, X. Huang, M. Khonji, R. Alyassi, and B. C. Williams, "An anytime algorithm for chance constrained stochastic shortest path problems and its application to aircraft routing," in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 475–481.

[29] S. Sen, "Algorithms for stochastic mixed-integer programming models," Handbooks in operations research and management science, vol. 12, pp. 515–558, 2005.

[30] A. Kolobov, "Planning with markov decision processes: An ai perspective," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 6, no. 1, pp. 1–210, 2012.

[31] P. Schillinger, M. Burger, and D. V. Dimarogonas, "Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning," The international journal of robotics research, 2019.

[32] D. S. Weld, "Solving concurrent markov decision processes," in Proceedings of the 19th national conference on Artifical intelligence, 2004, pp. 716–722.

[33] D. S. Weld et al., "Planning with durative actions in stochastic domains," Journal of Artificial Intelligence Research, vol. 31, pp. 33–82, 2008.

[34] D. S. Weld, "Concurrent probabilistic temporal planning," in Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling, 2005, pp. 120–129.

[35] M. Strens and N. Windelinckx, "Combining planning with reinforcement learning for multi-robot task allocation," in Adaptive Agents and Multi-Agent Systems II. Springer, 2004, pp. 260–274.

[36] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 2, pp. 156–172, 2008.

[37] 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.

[38] 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.

[39] K. Li, Q. Liu, W. Xu, J. Liu, Z. Zhou, and H. Feng, "Sequence planning considering human fatigue for human-robot collaboration in disassembly," Procedia CIRP, vol. 83, pp. 95–104, 2019.

[40] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," Computers & Operations Research, vol. 73, pp. 165–173, 2016.

[41] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/