Autonomous Robots
https://doi.org/10.1007/s10514-021-10022-9

# Dynamic multi-robot task allocation under uncertainty and temporal constraints

Shushman Choudhury[1] · Jayesh K. Gupta[1] · Mykel J. Kochenderfer[1] · Dorsa Sadigh[1] · Jeannette Bohg[1]

Received: 14 January 2021 / Accepted: 27 September 2021

**Abstract**
We consider the problem of dynamically allocating tasks to multiple agents under time window constraints and task completion uncertainty. Our objective is to minimize the number of unsuccessful tasks at the end of the operation horizon. We present a multi-robot allocation algorithm that decouples the key computational challenges of sequential decision-making under uncertainty and multi-agent coordination, and addresses them in a hierarchical manner. The lower layer computes policies for individual agents using dynamic programming with tree search, and the upper layer resolves conflicts in individual plans to obtain a valid multi-agent allocation. Our algorithm, *Stochastic Conflict-Based Allocation* (SCoBA), is optimal in expectation and complete under some reasonable assumptions. In practice, SCoBA is computationally efficient enough to interleave planning and execution online. On the metric of successful task completion, SCoBA consistently outperforms a number of baseline methods and shows strong competitive performance against an oracle with complete lookahead. It also scales well with the number of tasks and agents. We validate our results over a wide range of simulations on two distinct domains: multi-arm conveyor belt pick-and-place and multi-drone delivery dispatch in a city.

**Keywords** Task allocation · Multi-robot systems · Hierarchical planning · Conflict-based search

## 1 Introduction

Efficient and high-quality task allocation is crucial for modern cooperative multi-robot applications (Gerkey and Mataric 2004). For warehouse logistics, teams of mobile robots carry goods between assigned locations (Yan et al. 2012). Industrial and manufacturing operations involve manipulators collaborating on assembly lines (Johannsmeier and Haddadin 2016). On-demand ridesharing and delivery services dispatch agents to incoming requests (Hyland and Mahmassani 2018). Multi-robot task allocation needs to be computationally efficient and produce high-quality solutions under the challenges of real-world robotics: the uncertainty of executing tasks successfully, temporal constraints such as ordering and time windows, and tasks dynamically appearing online. For instance, in one of our simulation domains, a team of robot arms pick objects that appear on a conveyor belt from an external loading process and place them in bins (Fig. 1a). With time window constraints induced by workspace limits and uncertainty due to imperfect grasping, the arms attempt to pick-and-place as many objects as possible.

Multi-robot task allocation is a difficult problem; it inherits the combinatorial optimization challenges of classical allocation as well as the uncertainty and dynamic online environments of robotics settings. Time-extended tasks and time window constraints further require algorithms to plan over horizons rather than instantaneously, and account for spatio-temporal relationships among tasks (Gini 2017). The robotics community has worked on multi-agent task allocation with Markov Decision Processes (Campbell et al. 2013) and robust task matching (Liu and Shell 2011). The classic multi-robot task allocation problem has been studied extensively (Gerkey and Mataric 2004) and extended to account for uncertainty (Mataric et al. 2003), temporal and ordering constraints (Gini 2017), and dynamic task arrivals (Cordeau and Laporte 2007). The operations research community has developed methods that plan under task execution uncertainty (Timotheou 2010; Rahmani and Heydari 2014) and can efficiently recompute schedules online (O'Donovan et

This is one of the several papers published in Autonomous Robots comprising the Special Issue on Robotics: Science and Systems 2020.

✉ Shushman Choudhury
shushman@cs.stanford.edu

[1] Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Published online: 06 November 2021

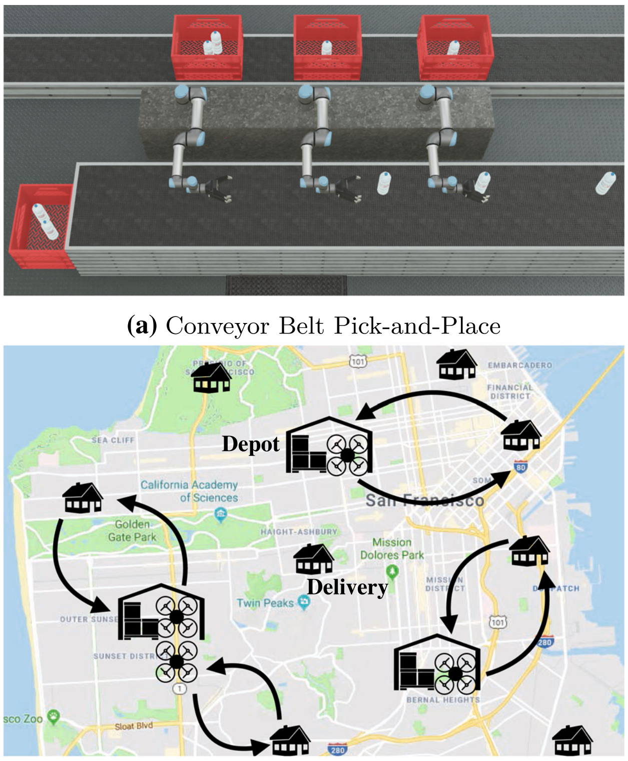

(a) Conveyor Belt Pick-and-Place
Depot
Delivery
(b) Multi-Drone Delivery Dispatch

**Fig. 1** The above domains motivate our multi-robot task allocation approach. We allocate robots (arms or drones) to tasks (pick-and-place or delivery) that arrive online. Task completion is subject to uncertainty (grasping or flight time) and time window constraints

al. 1999). However, they have simplified agent models (flow shops, job shops) that are unable to represent complex spatial relationships between tasks.

The algorithmic challenges for our allocation setting are *sequential planning under uncertainty* and *coordinating multi-agent decisions*. The prior works above typically attempt the computationally intractable joint problem. Thus, they require simplifying approximations or heuristics for either multi-agent planning or coordination, such as ignoring uncertainty or imposing arbitrary priority orderings on agents.

Our key idea is to *decouple the algorithmic challenges and address them hierarchically* in an efficient two-layer approach. The lower layer plans for individual agents, using dynamic programming on a policy tree to reason about the uncertainty over task completion. The upper layer uses optimal conflict resolution logic from the path planning community to coordinate the multi-agent allocation decisions (Sharon et al. 2012). Our overall algorithm, *Stochastic Conflict-Based Allocation* (SCoBA), yields allocation policies that minimize the expected cumulative penalty for unsuccessful tasks. Due to its computational efficiency and tree structure in both layers, SCoBA can also seamlessly interleave planning and execution online for new tasks.

The following are the contributions of our work:

1. We propose a general formulation for multi-robot allocation under task uncertainty and temporal constraints.
2. We present a hierarchical algorithm, SCoBA, which uses multi-agent conflict resolution with single-agent policy tree search. We prove that SCoBA is both optimal in expectation and complete under mild assumptions.
3. We demonstrate SCoBA's strong competitive performance against an oracle with complete lookahead, and its advantage over four baseline methods for successfuly executing tasks. Our results also show that SCoBA scales well with increasing numbers of agents and tasks. We run simulations on two distinct robotics domains (Fig. 1); a team of robot arms picking and placing objects from a conveyor belt and on-demand delivery of packages by a team of drones in a city-scale area.

This paper builds upon an earlier version that appeared in Robotics: Science and Systems 2020 (Choudhury et al. 2020a). It adds detailed proofs for both theoretical properties and elaborates on the other algorithmic features. It also augments the experiments section with an expanded description of how we generate tasks in our first simulation domain, an additional sets of results for each of the two domains with a different task execution probability model, and more detailed comparisons on computation time.

## 2 Background and related work

We briefly discuss three background areas: algorithms for assignment and scheduling, multi-agent decision-making, and relevant work in multi-robot task allocation.

### 2.1 Assignment and scheduling

A major topic in discrete optimization is assigning resources to tasks (Burkard et al. 2009), for which the Hungarian algorithm is a fundamental one (Munkres 1957). In temporal tasks, we use the *scheduling* model, where the objective is a function of completed jobs (Pinedo 2012). Scheduling problems with multiple resources and tasks are computationally hard, even when deterministic (Lenstra et al. 1977). In online scheduling, each task is only observed when made available (Albers 1999). Hard real-time tasks have a time window constraint for completion (Dertouzos and Mok 1989).

Approaches for scheduling under uncertainty address problems where task execution is not fully deterministic (Chaari et al. 2014). These approaches are either proactive in anticipating future disruptions (Lin et al. 2004), reactive to changes (Szelke and Kerr 1994; Raman et al. 2015), or hybrid (Church and Uzsoy 1992). For scenarios with ordering constraints, such as assembly lines, additional models like job shops (Al-Hinai and ElMekkawy 2011) and flow shops (González-Neira et al. 2017) are useful, particularly real-time flow shop scheduling (Rahmani and Heydari 2014; Framinan et al. 2019). This extensive body of work provides valuable insights but does not address task configurations more complex than an ordered sequence on an assembly line (such as delivery requests in a geographical area) and the uncertainty of travel time to reach these tasks.

## 2.2 Multi-agent sequential decision-making

The Markov Decision Process (MDP) is a mathematical model for our setting of sequential decision making under uncertainty (Kochenderfer 2015). Different solution techniques exist for MDPs, depending on available information; dynamic programming (Bertsekas 2005) when the explicit transition model is known, sample-based online planning (Péret and Garcia 2013) when only a generative model exists, and reinforcement learning (Sutton and Barto 2018) when no model is available. Our problem is a multi-agent MDP (MMDP), where agents coordinate to achieve a single shared objective; planning for MMDPs is generally computationally intractable in practice due to the exponentially large decision space (Boutilier 1996).

Reinforcement learning techniques are often employed to alleviate tractability issues (Littman 1994) by learning values of different states and actions, but model-free methods like Q-Learning face exploration challenges (Lanctot et al. 2017). Recently, graph neural network representation techniques have been used to learn good heuristics and reduce the overall combinatorial complexity of such multi-robot allocation policies (Wang and Gombolay 2020; Zhang et al. 2020). Online tree search methods can fare better than these offline approaches by focusing on relevant states that are reachable from the current one (Vodopivec et al. 2017), and a recent multi-robot allocation algorithm is based on online Monte Carlo Tree Search (Kartal et al. 2016). None of these methods have any guarantee on solution quality or completeness, due to the computational intractability of Multi-Agent MDPs.

## 2.3 Multi-robot task allocation

We outline a number of domain-agnostic and domain-specific works on multi-robot task allocation. MDP solvers have been used to generate a sequential greedy strategy, but without accounting for completion uncertainty (Campbell et al. 2013). The probability of task failure has been considered by two-stage Stochastic Integer Programs and network flow algorithms, which are exhaustive combinatorial approaches unsuitable for tasks that are streaming in online (Ahmed and Garcia 2003; Timotheou 2010, 2011). A sensitivity analysis approach to optimal assignment under uncertainty provides some insights on robustness but has no notion of temporal constraints (Liu and Shell 2011).

Existing taxonomies for multi-robot task allocation help us understand our problem difficulty (Gerkey and Mataric 2004; Nunes et al. 2017). Among the early works reviewed in these taxonomies, two fundamental ones study the effects of uncertain environments and dynamic tasks (Mataric et al. 2003; Lerman et al. 2006). However, they both consider distributed multi-robot systems, with simple allocation strategies based on local information that lack any global guarantees. More recent work has developed an auction-based method for distributed mobile robot teams that plans time-constrained pickup and delivery schedules online (Coltin and Veloso 2014). On the centralized side, another recent paper presents a comprehensive approach for task assignment and scheduling that handles tightly coupled spatial and temporal constraints (Gombolay et al. 2018). However, neither of them explicitly models or addresses task execution uncertainty and how it interacts with time constraints to yield task failure probabilities.

Previous work on assembly lines includes hierarchical planning frameworks, constraint programming, and robust scheduling for robotic flowshop systems (Johannsmeier and Haddadin 2016; Behrens et al. 2019; Che et al. 2017). However, they all simplify one or more of the key complexities such as task completion uncertainty or multi-agent configuration models. Dynamic vehicle dispatch problems have been explored in work on vehicle routing algorithms with time windows and trip assignment algorithms for ridesharing (Lau et al. 2003a; Alonso-Mora et al. 2017). However, they make restrictive assumptions on the uncertainty and environment dynamics. Driver-task assignment with uncertain durations and task windows do solve for a similar setting as ours but assume some knowledge of future requests (Cheung et al. 2005).

# 3 Problem formulation

We base our formulation on previous work for multi-robot task allocation with temporal constraints (Gini 2017). There is a set of $N$ **agents**, denoted as $[N]$ and $K$ **tasks**, denoted as $[K]$; the problem horizon is $T$ time-steps. For each agent $n \in [N]$ and task $k \in [K]$, the service **time window** is $W_{nk} = \left(t^l_{nk}, t^u_{nk}\right)$, where $l$ and $u$ are respectively the lower and upper time limits within which $n$ can attempt $k$. There may also be an additional so-called **downtime** if the agent executes the task successfully, e.g., the time for a robot arm to transfer the object to the bin. We represent **task duration**

**uncertainty** as

$$\tau_{nk}(t) = \text{Prob}\left[n \text{ completes } k \text{ within time } t\right]. \tag{1}$$

We assume knowledge of this cumulative distribution as part of the problem specification, typical for task scheduling under uncertainty (Chaari et al. 2014); the particular model is domain-dependent. By definition, the conjunction of $W$ and $\tau$ imposes an upper bound on **task completion probability**, i.e.,

$$\text{Prob}\left[n \text{ completes } k\right] \leq \tau_{nk}\left(t^u_{nk} - t^l_{nk}\right). \tag{2}$$

For all unsuccessful tasks, the system incurs a **penalty** of $\sum_k J(k)$ units. An agent can attempt only one task at a time.

We seek an allocation policy that *minimizes the expected cumulative penalty due to unsuccessful tasks*. An allocation policy $\pi$ is a mapping from the agents to the tasks and their respective attempt times, i.e. $\pi : [N] \to [K] \times [T]$. Since there is uncertainty about task completion, a single-shot allocation is insufficient. Of course, the attempt times for future tasks depend on when the earlier tasks are actually executed (successfully or unsuccessfully). Our optimization problem is

$$\begin{aligned} &\underset{\pi \in \Pi}{\text{argmin}} \quad \mathbb{E}\left[\sum_{k \in [K]} \mathbb{1}[k] \cdot J(k) \mid \pi\right] \\ &\text{s.t.} \qquad t \in W_{nk} \;\; \forall\, (k, t) \in \pi(n), \end{aligned} \tag{3}$$

where the indicator function $\mathbb{1}[k] = 1$ if the task $k$ remains incomplete at the end of the horizon, and $\Pi$ is the set of all possible allocation policies. The constraint enforces that an agent attempts a task within the valid time window. The expectation is over the task execution success distribution for the allocation policy. For the rest of the discussion, we will assume that $J(k) = 1$, i.e., all tasks are equally important; this objective is the *unweighted tardy jobs penalty* (Pinedo 2012).

Multi-robot task allocation algorithms have been developed for various different objective functions (Gini 2017). Our choice to minimize the number of missed tasks is a standard one for allocation problems with time window constraints (Lau et al. 2003b). This objective is entirely independent of the the underlying task, which makes it more domain-agnostic than some other ones. Its additive nature is also useful for how we design and prove the algorithmic properties of our proposed approach.

The discrete-time rolling horizon formulation described above is fairly expressive and useful. We are concerned with high-level allocation rather than the underlying task execution, so we avoid the added complexity of continuous-time representations. The underlying tasks typically involve time-constrained trajectory planning, for which there are well-established models and methods (Laumond 1998). Furthermore, we can interleave planning and execution suitably and recompute an allocation policy when new tasks appear online (and we do so in practice).

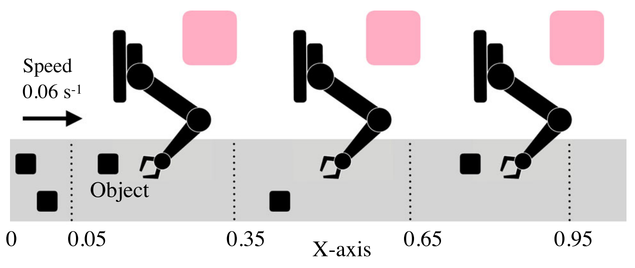


**Fig. 2** The illustrated conveyor belt has $N = 3$ arms and $K = 5$ objects. The belt is of unit length, and each arm's workspace spans 0.3 units (dashed lines are the limits). Given the belt speed, the agent-task time window for any arm-object pair is at most 5 s

## 3.1 Motivating examples

We describe two distinct robotics settings to instantiate our formulation. First, consider the previously introduced example of robot arms along a conveyor belt (see Fig. 2). Each arm has an associated collection bin for objects picked up from the belt. The objects appear on the belt through an external process. The arms take varying amounts of time for picking, depending on the quality of the grasp strategy or gripper attributes. Arms have finite reach, and an object may not be picked up before it goes out of reach. Objects missed by all arms must be sorted by hand afterwards. The goal is to successfully pick-and-place as many objects, or equivalently, miss as few objects as possible.

Second, consider on-demand multi-drone dispatch for package delivery in a city (note the underlying similarities to the previous example). Delivery tasks arise through an external process of customer requests. Drones take varying amounts of time to travel from the product depot to the package delivery location, depending on flight conditions. Requests arrive with time windows, such that drones must wait until the window starts to deliver the product to the customer, and late deliveries are penalized. Over a certain time horizon, our objective is to minimize the number of late deliveries.

## 3.2 Computational challenges

To motivate our approach, we briefly discuss the problem complexity. By the multi-robot task allocation taxonomy of Gerkey and Mataric (2004), the deterministic version of our problem is ST-SR-TA, i.e. a single robot (SR) executes a single task (ST) at a time, where tasks are time-extended (TA) rather than instantaneous. The ST-SR-TA is an instance of an NP-Hard scheduling problem, specifically multi-agent scheduling with resource constraints (Garey and Johnson 1975). The uncertainty of task execution success due to time windows exacerbates this difficulty (Gini 2017). Finally, new tasks streaming in require our approach to interleave planning and execution effectively, e.g., by replanning at task arrivals (Cordeau and Laporte 2007).

## 4 Hierarchical multi-robot task allocation

Our key algorithmic challenges are *sequential planning under uncertainty* of task completion and *multi-agent coordination* of allocations. The joint multi-agent planning problem is computationally prohibitive for large settings (Boutilier 1996); most closely related previous works either use simplifying approximations for planning and optimization (Timotheou 2011; Hyland and Mahmassani 2018) or simple coordination heuristics (Kartal et al. 2016; Mataric et al. 2003).

In contrast, *we address the challenges hierarchically in a two-layer approach* called Stochastic Conflict-Based Allocation (SCoBA). At the low level, we independently determine the optimal task attempt sequence for each individual agent, ignoring other agents. At the high level, we resolve potential conflicts in assigned tasks across multiple agents to obtain a valid multi-robot allocation. In this section, we will elaborate on the two layers and how they come together in SCoBA. We will then discuss briefly how we interleave planning and execution online and how SCoBA can exploit sparse agent interactions using coordination graphs.

### 4.1 Low-level: single agent policy

We first consider the perspective of an individual agent, independent of the other ones. From the definition in Sect. 3, we are given the set of current tasks, corresponding time windows, and task completion uncertainty distribution, and we want a task attempt policy tree for the agent. Since task execution is stochastic, the first time an agent can attempt a task depends on what the agent has attempted before it. We make a simplifying approximation to compute the policy tree: *the agent attempts a task as soon as possible and observes the outcome at the end of the window*. This approximation collapses the temporal dimension by treating tasks as discrete events rather than extended ones.

We illustrate the policy tree search process for a single robot $n_1$ and three tasks (objects) $k_1, k_2, k_3$ in Fig. 3. First, we sort tasks in increasing order of the start of their time window. Then, we sweep along the time axis and update the tree at every *event point*, i.e., the start or finish of the window (and the end of the downtime in case the task is successful). The updates to the policy tree depends on the event point (start/finish/downtime).

For the start of a time window, we introduce two new *decision nodes* (ovals) to attempt ($\leftrightarrow$) or leave ($\not\leftrightarrow$) the task respectively. At the end of a time window and the downtime, we introduce *outcome nodes* (rectangles) respectively for failure or success, where the outcome probability $p$ depends on the minimum feasible start time for the attempt, which in turn depends on the specific branch of the tree. For instance, notice in Fig. 3 the three copies of the decision node ($n_1 \leftrightarrow k_2$), with different probabilities, depending on whether it was attempted after the failure, success, or non-attempt of task $k_1$. This difference is due to the time left to complete the task, e.g., a non-attempt of $k_1$ leaves the most time and highest probability to complete $k_2$.

The leaves of the binary policy tree are annotated with the cumulative penalty along their branches, e.g., a penalty of 1 for each unsuccessful task. We then use dynamic programming to propagate values upwards from the leaves to the root. For a pair of outcome node siblings, we set the parent's value (denoted as $V$) to the expected value of its children,

$$V(\text{parent}) \;:=\; p \cdot V(\text{Fail}) + (1-p) \cdot V(\text{Succ}). \tag{4}$$

For a pair of decision node siblings, the parent's value is the minimum of the children's, i.e.,

$$V(\text{parent}) \;:= \min\{V(\text{child1}),\; V(\text{child2})\}. \tag{5}$$

In the running example of Fig. 3, we have $V(\text{root}) = \min\{V(n_1 \leftrightarrow k_1), V(n_1 \not\leftrightarrow k_1)\}$. The resulting tree encodes the policy that minimizes the agent's expected penalty for all tasks up to the planning horizon, and $V(\text{root})$ is the value of this expected penalty. We obtain the next task assigned to the agent by following child nodes of minimum value until the first attempt node (e.g., $n_1 \leftrightarrow k_1$).

### 4.2 High-level: multi-agent coordination

The policy tree determines the approximately optimal task attempt sequence for an individual agent (approximate due to the temporal simplification mentioned earlier). The tree searches are independent of each other, so two agents may have conflicting allocations. Since our objective function depends on all agents, *breaking ties naïvely could yield arbitrarily poor global allocations*. Multi-agent pathfinding algorithms face a similar challenge and have to resolve inter-agent conflicts between shortest paths (Felner et al. 2017). Conflict-Based Search is an effective strategy for this problem (Sharon et al. 2012); by decoupling single-agent path planning and inter-path conflict resolution, it is efficient in practice without losing optimality. It has even been

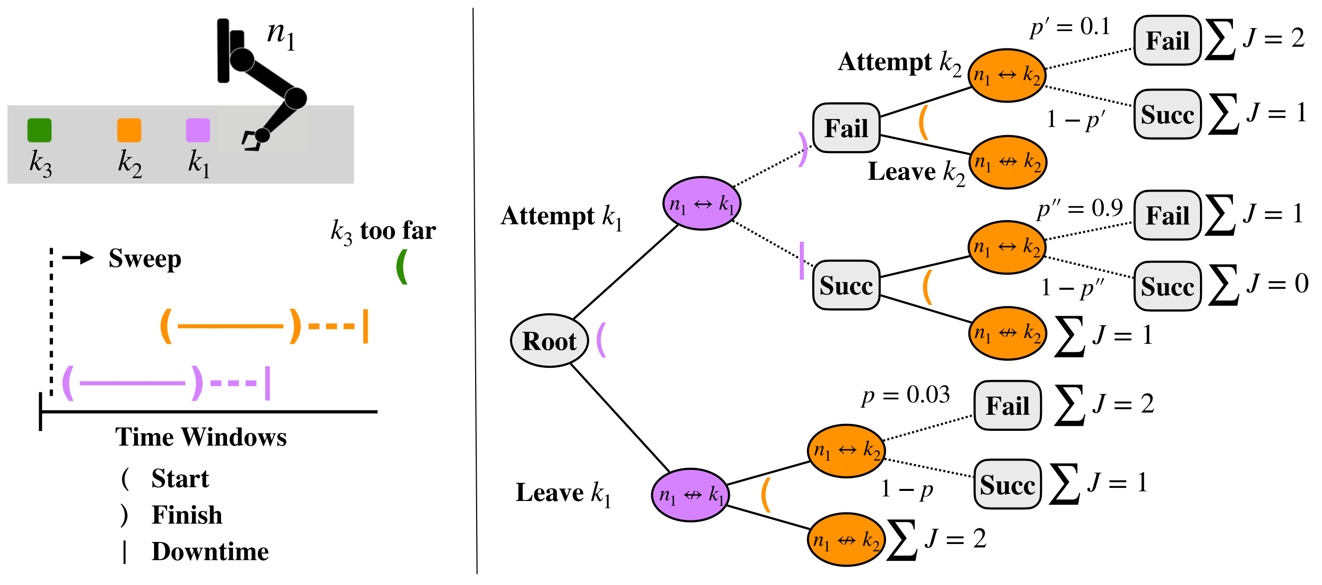


**Fig. 3** The low-level routine of SCoBA generates the policy tree over valid tasks for an individual agent, specifically, by sweeping along the time axis and branching on the start or finish of a task's time window. At the start of a window, two new *decision nodes* (ovals) are introduced: *to attempt* ($\leftrightarrow$) or *to leave* ($\not\leftrightarrow$) the task respectively. At the end of a time window and the downtime, the *outcome nodes* (rectangles) depict failure or success. After the tree generation, dynamic programming propagates the values from the leaves to the root. The probability values $p = 0.03$, $p' = 0.1$, $p'' = 0.9$ are just hypothetical values that illustrate how the same attempt node ($n_1 \leftrightarrow k_2$) has three different copies, with different outcome probabilities (depending on the branch of the tree). Since we interleave planning and execution, we can ignore task $k_3$ as its time window begins after the end of every task time window before it

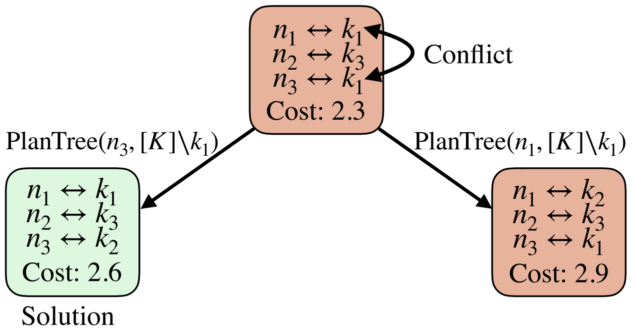
used to solve a joint task assignment and path finding problem (Hönig et al. 2018).

We leverage the idea of inter-agent conflict resolution from Conflict-Based Search. The high level of our algorithm, SCoBA, searches a binary *constraint tree* (Fig. 4) generated from conflicts between solutions for individual agents obtained from the low level, i.e., the policy tree search. Two agents $n_1$ and $n_2$ are in *conflict* if they are allocated the same task $k$ in overlapping time windows, i.e., if $(k, t_1) \in \pi(n_1), (k, t_2) \in \pi(n_2)$ and either $t_2 \in W_{n_1,k}$ or $t_1 \in W_{n_2,k}$. A *constraint* for an agent is a task excluded from consideration by the tree search for that agent.

Each node in the constraint tree maintains (i) a set of constraints, i.e., tasks to ignore, for each agent, (ii) a multi-agent allocation that respects all constraints, and (iii) the cost of the allocation. For SCoBA, the cost of the allocation is the sum of expected penalties for each agent, where the expected penalty for each agent is the value of the root node of its policy tree. The allocation cost is used as the criteria for *best-first search on the constraint tree*; this best first search continues until it finds a conflict-free allocation.

## 4.3 Stochastic conflict-based allocation (SCoBA)

Algorithm 1 describes SCoBA, using the tree search of Sect. 4.1 as the PLANTREE subroutine. Its structure is similar to an earlier presentation of Conflict-Based Search by Felner et al.



$n_1 \leftrightarrow k_1$
$n_2 \leftrightarrow k_3$
$n_3 \leftrightarrow k_1$
Cost: 2.3
Conflict
PlanTree($n_3$, $[K]\backslash k_1$)
PlanTree($n_1$, $[K]\backslash k_1$)
$n_1 \leftrightarrow k_1$
$n_2 \leftrightarrow k_3$
$n_3 \leftrightarrow k_2$
Cost: 2.6
Solution
$n_1 \leftrightarrow k_2$
$n_2 \leftrightarrow k_3$
$n_3 \leftrightarrow k_1$
Cost: 2.9

**Fig. 4** A constraint tree node with a conflict in the allocation generates two children with corresponding constraints on the conflicting agents ($n_1$ and $n_3$) and task ($k_1$). Best-first search on the constraint tree returns the first high-level node with a conflict-free allocation. The solution node of the constraint tree is coloured green and the others are coloured red

(2017). The constraint tree is initialized with the root node, which has an empty constraint set and the allocation from running PLANTREE for each individual agent (lines 2–6). When a high-level node is expanded, the corresponding allocation is checked for validity (line 9). If there is no conflict, we return this allocation as the solution. Otherwise, for every conflict between two or more agents, new child nodes are added, where constraints are imposed on the agents involved (line 15). A child constraint tree node inherits its parent's constraints and adds one more constraint for a particular agent.

Consider the simple illustrative example in Fig. 4. The root node has agents $n_1$ and $n_3$ both assigned to task $k_1$. This conflict yields two possible constraints, one inherited by each of the two child nodes. The first constraint excludes $k_1$ from the recomputed policy tree search for $n_1$. The second constraint does the same for $n_3$. For each new (non-root) node, the low level tree search is only re-run on the agent for which the constraint is added (line 19). Both of the resulting child nodes are conflict-free, but the left one, with a lower allocation cost of 2.6, is returned as the solution.

Our problem setting is both online and stochastic. However, under some simplifying assumptions, we can establish optimality and completeness properties for SCoBA. We derive them from the corresponding optimality and completeness proofs of the Conflict-Based Search algorithm for multi-agent pathfinding (Sharon et al. 2012).

**Algorithm 1** Stochastic Conflict-Based Allocation

```
 1: procedure MAIN($[N], [K], T, W_{n,k}, \tau_{nk}\ \forall\ n, k$)
 2:     Initialize A as the root
 3:     $A.soln \leftarrow$ PLANTREE($n, [K], T, W, \tau$) $\forall\ n$
 4:     $A.constr \leftarrow \{\}$                          ▷ Empty constraint set
 5:     $A.cost \leftarrow$ SumOfIndividualCosts($A.solution$)
 6:     Insert $A$ into OPEN      ▷ Open list of Constraint Tree
 7:     while OPEN not empty
 8:         $S \leftarrow$ PopBest(OPEN)          ▷ Min. Cost Allocation
 9:         if $S.soln$ is valid                         ▷ No conflicts
10:             return $S.soln$
11:         $C \leftarrow$ find-conflicts($S$)          ▷ Inter-agent conflicts
12:         for all conflicts $(n, k) \in C$
13:             $A \leftarrow$ GENERATENEWCHILDPLANTREE($S, n, k$)
14:             $A.cost \leftarrow$ SumOfIndividualCosts($A.soln$)
15:             Insert $A$ into OPEN

16: procedure GENERATENEWCHILDPLANTREE($S, n, k$)
17:     $A.constr \leftarrow S.constr \cup k$                  ▷ Task to exclude
18:     $A.soln \leftarrow S.soln$
19:     $A.soln \leftarrow$ PLANTREE($n, [K] \setminus A.constr, T, W, \tau$)
20:     return $A$
```

**Proposition 1** *If (i) no new tasks are added online, (ii) the tree search is executed to the full horizon, and (iii) task completion is determined at the end of the time window, then SCoBA is optimal in expectation, i.e., SCoBA minimizes the expected number of incomplete tasks at the end of the time horizon.*

***Proof*** Conflict-Based Search yields optimal multi-agent solutions (paths) if two conditions hold: (a) the low-level routine yields optimal solutions for individual agents and (b) the overall multi-agent objective is the sum-of-costs of individual agent solutions. SCoBA uses the same multi-agent conflict resolution logic as Conflict-Based Search and will inherit its optimality property if it satisfies the two sufficient conditions.

We first show that (a) is true for SCoBA. Its low-level single-agent routine uses dynamic programming with forward tree search to obtain a policy tree. By construction, this routine computes a policy that is optimal under expectation for any discrete, finite-horizon Markov Decision Process (MDP) *if it conducts the tree search exhaustively, i.e., to the full horizon*. The expectation is over the uncertainty of the action outcomes. Assumption (i) ensures we know all information of future tasks at the initial state of the agent and Assumption (ii) ensures the tree search is exhaustive.

Recall that in Sect. 4.1 we treated time-windows as discrete events, a simplifying approximation. Under Assumption (iii), this temporal approximation now becomes exact; if we begin each task at the earliest possible time-step and continue until the end of the window, each task attempt has a single probability mass function over the two outcomes of success and failure. Therefore, the single agent problem is a discrete finite-horizon MDP and the low-level policy tree search routine is optimal in expectation. That is, for each agent $n$, the policy $\pi^*(n)$ obtained from PLANTREE satisfies

$$\pi^*(n) = \underset{\pi(n)\in\Pi(n)}{\operatorname{argmin}}\ \mathbb{E}\left[\sum_{k\in\pi(n)} \mathbb{1}[k] \cdot J(k) \mid \pi(n)\right], \tag{6}$$

subject to the constraints in Eq. 3, where $\Pi(n)$ is the set of all possible policy trees for agent $n$. With some abuse of notation, we use $k \in \pi(n)$ to denote all the tasks allocated to agent $n$. *Thus, SCoBA satisfies condition (a) from above.*

Now we examine condition (b). Our overall objective is to minimize the expected total cost for tasks that the multi-agent allocation policy $\pi$ fails to complete. Denote this cost as $J(\pi)$. From Eq. 3, we have

$$J(\pi) = \mathbb{E}\left[\sum_{k\in[K]} \mathbb{1}[k] \cdot J(k) \mid \pi\right]. \tag{7}$$

By linearity of expectation, Eq. 7 becomes

$$J(\pi) = \sum_{k\in[K]} \mathbb{E}\big[\mathbb{1}[k] \cdot J(k) \mid \pi\big]. \tag{8}$$

SCoBA resolves conflicts between single-agent allocation policies to ensure that no two agents are allocated to the same task. Therefore, we can split the summation over all tasks based on their allocation to agents and rewrite Eq. 8 as

$$J(\pi) = \sum_{n}\sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k] \cdot J(k) \mid \pi(n)\big]. \tag{9}$$

where, again, we use $k \in \pi(n)$ to denote the tasks uniquely allocated to agent $n$.

Consider the cost function for the single-agent policy in Eq. 6. Recall from Sect. 4.1 that *the minimizing value of the single-agent cost is* $V(\mathrm{root}_n)$ where $\mathrm{root}_n$ is the root of the single-agent optimal plan tree $\pi^*(n)$, i.e.,

$$V(\text{root}_n) = \min_{\pi(n)\in\Pi(n)} \mathbb{E}\left[\sum_{k\in\pi(n)} \mathbb{1}[k]\cdot J(k) \mid \pi(n)\right]. \quad (10)$$

Using linearity of expectation, Eq. 10 becomes

$$V(\text{root}_n) = \min_{\pi(n)\in\Pi(n)} \sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\big]. \quad (11)$$

Finally, using Eqs. 9–11 and that no two agents are allocated to the same task, we have

$$\begin{aligned}\min_{\pi\in\Pi} J(\pi) &= \min_{\pi\in\Pi} \sum_n \sum_{k\in\pi(n)} \mathbb{E}\left[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\right] \\ &= \sum_n \min_{\pi(n)\in\Pi(N)} \sum_{k\in\pi(n)} \mathbb{E}\left[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\right] \quad (12) \\ &= \sum_n V(\text{root}_n),\end{aligned}$$

which is precisely the sum-of-costs of the individual agent solutions. Thus, *SCoBA satisfies condition (b) from above*.

Therefore, SCoBA is optimal in expectation under the given assumptions. □

**Proposition 2** *Under the assumptions of Proposition 1, SCoBA is complete. If a valid allocation exists, SCoBA returns it.*

***Proof*** As with the proof for optimality, our proof for completeness follows that of the original Conflict-Based Search. There, the authors showed completeness by establishing that if a valid multi-agent path exists, the high-level constraint tree has a finite number of nodes, i.e., an upper bound on the number of generated nodes. The upper bound arises because a multi-agent path problem has a finite number of constraints and Conflict-Based Search generates at least one new constraint per new high-level node. Since it executes best-first search with monotonically non-decreasing cost on the constraint tree, it is guaranteed to find a valid solution in finite time if one exists.

We now apply the same reasoning to SCoBA. First, every new high-level node $A$ in SCoBA's constraint tree must have at least one more constraint than its predecessor $A'$, derived from a conflict in the solution that $A'$ represents. Second, the maximum possible number of constraints is the number of ways $K$ tasks can be distributed across $N$ agents multiplied by the time horizon, i.e., $\frac{(K+N-1)!}{K!(N-1)!}\cdot T$.

The finite number of possible constraints and the addition of at least one new constraint per new node implies a finite number of nodes in the constraint tree. SCoBA's high-level routine uses systematic best-first search over the constraint tree, whose expanded nodes have monotonically non-decreasing cost. Therefore, a conflict-free allocation, if it exists, must be found after expanding a finite number of SCoBA constraint tree nodes. Thus, SCoBA is a complete algorithm. □

We reiterate that SCoBA exhibits the above properties only under simplifying assumptions about the multi-robot task allocation problem. In practice, these assumptions do not hold in our experimental domains, with the possible exception of (ii), i.e., running the policy tree search to the full horizon. As we describe in Sect. 3.2, our problem formulation adds two distinct axes of difficulty (task execution uncertainty and new tasks online) on top of an already NP-Hard problem. These assumptions are thus necessary to curb the intractability of the original formulation and make claims that are useful to know even in the restricted setting.

## 4.4 Interleaving planning and execution

We do not assume any particular model for new tasks arriving online as it would depend on the nature of tasks in a specific domain; contrast our task generation parameters in Sect. 5.2 with those in Sect. 5.3. A standard technique for dealing with online updates in allocation or scheduling problems is to *interleave planning and execution*, i.e., compute a solution up to some horizon based on the current information, execute the first part of that solution, and recompute a new solution in either a periodic or event-driven fashion (Church and Uzsoy 1992).

SCoBA is concerned with high-level allocation, so execution in this context simply refers to simulating the attempt of a task by the agent allocated to it (in practice this simulation depends on the specific domain). For replanning, we use the event-driven replanning strategy, where *each event is the outcome of the next task attempt by any agent*. After each such event, we have several updates to consider: the attempted task is either removed or retained in the set of available tasks depending on whether the attempt was a success or failure; all new tasks that arrived between the previous and current event are added; time windows of any existing but yet unattempted tasks may have changed as well. After incorporating all these updates, we re-run SCoBA from scratch on the current problem.

SCoBA's elegant representation makes interleaving planning and execution straightforward at both levels. For the single agent policy tree search, we truncate the search horizon based on domain knowledge or computation requirements. In our implementation, we run the sweep until the first task whose time window begins after the downtime of all tasks before it ($k_3$ in Fig. 3). For the multi-agent coordination, we can set a threshold on the number of high-level conflicts, once again based on domain or computation requirements. If the threshold is exceeded, we return the current high-level solution. In this case, for robots allocated to the same task, we break ties arbitrarily and keep the unassigned robots free for the next event. In principle, we can thus plan for a variable number of robots as well, because the effective number of available robots at each event would change depending on the current status.

### 4.5 Coordination graphs

SCoBA works with any arbitrary configuration of agents, but we can use coordination graphs (CGs) from multi-agent decision-making for more efficiency (Kok et al. 2003). In CGs, each node represents an agent, and each edge encodes a dependency between them, such that only connected agents need to coordinate allocations. The choice of coordination graph for a problem is domain-dependent. For instance, the arms are ordered along the conveyor belt and their workspaces are mutually exclusive, but we want to account for objects near a boundary that may enter the next workspace soon (if unattempted). Thus, the coordination graph is a directed chain from the first arm to the last.

The CG structure impacts the high-level multi-agent coordination stage of SCoBA. The *absence of an edge between two agents implies that their sets of possible tasks are disjoint*, i.e., they cannot have conflicting allocations. Therefore, in practice, SCoBA need not consider dependencies between all the agents. If the CG is directed, as in the conveyor belt, we run the tree search for agents along a topological ordering of the CG. For any agent, we exclude the tasks already assigned to its predecessors. By construction, we will obtain a conflict-free allocation at the end, without any child nodes being generated in the high-level constraint tree). If the CG is undirected, as in multi-drone delivery, such a topological ordering is not feasible, and conflicts may be unavoidable. However, if the CG has multiple connected components, then nodes (agents) in different components cannot conflict with each other, so we can run SCoBA on each component in parallel.

## 5 Experiments and results

The primary metric for evaluating SCoBA is the accumulated penalty for unsuccessful tasks. We will also evaluate its scalability to problem size. We first outline the range of methods we use to baseline SCoBA. We then present and discuss the results for multiple performance metrics on simulations for each of our two distinct robotics-inspired domains: conveyor belt pick-and-place and on-demand multi-drone delivery dispatch. We use the Julia programming language (Bezanson et al. 2017) on a 16 GiB RAM machine and a 6-core 3.7 GHz CPU for all simulations.[1]

### 5.1 Baselines and evaluation

We use multiple complementary algorithms as baselines:

1. **EDD**: The Earliest Due Date heuristic assigns each agent to the task with the nearest time window deadline (Pinedo 2012).
2. **Hungarian**: An unbalanced Hungarian algorithm, where the edge weight for an agent-task pair is the probability of successful task completion (Munkres 1957). This method is a special case of a general purpose network-flow approach that assigns one agent to multiple tasks at a time (Timotheou 2011).
3. **MCTS**: A recent Monte-Carlo Tree Search approach specifically for multi-robot task allocation (Kartal et al. 2016). The tree search is conceptually similar to ours (albeit with Monte Carlo sampling of outcomes) but it uses arbitrary priority orderings among agents to coordinate decisions and control the tree branching factor. For all scenarios, we used 100 Monte Carlo trials to compute each action, an exploration constant of 0.1, and a tree depth of 20.

The baselines cover a range of approaches for multi-robot allocation from scheduling to sequential decision-making under uncertainty. Both EDD and Hungarian are reactive, i.e., do not plan sequentially. The latter optimizes for multiple agents, unlike the former. MCTS is a model-based online method that plans sequentially by framing the allocation problem as a Markov Decision Process. As with SCoBA, we implemented all baselines in Julia. For MCTS, we used the POMDPs.jl framework for modeling and solving Markov Decision Processes (Egorov et al. 2017). The rollout policy of MCTS used the Earliest Due Date heuristic.

We compare SCoBA to the baselines on both the metrics of total unsuccessful tasks and computation time. The latter comparison is not apples-to-apples, because the baselines have different input interfaces that impose different restrictions on problem size, depending on the domain. Furthermore, computation time is not an optimizing metric but rather a satisficing one; our objective for SCoBA's computation time is to be reasonable for the requirements of the respective domains.

### 5.2 Conveyor belt domain

In this domain, three identical robot arms are arranged along a moving conveyor belt, picking objects from the belt and plac-

[1] The code is available at https://github.com/sisl/SCoBA.jl.

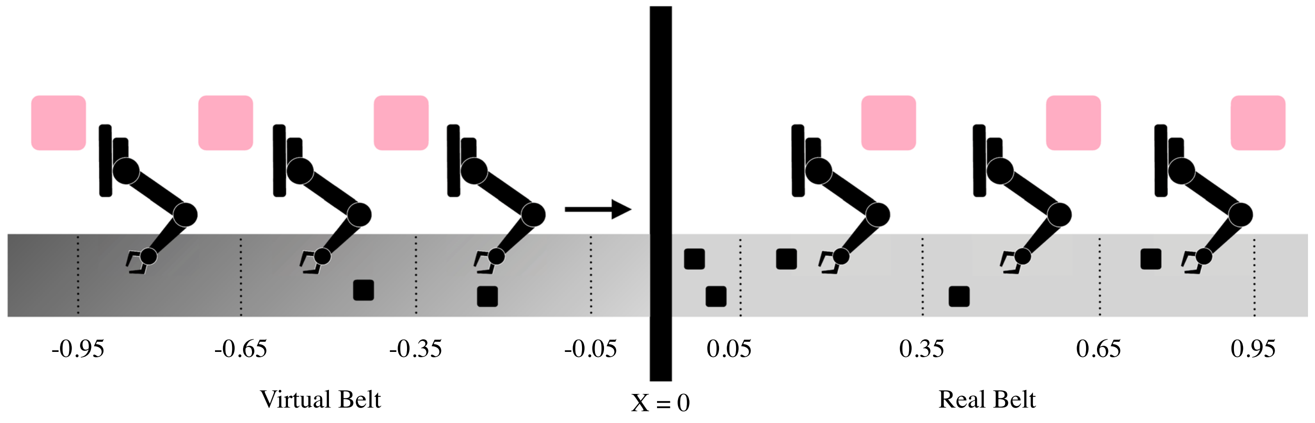


**Fig. 5** We generate tasks for the conveyor belt domain by reflecting the setup in space and time. When the objects placed on the virtual belt (shaded with a gradient fill) by the reflected arms cross the Y-axis, they appear as new tasks on the real belt (with solid grey fill)

ing them in collection bins (Fig. 1a). We design an abstracted simulation of the scenario (Fig. 2), scaled along an X-axis of unit length. The arms have mutually exclusive adjacent workspaces of 0.3 units each, from $x = 0.05$ to $x = 0.95$. New tasks arrive as new objects appearing at the head of the belt.

### 5.2.1 Task generation and scenario parameters

To generate new tasks, we reflect the conveyor belt setup in space and time and create a virtual assembly line (on the left side of Fig. 5) where arms pick objects from bins and place them on a virtual belt. When the virtual belt crosses $x = 0$ (the Y-axis), the virtual objects appear as new real objects. As we explain shortly, this process allows us to generate task sequences that are solvable under the assumption of no grasp uncertainty.

Three scenario parameters instantiate a conveyor belt problem and affect its difficulty:

(i) *Belt speed:* The speed of the belt determines the effective time window for each arm-object pair, e.g., 5 s in Fig. 2. If task execution is successful, each arm has a downtime of $\Delta t = 2$s to deposit the object in the bin. We expect performance to degrade as belt speed increases.

(ii) *New object probability:* At each timestep, a virtual arm drops its object onto the virtual belt with some Bernoulli probability (all virtual arms have the same such probability). The drop location is uniformly sampled within the virtual arm's workspace. After a virtual arm drops an object, it moves to the virtual bin to collect the next one. We expect performance to degrade as new object probability increases.

(iii) *Grasp success probability:* We use two different models of uncertainty over task completion due to imperfect grasping. The first is a geometric distribution where $p$ is the Bernoulli probability of a successful pick by each arm (since arms are identical, they all have the same grasp success probability); the corresponding cumulative probability function from Eq. 1 is

$$\tau_{nk}(t) = 1 - (1 - p)^t, \text{ where } t \in \mathbb{N}. \tag{13}$$

The second is a uniform distribution over the maximum number of time-steps $T_{\max}$ that an object can spend in an arm's workspace, where the total probability of success by each arm is $p$. The corresponding cumulative function from Eq. 1 is

$$\tau_{nk}(t) = \frac{t}{T_{\max}} \cdot p, \text{ where } t, T_{\max} \in \mathbb{N}. \tag{14}$$

We expect performance to improve as $p$ increases, but the second model yields much more adversarial scenarios than the first one.

### 5.2.2 Competitive performance against oracle

For online algorithms like SCoBA, a standard metric is the competitive performance against an oracle with complete information, i.e., full lookahead. Our task generation process allows an *ablation study for the effect of lookahead alone* (deconfounded from uncertainty). The process ensures that when there is no grasp uncertainty, there exists at least one allocation policy that successfully completes all tasks. One such policy is that which reflects the virtual setup itself. For instance, it allocates the first arm, i.e., the arm closest to the Y-axis, to pick up the real objects whose corresponding virtual forms were placed by the first arm's reflection (similarly for other arms). The attempt location for any arm-object pair is the reflection, on the real belt, of the drop point of the corresponding virtual object from the corresponding virtual arm.

**Table 1** The mean proportions of objects lost per trial by SCoBA when grasping is perfect

| Belt speed (units/s) | New object probability | | |
|---|---|---|---|
| | 0.5 | 0.75 | 1.0 |
| 0.04 | 0.0 | 0.0 | 0.0 |
| 0.07 | $2.7 \times 10^{-5}$ | $7.9 \times 10^{-5}$ | $1.3 \times 10^{-4}$ |
| 0.1 | $1.7 \times 10^{-4}$ | $3.7 \times 10^{-4}$ | $5.2 \times 10^{-4}$ |

We vary the other two parameters, that affect the rate of appearance of new tasks. The negligible values demonstrate SCoBA's strong competitive performance relative to the oracle

To obtain the perfect allocation policy in practice (assuming no uncertainty), we would need complete access to the virtual generator for a problem instance. However, we do not need to know the oracle itself, only its performance, which is an upper bound on the performance of any other method. If a perfect allocation policy exists, an oracle with full lookahead would have full success rate for any task sequence generated by our process. We can thus evaluate the competitive performance of SCoBA simply by evaluating the proportion of tasks that it fails to complete. *The smaller this number, the better is SCoBA's competitive performance*.

We set $p = 1$ with the geometric distribution of grasp success probability for all arms and jointly varied the other two parameters, belt speed and new object probability. For each trial in a setting, we simulated $T = 500$ time-steps (seconds) and evaluated the proportion of objects missed by SCoBA relative to the total number of objects. We compute the average of this proportion-per-trial over 100 trials (standard error negligible) in Table 1. The low magnitudes demonstrate SCoBA's robustness to insufficient lookahead. With increasing value of either parameter, performance degrades.

#### 5.2.3 Unsuccessful task penalty

We varied all three scenario parameters independently and compared the fraction of missed objects for SCoBA versus the other baselines. Figure 6 demonstrates the results, with the upper panel for the geometric distribution of grasp probability and the lower panel for the uniform distribution.

We average all numbers over 100 trials with standard error bars. For each subplot, only one parameter varies (the x-label), while the other two stay at their default values: grasp probability $p = 0.75$ for the geometric and $p = 0.8$ for the uniform distributions, 0.07 units/s for belt speed, and 0.75 for new object probability. We use a different range of values for grasp probability in the lower panel (the left-most subplot) because the uniform distribution case is much more adversarial than the geometric one. This increase in difficulty reflects in the much higher absolute values over all settings in the lower panel. Note that $p = 1.0$ in the lower left-most subplot does not mean that grasping is perfect in those scenarios, just that the maximum probability of success from Eq. 14 is 1 (if the arm can attempt the object at its first point of entry into the workspace).

*SCoBA considerably outperforms the other baselines across all settings*. Furthermore, its performance degrades or improves as expected relative to the change in each problem parameter (e.g., more objects missed with increasing new object probability). Among the baselines, the reactive Hungarian method has the best performance, likely because sequential deliberation is not as crucial with non-overlapping workspaces and small downtime (unlike in the next domain).

The MCTS baseline (Kartal et al. 2016) assumed a task environment similar to our drone delivery domain. Thus, for the conveyor belt domain, we had to make our own design choices for the state and action space; we chose to discretize the belt into uniform width slots, each with some number of objects at every time-step, for the system state. The action space for each arm is the set of slots in its corresponding workspace. We used 0.02 units per slot for MCTS, which is a resolution fine enough to ensure that MCTS does not lose any information compared to SCoBA. Too much finer would have led to a prohibitively large state space.

#### 5.2.4 Scalability

In this domain, the Coordination Graph is a directed chain (see Sect. 4), so the computational bottleneck for SCoBA is the policy tree search (multi-agent coordination is trivial). In Table 2 we report average tree search computation times for a single arm with an increasing number of objects (scattered throughout the arm workspace). Empirically, we observe that the computation time is roughly cubic in the number of objects, and the wall clock times are quite reasonable.

The table also reports the computation times for the Hungarian algorithm. It is much faster than SCoBA, but unlike SCoBA, it does not plan sequentially and only matches each arm to one object at a time. EDD is a simple heuristic that does not plan jointly for all agents, so its computation time is not useful. For MCTS, the action computation time is independent of the number of the objects because for the action space, we discretize the conveyor belt into 15 slots per agent. With 100 iterations and a search depth of 20, the average MCTS action takes 0.1 s to compute. Increasing the MCTS computation time with more iterations or samples would not have yielded different decisions and hence would not have changed its performance on the task penalty.

### 5.3 Drone delivery domain

In the second domain, we dispatch drones from depots to deliver packages around a city-scale area, subject to delivery time window constraints. Each drone is associated with

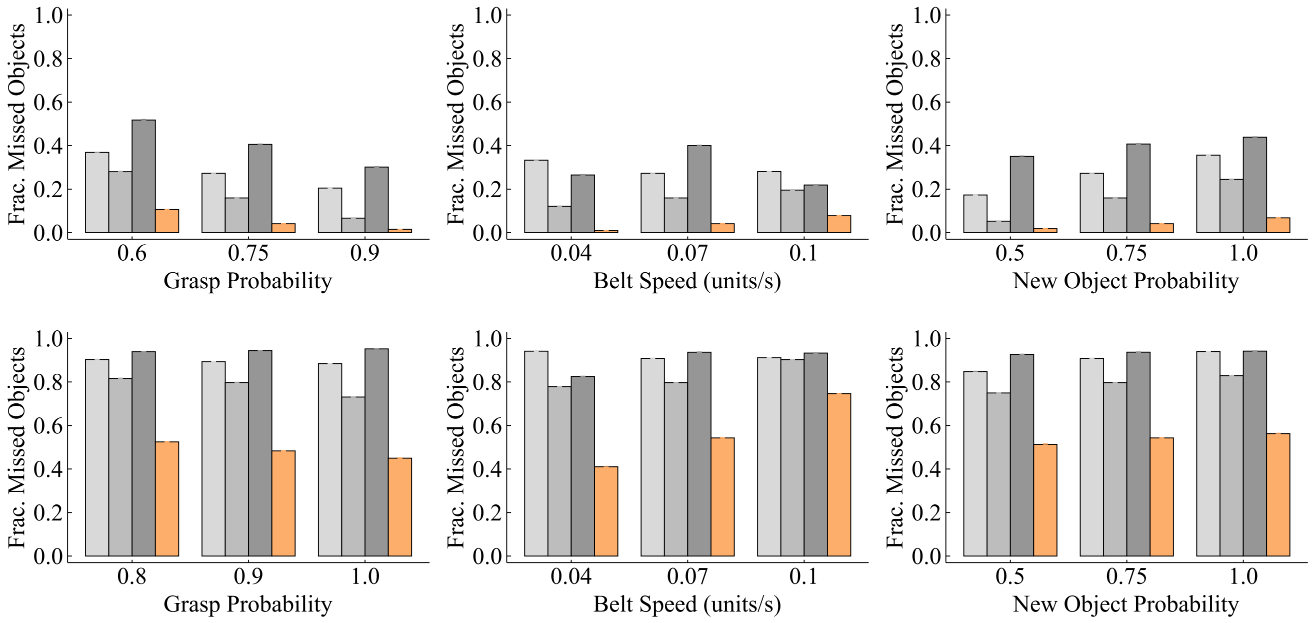


**Fig. 6** Legend: ▢ EDD ▢ Hungarian ▢ MCTS 🟧 SCoBA. All results are averaged over 100 trials, with $T = 500$ time-steps per trial (standard error bars are negligible). Top row: geometric distribution of grasp success probability (Eq. 13). Bottom row: uniform distribution (Eq. 14). The latter yields much more adversarial scenarios than the former, which is why the bars in the lower panel are much higher across the board. On the metric of the fraction of unsuccessful tasks, i.e., objects missed, SCoBA consistently outperforms all other baselines

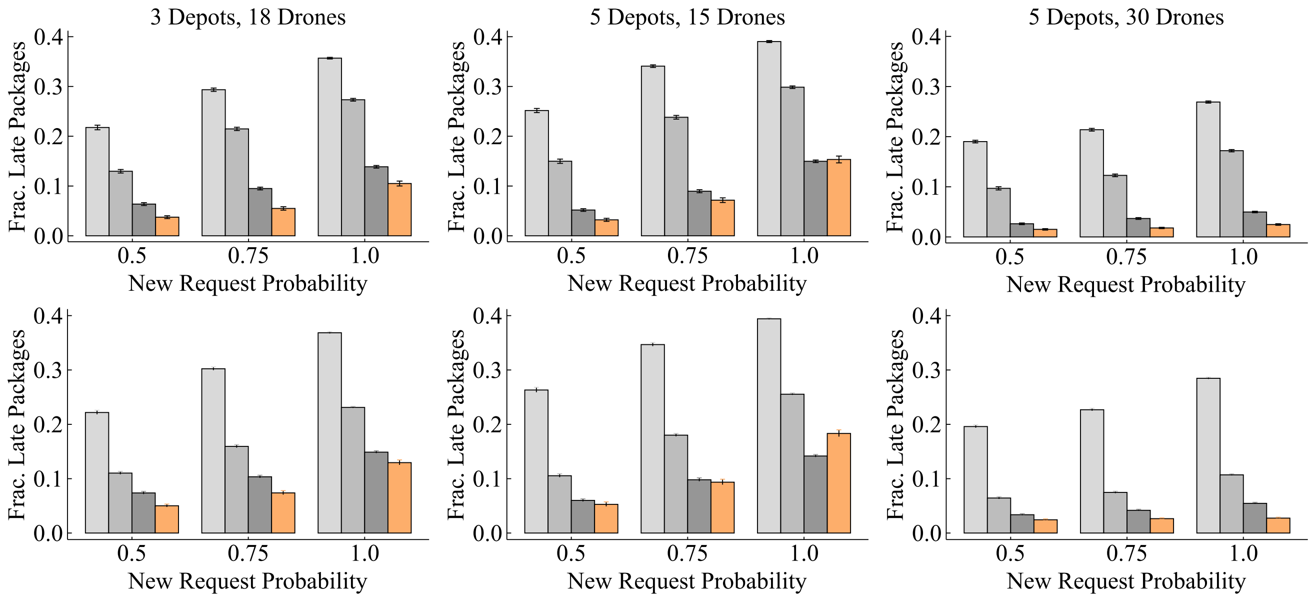


**Fig. 7** Legend: ▢ EDD ▢ Hungarian ▢ MCTS 🟧 SCoBA. Results are averaged over 100 trials, each of $T = 100$ time-steps. The upper and lower panels are respectively for the Epanechnikov (Eq. 15) and Normal (Eq. 16) distributions over travel time. For the drone delivery domain, on the primary metric of the fraction of late package deliveries, SCoBA outperforms the baselines on all but one setting

**Table 2** The low computation times demonstrate that SCoBA's tree search for an individual arm scales with the number of objects in the arm's workspace

| Objects | SCoBA | Hungarian |
|---|---|---|
| 40 | $9 \times 10^{-4}$ s | $1.7 \times 10^{5}$ s |
| 80 | 0.004 s | $3.1 \times 10^{-5}$ s |
| 120 | 0.013 s | $4.3 \times 10^{-5}$ s |
| 160 | 0.029 s | $5.7 \times 10^{-5}$ s |
| 200 | 0.052 s | $8.3 \times 10^{-5}$ s |

The Hungarian algorithm is faster as it solves a one-shot deterministic problem rather than a sequential stochastic one

a specific depot and can carry only one package at a time; therefore, each drone makes trips from its depot to a destination and back to the same depot. Every package request arrives with a specific time window within which a drone from the team must deliver it.

Our setup is based on our recent work for multi-drone delivery over ground transit (Choudhury et al. 2020b). To estimate drone travel time, we use the location-to-location estimates from its North San Francisco scenario, which simulates deliveries over an area of $150\,\text{km}^2$ (see Fig. 1b). We pre-select geographic locations for up to 5 depots scattered around the city to ensure good coverage. Drones have a maximum flight range of 10 km, which restricts the set of possible package deliveries for each drone. Two scenario parameters affect the performance here:

(i) *Drones and depots:* The number of depots and the ratio of drones to depots both impact the ability of the system to dispatch agents to a given delivery location in time. We distribute drones equally across depots. With better coverage, we expect performance to improve.
(ii) *New request probability:* A new delivery request arrives per minute with some probability. Each delivery location is sampled uniformly within a bounding box. We start with a number of packages roughly 1.5 times the number of drones in the scenario. With higher probability, we expect performance to degrade. Each request has a window duration sampled uniformly between 15 and 30 minutes.

Our reference framework gives us deterministic drone travel-time estimates between a depot $d$ and package location $p$, say $TT(d, p)$. Similar to the previous domain, we use two different models for task completion uncertainty, i.e., travel time uncertainty based on $TT(d, p)$: a finite-support Epanechnikov distribution,

$$\tau_{n,k}(t) \sim \text{Epan}(\mu = TT(d, p), r = TT(d, p)/3.0), \quad (15)$$

and an infinite-support Normal distribution,

$$\tau_{n,k}(t) \sim \mathcal{N}(\mu = TT(d, p), \sigma = TT(d, p)/3.0). \quad (16)$$

Unlike the previous domain, the uncertainty models here do not have parameters that we vary to change the problem difficulty, and we do not expect either model to be much more adversarial than the other. In each experiment for both models, the true travel times between two locations are drawn from Eqs. 15 and 16 respectively. Our synthetic distribution choices are arbitrary but reasonable because a high mean travel time is likely to have higher variance, due to more opportunities for delays or speedups.

### 5.3.1 Unsuccessful task penalty

We vary the scenario parameters and compare the fraction of late package deliveries for SCoBA versus the other baselines in Fig. 7. The upper panel is for the Epanechnikov travel time distribution and the lower one is for the Normal travel time distribution. We choose three sets of depot-and-drone numbers with complementary coverage properties, e.g., (3, 18) has fewer depots and a higher drone-depot ratio while (5, 15) has more depots but a smaller ratio. We vary the new request probability, simulate $T = 100$ time-steps (minutes) per trial, and average results over 100 trials. SCoBA is generally the best across all settings except in one (5 depots, 15 drones, probability 1.0) where MCTS is slightly better. Having more drones per depot appears to be more influential than having more depots, e.g., the errors for (5, 15) are higher than the corresponding ones for (3, 18).

The improvement in relative performance of the MCTS baseline (Kartal et al. 2016) is not surprising. It is tailored for vehicle dispatch problems, with search heuristics that exploit the domain structure, e.g., agents have longer downtime to return to their depots, and the per-agent action space is a subset of valid tasks rather than a discretization of a conveyor belt into slots. The drone delivery domain also penalizes MCTS less for its inability to coordinate between agents, compared to the conveyor belt. In drone delivery, each task has a single time window for any agent to complete it in. In conveyor belt, each object has effectively three time windows, one for each arm; unlike MCTS, SCoBA can coordinate between arms such that an earlier one delegates an object to a later arm to improve overall system performance.

### 5.3.2 Scalability

High-level conflicts may occur in this domain, and SCoBA will invoke its multi-agent coordination layer (on top of policy tree search) to compute a valid multi-agent allocation. Therefore, in Table 3 we report the mean and standard error for the computation times of the full SCoBA algorithm (over

**Table 3** The mean and standard error (over 50 trials in each setting) for SCoBA's computation time on the multi-drone delivery domain, as well as the mean times for Hungarian and MCTS

| (Dep., Dr.,) | 20 Requests | | | 50 Requests | | | 100 Requests | | |
|---|---|---|---|---|---|---|---|---|---|
| | SCoBA | Hung | MCTS | SCoBA | Hung | MCTS | SCoBA | Hung | MCTS |
| (3, 18) | (0.02, 0.003) | 0.00008 | 0.007 | (1.48, 0.16) | 0.0001 | 0.008 | (2.55, 0.23) | 0.0002 | 0.009 |
| (5, 15) | (0.06, 0.008) | 0.00008 | 0.006 | (2.13, 0.2) | 0.0001 | 0.007 | (5.42, 0.5) | 0.0002 | 0.009 |
| (5, 30) | (0.17, 0.003) | 0.0002 | 0.01 | (1.76, 0.12) | 0.0002 | 0.013 | (7.08, 0.47) | 0.0003 | 0.016 |

All times are in seconds

50 different trials for each setting). We vary the number of drones and depots and the number of tasks, i.e., the current package delivery requests. The absolute wall clock values are reasonable considering the time-scale of operation of the system in the real world is minutes and hours. Some scenarios have disproportionately high mean and variance due to more high-level conflicts, a known behavioral property of Conflict-Based Search algorithms (Sharon et al. 2012).

Table 3 also shows average times for the Hungarian and MCTS baselines on the same settings; their standard errors are negligible in comparison. As expected, Hungarian is orders of magnitude faster than SCoBA. For MCTS the action space is directly proportional to the number of tasks (unlike in the previous domain) and the computation times vary slightly across settings. The absolute values of MCTS are lower than that of SCoBA; the former imposes an arbitrary ordering on agents and computes allocations for agents one-by-one based on that order, thus circumventing all the complexity of multi-agent coordination. As in the conveyor belt domain, increasing the MCTS computation time with more iterations or samples would not have changed its performance on the task penalty.

## 6 Conclusion

We presented SCoBA, a hierarchical approach for multi-robot task allocation under uncertainty and time constraints. In theory, SCoBA is optimal in expectation and complete under mild technical assumptions. In practice, over two distinct domains, it has strong competitive performance against an oracle, consistently outperforms a number of baselines, and is scalable in terms of computation time to both number of agents and tasks.

### 6.1 Limitations and future work

We assume a known uncertainty model, which is typical for multi-robot task allocation algorithms. However, since SCoBA is based on policy tree search, we could use it in-the-loop with model-based reinforcement learning in case the uncertainty model needs to be estimated online. We focus on high-level allocation here, but we could integrate SCoBA in a full pipeline for robotics applications.

SCoBA's computation time is sensitive to the number of high-level conflicts; future work could run ablation studies to investigate how many conflicts arise in practice, how many it resolves, and what the correlation is with computation time. This analysis could motivate improvements to the multi-agent conflict resolution level such as bounded sub-optimal variants (Barer et al. 2014), better resolution heuristics (Boyarski et al. 2015) and reasoning about the stochasticity of conflicts.

**Acknowledgements** This work was supported by the Ford Motor Company, National Science Foundation Grant Number 1941722 and National Science Foundation Grant Number 1849952.

## References

Ahmed, S., & Garcia, R. (2003). Dynamic capacity acquisition and assignment under uncertainty. *Annals of Operations Research, 124*(1–4), 267–283.

Al-Hinai, N., & ElMekkawy, T. Y. (2011). Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics, 132*(2), 279–291.

Albers, S. (1999). Better bounds for online scheduling. *SIAM Journal on Computing, 29*(2), 459–473.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences, 114*(3), 462–467.

Barer, M., Sharon, G., Stern, R., & Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *European conference on artificial intelligence (ECAI)*, pp. 961–962.

Behrens, J. K., Lange, R., & Mansouri, M. (2019). A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks. In *IEEE international conference on robotics and automation (ICRA)*, IEEE, pp. 8705–8711.

Bertsekas, D. P. (2005). *Dynamic programming and optimal control*. Athena Scientific.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review, 59*(1), 65–98.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Conference on theoretical aspects of* *rationality and knowledge*, Morgan Kaufmann Publishers Inc., pp. 195–210.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., & Shimony, S. E. (2015). ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *International joint conference on artificial intelligence (IJCAI)*, pp. 740–746.

Burkard, R. E., Dell'Amico, M., & Martello, S. (2009). *Assignment problems*, SIAM.

Campbell, T., Johnson, L., & How, J. P. (2013). Multiagent allocation of Markov decision process tasks. In *American control conference (ACC)*, IEEE, pp. 2356–2361.

Chaari, T., Chaabane, S., Aissani, N., & Trentesaux, D. (2014). Scheduling under uncertainty: Survey and research directions. In *International conference on advanced logistics and transport, ICALT*, pp. 229–234.

Che, A., Kats, V., & Levner, E. (2017). An efficient bicriteria algorithm for stable robotic flow shop scheduling. *European Journal of Operational Research, 260*(3), 964–971.

Cheung, R. K., Hang, D. D., & Shi, N. (2005). A labeling method for dynamic driver-task assignment with uncertain task durations. *Operations Research Letters, 33*(4), 411–420.

Choudhury, S., Gupta, J. K., Kochenderfer, M. J., Sadigh, D., & Bohg, J. (2020a). Dynamic multi-robot task allocation under uncertainty and temporal constraints. *Robotics: Science and Systems Foundation*.

Choudhury, S., Solovey, K., Kochenderfer, M. J., & Pavone, M. (2020b). Efficient large-scale multi-drone delivery using transit networks. In *IEEE international conference on robotics and automation (ICRA)*.

Church, L. K., & Uzsoy, R. (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing, 5*(3), 153–163.

Coltin, B., & Veloso, M. M. (2014). Online pickup and delivery planning with transfers for mobile robots. In *IEEE international conference on robotics and automation (ICRA)*, IEEE, pp. 5786–5791.

Cordeau, J., & Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research, 153*(1), 29–46.

Dertouzos, M. L., & Mok, A. K. (1989). Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering, 15*(12), 1497–1506.

Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., & Kochenderfer, M. J. (2017). POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research (JMLR), 18*(26), 1–5.

Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N., Wagner, G., & Surynek, P. (2017). Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Symposium on combinatorial search*.

Framinan, J. M., Fernandez-Viagas, V., & Perez-Gonzalez, P. (2019). Using real-time information to reschedule jobs in a flowshop with variable processing times. *Computers & Industrial Engineering, 129,* 113–125.

Garey, M. R., & Johnson, D. S. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing, 4*(4), 397–411.

Gerkey, B. P., & Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research, 23*(9), 939–954.

Gini, M. L. (2017). Multi-robot allocation of tasks with temporal and ordering constraints. In *AAAI conference on artificial intelligence (AAAI)*, pp. 4863–4869.

Gombolay, M. C., Wilcox, R., & Shah, J. A. (2018). Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics (TRO), 34*(1), 220–239.

González-Neira, E., Montoya-Torres, J., & Barrera, D. (2017). Flowshop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations, 8*(4), 399–426.

Hönig, W., Kiesel, S., Tinka, A., Durham, J., & Ayanian, N. (2018). Conflict-based search with optimal task assignment. In *International conference on autonomous agents and multiagent systems (AAMAS)*.

Hyland, M., & Mahmassani, H. S. (2018). Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign AVs to immediate traveler demand requests. *Transportation Research Part C: Emerging Technologies, 92,* 278–297.

Johannsmeier, L., & Haddadin, S. (2016). A hierarchical human–robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters, 2*(1), 41–48.

Kartal, B., Nunes, E., Godoy, J., & Gini, M. L. (2016). Monte Carlo tree search for multi-robot task allocation. In *AAAI conference on artificial intelligence (AAAI)*, pp. 4222–4223.

Kochenderfer, M. J. (2015). *Decision making under uncertainty: Theory and application*. MIT Press.

Kok, J. R., Spaan, M. T., & Vlassis, N. (2003). Multi-robot decision making using coordination graphs. *International Conference on Advanced Robotics (ICAR), 3,* 1124–1129.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., & Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in neural information processing systems*, pp. 4190–4203.

Lau, H. C., Sim, M., & Teo, K. M. (2003a). Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research, 148*(3), 559–569.

Lau, H. C., Sim, M., & Teo, K. M. (2003b). Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research, 148*(3), 559–569.

Laumond, J. P., et al. (1998). *Robot motion planning and control* (Vol. 229). Springer.

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*, Vol. 1, Elsevier, pp. 343–362.

Lerman, K., Jones, C. V., Galstyan, A., & Mataric, M. J. (2006). Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Researh (IJRR), 25*(3), 225–241.

Lin, X., Janak, S. L., & Floudas, C. A. (2004). A new robust optimization approach for scheduling under uncertainty: I—Bounded uncertainty. *Computers & Chemical Engineering, 28*(6–7), 1069–1085.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning*, Elsevier, pp. 157–163.

Liu, L., & Shell, D. A. (2011). Assessing optimal assignment under uncertainty: An interval-based algorithm. *The International Journal of Robotics Research, 30*(7), 936–953.

Mataric, M. J., Sukhatme, G. S., & Østergaard, E. H. (2003). Multi-robot task allocation in uncertain environments. *Autonomous Robots, 14*(2–3), 255–263.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics, 5*(1), 32–38.

Nunes, E., Manner, M. D., Mitiche, H., & Gini, M. L. (2017). A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems, 90,* 55–70.

Ooonovan, R., Uzsoy, R., & McKay, K. N. (1999). Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research, 37*(18), 4217–4233.

Péret, L., & Garcia, F. (2013). Online resolution techniques. In *Markov decision processes in artificial intelligence*, pp. 153–184.

Pinedo, M. (2012). *Scheduling* (Vol. 29). Springer.

Rahmani, D., & Heydari, M. (2014). Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times. *Journal of Manufacturing Systems, 33*(1), 84–92.

Raman, V., Donzé, A., Sadigh, D., Murray, R. M., & Seshia, S. A. (2015). Reactive synthesis from signal temporal logic specifications. In *International conference on hybrid systems: Computation and control*, pp. 239–248.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. (2012). Conflict-based search for optimal multi-agent path finding. In *AAAI conference on artificial intelligence (AAAI)*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szelke, E., & Kerr, R. M. (1994). Knowledge-based reactive scheduling. *Production Planning & Control, 5*(2), 124–145.

Timotheou, S. (2010). Asset-task assignment algorithms in the presence of execution uncertainty. *The Computer Journal, 54*(9), 1514–1525.

Timotheou, S. (2011). Network flow approaches for an asset-task assignment problem with execution uncertainty. In *Computer and information sciences*, Springer, pp. 33–38.

Vodopivec, T., Samothrakis, S., & Ster, B. (2017). On Monte Carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research, 60,* 881–936.

Wang, Z., & Gombolay, M. (2020). Learning scheduling policies for multi-robot coordination with graph attention networks. *IEEE Robotics and Automation Letters, 5*(3), 4509–4516.

Yan, Z., Jouandeau, N., & Chérif, A. A. (2012). Multi-robot heuristic goods transportation. In *IEEE international conference on intelligent systems*, pp. 409–414.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., & Chi, X. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in neural information processing systems (NIPS),* Vol. 33.

**Shushman Choudhury** is a Ph.D. student in Computer Science at Stanford University. He is generally interested in computational decision-making for intelligent systems and spaces. In his Ph.D., he has developed hierarchical algorithms for coordinated multi-robot networks. His work has been nominated for Best Multi-Robot Paper at ICRA 2020 and featured in BBC Digital Planet. He has also contributed to many open-sourced repositories for planning and decision-making methods. Prior to joining Stanford, Shushman obtained an MS in Robotics from Carnegie Mellon University in 2017 and a B.Tech in Computer Science and Engineering from IIT Kharagpur in 2015.

**Jayesh K. Gupta** is Senior Researcher at Microsoft. Prior to joining Microsoft and during the work in this paper, he was a Computer Science Ph.D. student at Stanford University. He was advised by Mykel Kochenderfer. He is interested in enabling autonomous agency for the real world. To this end, he studies how agents can learn better models of the world for effective control, learn from each other, and work well together in a team. Jayesh is an avid contributor/maintainer of a variety of open-source software in the JuliaPOMDP ecosystem. Previously, Jayesh obtained his Bachelors in Electrical Engineering from IIT Kanpur in 2015.

**Mykel J. Kochenderfer** received the B.S. and M.S. degrees in computer science from Stanford University, Stanford, CA, USA, and the Ph.D. degree from The University of Edinburgh, Edinburgh, U.K. He is currently an Associate Professor of Aeronautics and Astronautics with Stanford University. He is also the Director of the Stanford Intelligent Systems Laboratory, conducting research on advanced algorithms and analytical methods for the design of robust decision-making systems.

**Dorsa Sadigh** is an assistant professor in Computer Science and Electrical Engineering at Stanford University. Her research interests lie in the intersection of robotics, learning, and control theory. Specifically, she is interested in developing algorithms for safe and adaptive human-robot interaction. Dorsa has received her doctoral degree in Electrical Engineering and Computer Sciences (EECS) from UC Berkeley in 2017, and has received her bachelor's degree in EECS from UC Berkeley in 2012. She is awarded the NSF CAREER award, the AFOSR Young Investigator Program Award, the IEEE TCCPS early career award, the Google Faculty Award, and the Amazon Faculty Research Award.

**Jeannette Bohg** is an Assistant Professor of Computer Science at Stanford University. She was a group leader at the Autonomous Motion Department (AMD) of the MPI for Intelligent Systems until September 2017. Before joining AMD in January 2012, Jeannette Bohg was a PhD student at the Division of Robotics, Perception and Learning (RPL) at KTH in Stockholm. In her thesis, she proposed novel methods towards multi-modal scene understanding for robotic grasping. She also studied at Chalmers in Gothenburg and at the Technical University in Dresden where she received her Master in Art and Technology and her Diploma in Computer Science, respectively. Her research focuses on perception and learning for autonomous robotic manipulation and grasping. She is specifically interesting in developing methods that are goal-directed, real-time and multi-modal such that they can provide meaningful feedback for execution and learning. Jeannette Bohg has received several awards, most notably the 2019 IEEE International Conference on Robotics and Automation (ICRA) Best Paper Award, the 2019 IEEE Robotics and Automation Society Early Career Award and the 2017 IEEE Robotics and Automation Letters (RA-L) Best Paper Award.