Dynamic Multi-Robot Task Allocation under Uncertainty and Temporal Constraints

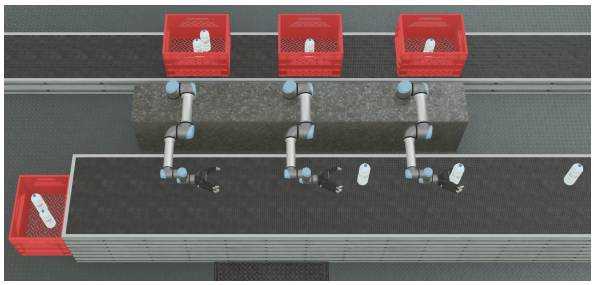
Shushman Choudhury, Jayesh K. Gupta, Mykel J. Kochenderfer, Dorsa Sadigh, and Jeannette Bohg

***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.**

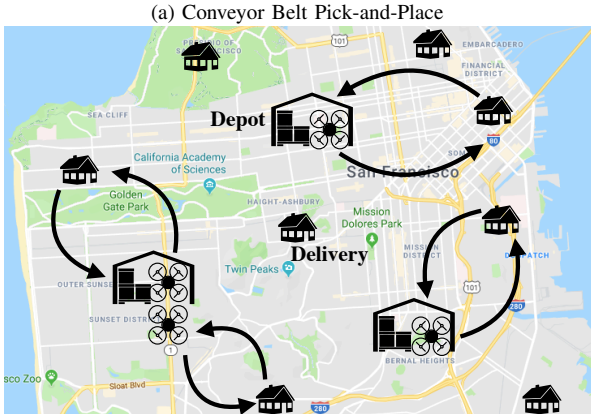
I. Introduction

Efficient and high-quality task allocation is crucial for modern cooperative multi-robot applications [1]. For warehouse logistics, teams of mobile robots carry goods between assigned locations [2]. Industrial and manufacturing operations involve manipulators collaborating on assembly lines [3]. On-demand ridesharing and delivery services dispatch agents to incoming requests [4]. Multi-robot task allocation needs to be computationally efficient and produce high-quality solutions under the challenges of real-world robotics: uncertainty of task execution success, 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 (Figure 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 nature of robotics. 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 [5]. The robotics community has worked on multi-agent task allocation with Markov Decision Processes [6] and robust task matching [7]. The classic multi-robot task allocation problem has been classified extensively [1] and extended to account for uncertainty [8], temporal and ordering constraints [5], and dynamic task arrivals [9] (some of these will be baselines for our method). The operations research community has developed methods for task execution uncertainty [10, 11] and online rescheduling [12]. However, their simplified agent models (such as flow shops) do not address the spatial relationships between tasks or the intersection of uncertainty and time constraints.



(a) Conveyor Belt Pick-and-Place



(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.

The algorithmic challenges for our allocation setting are *sequential planning under uncertainty* and *coordinating multi-agent decisions*. Prior works typically attempt the computationally intractable joint problem and thus require simplifying approximations or heuristics for either planning or coordination. Our key idea is to *decouple these 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

All authors are with Stanford University, CA, USA. Please send correspondence to shushman@cs.stanford.edu

completion. The upper layer uses optimal conflict resolution logic from the path planning community to coordinate the multi-agent allocation decisions [13]. Our overall algorithm, *Stochastic Conflict-Based Allocation* (SCoBA), yields allocation policies that minimize the expected cumulative penalty for unsuccessful tasks; SCoBA can also seamlessly interleave planning and execution online for new tasks.

The following are the contributions of our work:

- We propose a general formulation for multi-robot allocation under task uncertainty and temporal constraints.
- 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.
- We demonstrate SCoBA's strong competitive performance against an oracle with complete lookahead, and its performance advantage over four baseline methods for successful task execution. Our results also show that SCoBA scales with increasing numbers of agents and tasks. We run simulations on two distinct robotics domains (Figure 1); a team of robot arms picking and placing objects from a conveyor belt and on-demand multi-drone delivery of packages in a city-scale area.

II. Background and Related Work

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

Assignment and Scheduling: A major topic in discrete optimization is assigning resources to tasks [14], for which the Hungarian algorithm is a fundamental approach [15]. In temporal tasks, we use the *scheduling* model, where the objective is a function of completed jobs [16]. Scheduling problems with multiple resources and tasks are computationally hard, even when deterministic [17]. In online scheduling, each task is only observed when made available [18]. Hard real-time tasks have a time window constraint for completion [19]. Approaches for scheduling under uncertainty address problems where task execution is not fully deterministic [20, 21]. These approaches are either proactive in anticipating future disruptions [22], reactive to changes [23, 24], or hybrid [25]. For scenarios with ordering constraints (such as assembly lines), additional models like job shops [26] and flow shops [27] are useful, particularly real-time flow shop scheduling [11, 28]. Iterative conflict resolution, which we adapt in our approach, was used for scheduling [29], joint task assignment and pathfinding [30], and diagnosis and repair [31]. This extensive body of work provides valuable insights but does not consider spatial relationships between tasks and their coupling with temporal uncertainty.

Multi-Agent Sequential Decision-Making: The Markov Decision Process (MDP) is a mathematical model for our setting of sequential decision making under uncertainty [32]. Different solution techniques exist for MDPs, depending on available information; dynamic programming [33] when the explicit transition model is known, sample-based online planning [34] when only a generative model exists, and reinforcement learning [35] 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 due to the exponentially large decision space [36]. Although reinforcement learning techniques are often employed to alleviate some tractability issues [37, 38] by learning values of different states and actions, model-free methods like Q-Learning face exploration challenges [38]. Online tree search methods can fare better by focusing on relevant states more effectively [39]. As we will demonstrate, merely framing multi-robot task allocation as an MMDP will not yield good quality solution strategies (due to its computational challenges).

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 [6]. 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 streaming in online [10, 40, 41]. A sensitivity analysis approach to optimal assignment under uncertainty provides some insights on robustness but has no notion of temporal constraints [7]. The taxonomies for multi-robot task allocation under temporal constraints help us characterize our problem's difficulty [1, 42].

Previous work on collaborative assembly lines includes hierarchical planning frameworks, constraint programming, and robust scheduling for robotic flowshops [3, 43, 44]. However, they all simplify one or more 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 [45, 46]. However, they make restrictive assumptions on the uncertainty and environment dynamics [5]. Driver-task assignment with uncertain durations and task windows do solve for a similar setting as ours but assume some knowledge of future requests [47].

III. Problem Formulation

We base our formulation on previous work for multi-robot task allocation with temporal constraints [5]. 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

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

We assume knowledge of this cumulative distribution as part of the problem specification, typical for task scheduling under uncertainty [20]; the particular model is domain-dependent.

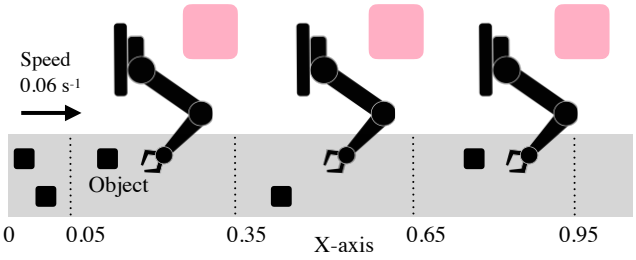


Fig. 2: The illustrated conveyor belt has $N = 3$ arms and $K = 5$ objects. The belt is of unit length, and each arm's workspace spans 0.3 units (dashed lines are the limits). Given the belt speed, the agent-task time window for any arm-object pair is at most $5\,\text{s}$.

By definition, the conjunction of W and τ imposes an upper bound on **task completion probability**, i.e.,

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

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 π 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

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

where the indicator function $\mathbb{1}[k] = 1$ if the task k remains incomplete at the end of the horizon, and Π 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* [16].

The discrete-time rolling horizon formulation described above is fairly general 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 [48]. Furthermore, we can interleave planning and execution suitably and recompute an allocation policy when new tasks appear online (and we do so in practice).

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 Figure 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.

Challenges: To motivate our approach, we briefly discuss the problem complexity. By the multi-robot task allocation taxonomy of Gerkey *et al.* [1], 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. ST-SR-TA problems are an instance of an NP-Hard scheduling problem, specifically multi-agent scheduling with resource constraints [49]. The uncertainty of task execution success exacerbates this difficulty. Time windows make allocation harder by requiring algorithms to account for spatio-temporal task relationships [5]. Finally, new tasks streaming in require our approach to interleave planning and execution effectively, e.g., by replanning at task arrivals [9].

IV. 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 [36]; most closely related previous works either use simplifying approximations for planning and optimization [4, 41] or simple coordination heuristics [8, 50]. 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 discuss in detail the two layers and how they come together in SCoBA. We will then briefly mention how we interleave planning and execution online and how SCoBA can exploit sparse agent interactions using coordination graphs.

A. Low-Level: Single Agent Policy

We consider the perspective of an individual agent, independent of the other ones. From the definition in Section III, we have 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 possible attempt time of a task depends

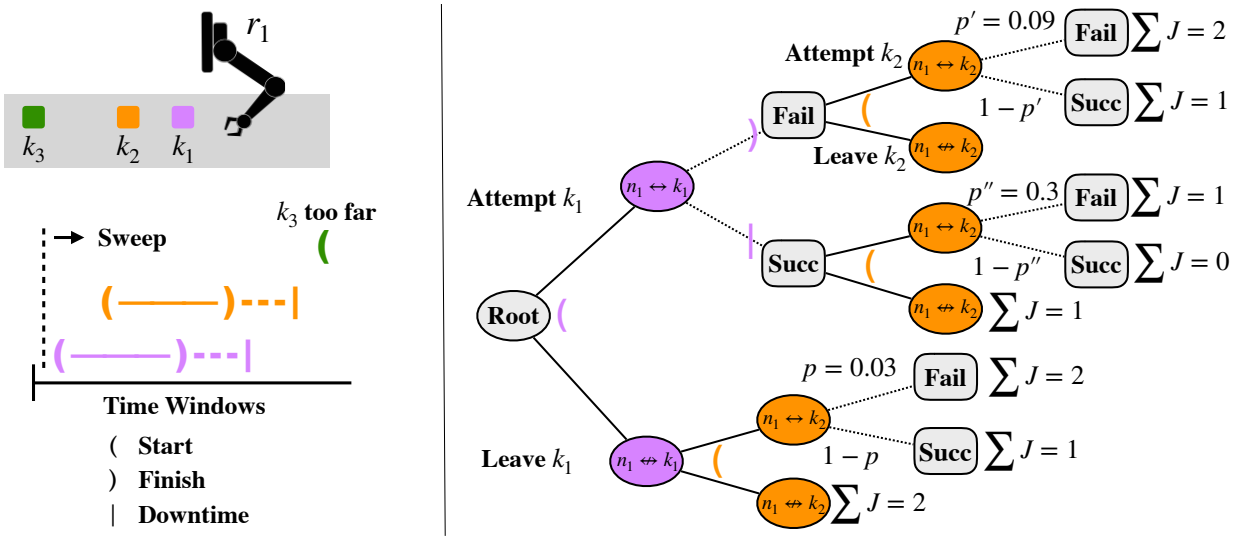


Fig. 3: The low-level routine of SCoBA generates the policy tree over valid tasks for an individual agent, specifically, by sweeping along the time axis and branching on the start or finish of a task's time window. At the start of a window, two new *decision nodes* (ovals) are introduced: *to attempt* ($\leftrightarrow$) 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 $p = 0.03$, $p' = 0.09$, $p'' = 0.3$ 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).

on the tasks attempted before it. We make a simplifying approximation – *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 Figure 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 if the task were to be 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 Figure 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.

The leaves of the binary policy tree contain 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,

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

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

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

in the running example in Figure 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., $n_1 \leftrightarrow k_1$).

B. 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 [51]. Conflict-Based Search is an effective strategy for this problem [13]; by decoupling single-agent path planning and inter-path conflict resolution, it is efficient in practice without losing optimality.

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* (Figure 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

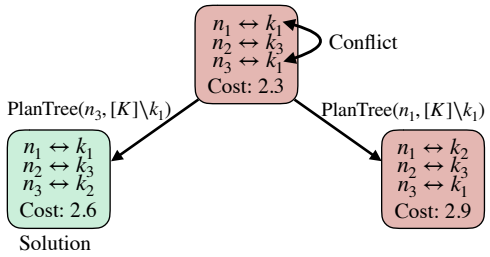


Fig. 4: A constraint tree node with a conflict in the allocation generates two children with corresponding constraints on the conflicting agents (n_1 and n_3) and task (k_1). Best-first search on the constraint tree returns the first high-level node with a conflict-free allocation.

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.

C. Stochastic Conflict-Based Allocation (SCoBA)

Algorithm 1 describes SCoBA, using the tree search of Section IV-A as the PLANTREE subroutine. Its structure is similar to a presentation of Conflict-Based Search by Felner *et al.* [51]. 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, this allocation is returned 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 14). 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 Figure 4. The root node has agents n_1 and n_3 both assigned to task k_1. This conflict yields two 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 18). 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.

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 in expectation the number of incomplete tasks at the end of the time horizon.*

Algorithm 1 Stochastic Conflict-Based Allocation

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

15: procedure GENERATECHILD(S, n, k)
16:     A.constr ← S.constr ∪ k              ▷ Task to exclude
17:     A.soln ← S.soln
18:     A.soln ← PLANTREE(n, [K] \ A.constr, T, W, τ)
19:     A.cost ← SumOfIndividualCosts(A.soln)
20:     return A
```

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

We provide detailed proofs in the appendix. We derive them from the corresponding optimality and completeness proofs of the Conflict-Based Search algorithm for multi-agent pathfinding [13]. For optimality, we use existing results in sequential decision-making to show that SCoBA's low-level routine, i.e., policy tree generation, is optimal in expectation for an individual agent [32]. We then prove that SCoBA's high-level multi-agent coordination satisfies the sufficient condition to inherit the multi-agent optimality of Conflict-Based Search. For completeness, we show how the high-level constraint tree of SCoBA, as in Conflict-Based Search, has a finite number of nodes. Therefore, systematic best-first search on it will find a valid solution if one exists.

Interleaving Planning and Execution: To account for new tasks beyond the horizon, we interleave planning and execution online. SCoBA's elegant representation makes interleaving straightforward at both levels. For the single agent policy tree search, we truncate the search horizon based on 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 Figure 3). For the multi-agent coordination, we set a threshold on the number of high-level conflicts, once again based on real-time computation constraints. If the threshold is exceeded, we return the current high-level solution. For agents allocated to the same task, we break ties arbitrarily and keep the unassigned agents for allocation to new tasks at the next timestep.

Coordination Graphs: SCoBA works with any arbitrary

configuration of agents and inter-agent constraints. We also use coordination graphs (CGs) from multi-agent decision-making for greater efficiency [52]. In CGs, each node represents an agent, and each edge encodes a (potentially directed) dependency between agents, such that only connected agents need to coordinate actions (or allocations in our case). The choice of coordination graph for a problem is domain-dependent and often quite natural. For instance, in the conveyor belt example, the arms are ordered along the belt and their workspaces are mutually exclusive, therefore 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 our multi-drone delivery domain), 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.

V. Evaluation

The primary metric for evaluating SCoBA is the accumulated penalty for unsuccessful tasks. We will also evaluate its scalability to tasks and agents via computation time. We first outline the range of methods we use to baseline SCoBA. We then present and discuss the results for both 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 Julia [53] on a 16 GiB RAM machine and a 6-core 3.7 GHz CPU for all simulations[1].

A. Baselines for Unsuccessful Task Penalty

We use multiple complementary methods to baseline SCoBA on our primary metric of unsuccessful task penalty:

1) **EDD**: The Earliest Due Date heuristic assigns each agent to the task with the nearest time window deadline and is a common heuristic for scheduling [16].
2) **Hungarian**: An unbalanced Hungarian algorithm, where the edge weight for an agent-task pair is the probability of successful task completion [15]. This method is a special case of a general purpose network-flow approach, where only one task is assigned at a time [41].
3) **Q-Learning**: We frame the multi-robot task allocation problem as a discrete-time Markov Decision Process and pre-compute a policy with Q-Learning.
4) **MCTS**: A recent Monte-Carlo Tree Search approach specifically for multi-robot task allocation [50]. 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 the baselines, we 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. Both Q-Learning and MCTS plan sequentially by framing an MDP, but the former is model-free and offline while the latter is model-based and online.

B. Conveyor Belt: Experiments and Results

Three robot arms are arranged along a moving conveyor belt, picking objects from the belt and placing them in collection bins (Figure 1a). We design an abstracted simulation of the scenario (Figure 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 appear at the head of the belt. Three scenario parameters instantiate the problem and affect the difficulty:

- *Grasp Success Probability*: We model uncertainty over task completion due to imperfect grasping with a Bernoulli process where p_i is the probability of a successful pick by the ith arm. Accordingly, the cumulative distribution from Equation (1) is

$$\tau_{nk}(t) = 1 - (1 - p_i)^t, \quad t \in \mathbb{N}. \tag{6}$$

  We expect performance to improve as p_i increases.
- *Belt Speed*: The speed of the belt determines the effective time window for each arm-object pair, e.g., $5\,\mathrm{s}$ in Figure 2. If task execution is successful, each arm has a downtime of $\Delta t = 2\,\mathrm{s}$ to deposit the object in the bin. We expect the performance to degrade as speed increases.
- *New Object Probability*: We briefly describe how new objects arrive (more details in the appendix). We reflect the setup in space (about Y-axis) and time, with virtual arms operating in reverse, transferring objects from virtual bins to a virtual belt. Upon crossing the Y-axis, the virtual belt becomes the true belt and virtual objects appear as real ones. The new object probability parameter is the per-timestep Bernoulli probability with which a virtual arm drops its object onto the virtual belt (all virtual arms have the same such probability). The drop location is sampled uniformly within the virtual arm's workspace. As soon as the 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.

Competitive Performance against Oracle: A standard metric for online algorithms is the competitive performance against an oracle with complete lookahead. The task generation process allows an ablation study for the effect of lookahead alone (decoupled from the effect of uncertainty). If *grasping is perfect*,

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

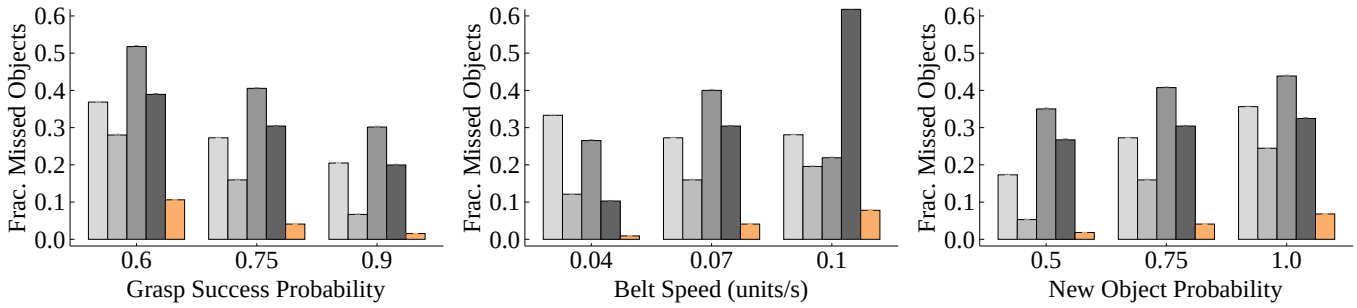


Fig. 5: Legend: EDD Hungarian MCTS Q-Learning SCoBA. On the metric of the fraction of unsuccessful tasks, i.e. objects missed, SCoBA consistently outperforms all other baselines. All results are averaged over 100 trials, with $T = 500$ time-steps per trial.

TABLE I: Average proportions of objects lost per trial by SCoBA when grasping is perfect. The two varied parameters affect SCoBA's lookahead. The negligible values demonstrate the strong competitive performance of SCoBA relative to the oracle.

Belt Speed	New Object Probability		
(units/s)	0.5	0.75	1.0
0.04	0.0	0.0	0.0
0.07	2.7×10^{-5}	7.9×10^{-5}	1.3×10^{-4}
0.1	1.7×10^{-4}	3.7×10^{-4}	5.2×10^{-4}

i.e., $p_i = 1$ for all arms, there exists an oracle strategy that can complete all tasks successfully. The oracle is that which mirrors the generation sequence itself in space and time. We compare SCoBA (without full lookahead) to this oracle by evaluating the proportion of tasks it fails to complete with this generation process. *The smaller this number, the better is SCoBA's competitive performance*. We set $p_i = 1$ for all arms and jointly vary the other two parameters, belt speed and new object probability. We choose a maximum belt speed of 0.1 units/s so that each object spends at least $2\,\mathrm{s}$ in an arm's workspace. For each trial in a setting, we simulate $T = 500$ time-steps (seconds) and evaluate 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 I. The low magnitudes demonstrate SCoBA's robustness to insufficient lookahead. With increasing value of either parameter, performance degrades.

Unsuccessful Task Penalty: We vary all three scenario parameters independently and compare the fraction of missed objects for SCoBA versus the other baselines. Figure 5 demonstrates the results. For each subplot, only one parameter varies (the x-label), while the other two stay at their default values – grasp probability $p_i = 0.75$ for all arms, 0.07 units/s for belt speed, and 0.75 for new object probability. We average all numbers over 100 trials (with standard error bars).

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). For Q-Learning and MCTS, the performance depends on how finely the conveyor belt is discretized. In Q-Learning, the entire state space also needs to be explicitly enumerated. We used a discretization of 0.05 units for Q-Learning and 0.02 units for MCTS; too much finer would result in a prohibitively large state space due to the curse of dimensionality [32].

TABLE II: The low computation time values demonstrate that the tree search for an individual arm is quite scalable with respect to the number of objects in the arm's workspace.

Objects	Tree Size	Comp. Time
40	640.9	$9 \times 10^{-4}\,\mathrm{s}$
80	2215.3	$0.004\,\mathrm{s}$
120	5102.3	$0.013\,\mathrm{s}$
160	8791.7	$0.029\,\mathrm{s}$
200	13028.4	$0.051\,\mathrm{s}$

Scalability: In this domain, the Coordination Graph is a directed chain (see Section IV), so the computational bottleneck for SCoBA is the policy tree search (multi-agent coordination is trivial). In Table II 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 number of tree nodes is roughly quadratic, and the computation time roughly cubic in the number of objects, and the wall clock times are quite reasonable. The appendix has further timing results for the applicable baselines.

C. Drone Delivery: Experiments and Results

In the second domain, we dispatch drones to deliver packages around a city-scale area, subject to delivery time window constraints. Our setup is based on our recent work for multi-drone delivery over ground transit [54]. We use the location-to-location travel time estimates from its North San Francisco scenario, which simulates deliveries over an area of $150\,\mathrm{km}^2$ (see Figure 1b). We handpick locations for up to 5 depots scattered around the city to ensure good coverage. Drones have a maximum flight range of $10\,\mathrm{km}$, which restricts the set of possible package deliveries for each drone. Two scenario parameters affect the performance here:

- *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.

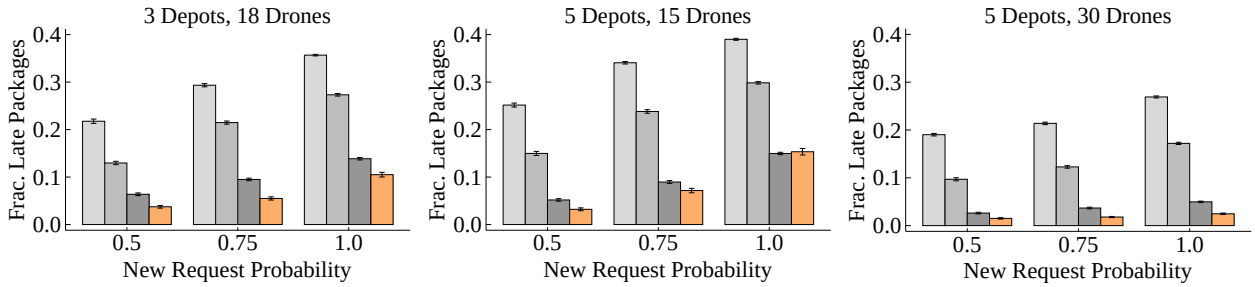


Fig. 6: Legend: EDD Hungarian MCTS SCoBA. For the drone delivery domain, on the primary metric of the fraction of late package deliveries, SCoBA outperforms the baselines on all but one setting. Results are averaged over 100 trials each of $T = 100$ time-steps.

- *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 $TT(d, p)$. We model travel time uncertainty with a finite-support Epanechnikov distribution around $TT(d, p)$, i.e.,

$$\tau_{n,k}(t) \sim \text{Epan}(\mu = TT(d, p), r = TT(d, p)/3.0). \quad (7)$$

The true travel time is drawn from Equation (7). This choice of a synthetic distribution is arbitrary but reasonable because a high mean travel time is likely to have higher variance, due to more opportunities for delays or speedups.

1) Unsuccessful Task Penalty: We vary the scenario parameters and compare the fraction of late package deliveries for SCoBA versus the other baselines in Figure 6. 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. We omit Q-Learning because the enumeration of the multi-agent MDP state space is unacceptably large for any useful time-axis discretization. SCoBA is generally the best across all settings except in one (5 depots, 15 drones, probability 1.0) where MCTS is slightly better. The vast improvement in relative performance of the MCTS baseline [50] 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; the per-agent action space is a subset of valid tasks rather than a discretization of a conveyor belt into slots. For the coverage parameter, 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 $(3, 18)$.

2) Scalability: In this domain, high-level conflicts may occur, and SCoBA will invoke its multi-agent coordination layer (over and above policy tree search) while computing a valid multi-agent allocation. Therefore, in Table III we report the mean and standard error for computation times for the full SCoBA algorithm (over 50 different trials for each setting). We vary the number of drones and depots and the number of currently available tasks, i.e., the current package delivery requests. The absolute wall clock values are quite reasonable given that the time-scale of operation of the system in the real world is minutes and hours. Some scenarios have disproportionately high mean and variance because of more high-level conflicts, a known behavioral property of Conflict-Based Search algorithms [13]. The appendix has timing results for other baselines.

TABLE III: The mean and standard error (over 50 trials in each setting) for SCoBA computation times on the multi-drone delivery domain. All times are in seconds.

	Number of Requests		
(Depots,Drones)	20	50	100
$(3, 18)$	$(0.02, 0.003)$	$(1.48, 0.16)$	$(2.55, 0.23)$
$(5, 15)$	$(0.06, 0.008)$	$(2.13, 0.2)$	$(5.42, 0.5)$
$(5, 30)$	$(0.17, 0.003)$	$(1.76, 0.12)$	$(7.08, 0.47)$

VI. Conclusion

We presented SCoBA, a hierarchical approach for dynamic multi-robot task allocation under uncertainty and temporal constraints. In theory, SCoBA is optimal in expectation and complete under mild technical assumptions. In practice, over two distinct and realistic 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 agents and tasks.

Limitations and Future Work: We assume a known uncertainty model, which is typical for multi-robot task allocation research. However, since SCoBA is based on policy tree search, we could use it in-the-loop with model-based RL in case the uncertainty model needs to be estimated online. SCoBA's computation time is sensitive to the number of high-level conflicts; future work could incorporate efficiency improvements to Conflict-Based Search such as bounded sub-optimal variants [55] and improved conflict resolution [56]. Finally, we focus on high-level allocation here, but we could integrate SCoBA in a full pipeline for robotics applications.

Acknowledgments

This work was supported by the Ford Motor Company, NSF grant number 1241349 and NSF grant number 138329.

References

[1] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[2] Z. Yan, N. Jouandeau, and A. A. Chérif, "Multi-robot heuristic goods transportation," in *IEEE International Conference on Intelligent Systems*, 2012, pp. 409–414.

[3] 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.

[4] M. Hyland and H. S. Mahmassani, "Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign AVs to immediate traveler demand requests," *Transportation Research Part C: Emerging Technologies*, vol. 92, pp. 278–297, 2018.

[5] M. L. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2017, pp. 4863–4869.

[6] T. Campbell, L. Johnson, and J. P. How, "Multiagent allocation of Markov decision process tasks," in *American Control Conference (ACC)*, IEEE, 2013, pp. 2356–2361.

[7] L. Liu and D. A. Shell, "Assessing optimal assignment under uncertainty: An interval-based algorithm," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 936–953, 2011.

[8] M. J. Mataric, G. S. Sukhatme, and E. H. Østergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2-3, pp. 255–263, 2003.

[9] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: Models and algorithms," *Annals of Operations Research*, vol. 153, no. 1, pp. 29–46, 2007.

[10] S. Timotheou, "Asset-task assignment algorithms in the presence of execution uncertainty," *The Computer Journal*, vol. 54, no. 9, pp. 1514–1525, 2010.

[11] D. Rahmani and M. Heydari, "Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times," *Journal of Manufacturing Systems*, vol. 33, no. 1, pp. 84–92, 2014.

[12] R. O'Donovan, R. Uzsoy, and K. N. McKay, "Predictable scheduling of a single machine with breakdowns and sensitive jobs," *International Journal of Production Research*, vol. 37, no. 18, pp. 4217–4233, 1999.

[13] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[14] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. SIAM, 2009, ISBN: 978-0-89871-663-4.

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

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

[17] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," in *Annals of Discrete Mathematics*, vol. 1, Elsevier, 1977, pp. 343–362.

[18] S. Albers, "Better bounds for online scheduling," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 459–473, 1999.

[19] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[20] T. Chaari, S. Chaabane, N. Aissani, and D. Trentesaux, "Scheduling under uncertainty: Survey and research directions," in *International Conference on Advanced Logistics and Transport, ICALT*, 2014, pp. 229–234.

[21] D. Sadigh and A. Kapoor, "Safe control under uncertainty with probabilistic signal temporal logic," in *Proceedings of Robotics: Science and Systems (RSS)*, Jun. 2016.

[22] X. Lin, S. L. Janak, and C. A. Floudas, "A new robust optimization approach for scheduling under uncertainty:: I. bounded uncertainty," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 1069–1085, 2004.

[23] E. Szelke and R. M. Kerr, "Knowledge-based reactive scheduling," *Production Planning & Control*, vol. 5, no. 2, pp. 124–145, 1994.

[24] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*, 2015, pp. 239–248.

[25] L. K. Church and R. Uzsoy, "Analysis of periodic and event-driven rescheduling policies in dynamic shops," *International Journal of Computer Integrated Manufacturing*, vol. 5, no. 3, pp. 153–163, 1992.

[26] N. Al-Hinai and T. Y. ElMekkawy, "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm," *International Journal of Production Economics*, vol. 132, no. 2, pp. 279–291, 2011.

[27] E. González-Neira, J. Montoya-Torres, and D. Barrera, "Flow-shop scheduling problem under uncertainties: Review and trends," *International Journal of Industrial Engineering Computations*, vol. 8, no. 4, pp. 399–426, 2017.

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

[29] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus, "Conflict ordering search for scheduling problems," in *International Conference on Principles and Practice of Constraint Programming*, Springer, 2015, pp. 140–148.

[30] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *International*

Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2016, pp. 1144–1152.

[31] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donzé, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia, "Diagnosis and repair for synthesis from signal temporal logic specifications," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, 2016, pp. 31–40.

[32] M. J. Kochenderfer, *Decision Making under Uncertainty: Theory and Application*. MIT Press, 2015.

[33] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.

[34] L. Péret and F. Garcia, "Online resolution techniques," *Markov Decision Processes in Artificial Intelligence*, pp. 153–184, 2013.

[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[36] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, Morgan Kaufmann Publishers Inc., 1996, pp. 195–210.

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

[38] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4190–4203.

[39] 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.

[40] S. Ahmed and R. Garcia, "Dynamic capacity acquisition and assignment under uncertainty," *Annals of Operations Research*, vol. 124, no. 1-4, pp. 267–283, 2003.

[41] S. Timotheou, "Network flow approaches for an asset-task assignment problem with execution uncertainty," in *Computer and Information Sciences*, Springer, 2011, pp. 33–38.

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

[43] J. K. Behrens, R. Lange, and M. Mansouri, "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, 2019, pp. 8705–8711.

[44] A. Che, V. Kats, and E. Levner, "An efficient bicriteria algorithm for stable robotic flow shop scheduling," *European Journal of Operational Research*, vol. 260, no. 3, pp. 964–971, 2017.

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

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

[47] R. K. Cheung, D. D. Hang, and N. Shi, "A labeling method for dynamic driver-task assignment with uncertain task durations," *Operations Research Letters*, vol. 33, no. 4, pp. 411–420, 2005.

[48] J.-P. Laumond *et al.*, *Robot Motion Planning and Control*. Springer, 1998, vol. 229.

[49] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 397–411, 1975.

[50] B. Kartal, E. Nunes, J. Godoy, and M. L. Gini, "Monte Carlo tree search for multi-robot task allocation," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2016, pp. 4222–4223.

[51] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Symposium on Combinatorial Search*, 2017.

[52] J. R. Kok, M. T. Spaan, and N. Vlassis, "Multi-robot decision making using coordination graphs," in *International Conference on Advanced Robotics (ICAR)*, vol. 3, 2003, pp. 1124–1129.

[53] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[54] S. Choudhury, K. Solovey, M. J. Kochenderfer, and M. Pavone, "Efficient large-scale multi-drone delivery using transit networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[55] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *European Conference on Artificial Intelligence (ECAI)*, 2014, pp. 961–962.

[56] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. E. Shimony, "ICBS: improved conflict-based search algorithm for multi-agent pathfinding," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 740–746.

[57] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research (JMLR)*, vol. 18, no. 26, pp. 1–5, 2017.

APPENDIX

OPTIMALITY AND COMPLETENESS PROOFS

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 in expectation the number of incomplete tasks at the end of the time horizon.*

Proof. Conflict-Based Search has been proved to yield 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 the individual agent solutions. SCoBA uses the same high-level multi-agent conflict resolution logic as Conflict-Based Search, and will inherit this optimality property if it satisfies the two sufficient conditions.

We first show (a) is true for SCoBA. The low-level single-agent routine uses dynamic programming with forward tree search to obtain a policy tree. Such a method, by construction, computes a policy from the initial state that is optimal under expectation for a discrete, finite-horizon Markov Decision Process (MDP) *if the tree search is exhaustively conducted up to the full horizon*. The expectation is over the uncertainty of the outcomes of actions. Assumption (i) ensures that all information of future tasks is known at the initial state of the agent and Assumption (ii) ensures the exhaustive tree search.

Recall the simplifying approximation we made along the temporal dimension, by treating time-windows as discrete events. With Assumption (iii), the temporal approximation now becomes exact. i.e., by attempting each task at the first possible time-step and continuing until the end of the window, each task attempt has a single probability mass function over the two possible outcomes, success or fail. Therefore, the single agent problem can be framed as a discrete finite-horizon MDP and the low-level policy tree search routine is optimal in expectation for an individual agent. That is, for each agent n, the policy $\pi^*(n)$ obtained from PLANTREE satisfies

$$\pi^*(n) = \underset{\pi(n)\in\Pi(n)}{\operatorname{argmin}} \mathbb{E}\Big[\sum_{k\in\pi(n)} \mathbb{1}[k]\cdot J(k) \mid \pi(n)\Big] \tag{8}$$

subject to the constraints in Equation (3) of the main text, 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 let us show (b) holds for SCoBA. The true overall objective in SCoBA is the expected cumulative penalty of unsuccessful tasks due to the computed multi-agent allocation policy π. For convenience, denote this objective as $J(\pi)$. Then, once again from Equation (3) in the main text, we have

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

By the linearity of expectation, we rewrite Equation (9) as

$$J(\pi) = \sum_{k\in[K]} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi\big]. \tag{10}$$

By construction, 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 by the tasks allocated to agents and rewrite Equation (10) as

$$J(\pi) = \sum_{n}\sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\big]. \tag{11}$$

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

Now, consider the objective function for the single-agent policy in Equation (8). Recall from Section IV-A of the main text that *the minimizing value of the single-agent objective is* $V(\text{root}_n)$ where root_n is the root of the single-agent optimal plan tree $\pi^*(n)$, i.e.

$$V(\text{root}_n) = \min_{\pi(n)\in\Pi(n)} \mathbb{E}\Big[\sum_{k\in\pi(n)} \mathbb{1}[k]\cdot J(k) \mid \pi(n)\Big]. \tag{12}$$

By using linearity of expectation again, we can write Equation (12) as

$$V(\text{root}_n) = \min_{\pi(n)\in\Pi(n)} \sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\big]. \tag{13}$$

Finally, using Equations (11)–(13) and that no two agents are allocated to the same task, we have

$$\begin{aligned}
\min_{\pi\in\Pi} J(\pi) &= \min_{\pi\in\Pi} \sum_{n}\sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\big] \\
&= \sum_{n} \min_{\pi(n)\in\Pi(N)} \sum_{k\in\pi(n)} \mathbb{E}\big[\mathbb{1}[k]\cdot J(k) \mid \pi(n)\big] \\
&= \sum_{n} V(\text{root}_n)
\end{aligned} \tag{14}$$

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 same assumptions as Proposition 1, SCoBA is complete. If a conflict-free allocation exists, SCoBA will return it.*

Proof. As with the proof for optimality, our proof for completeness follows that for the original Conflict-Based Search algorithm. In the case of Conflict-Based Search, completeness was shown by establishing that the high-level constraint tree has a finite number of nodes, i.e., an upper bound on how many nodes can be generated, if a valid multi-agent path does exist. Because Conflict-Based Search generates at least one new constraint per new high-level node, and 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.

Let us now apply the same reasoning to SCoBA. First, every new high-level node A in SCoBA's constraint tree must have at least one additional constraint as compared to its predecessor

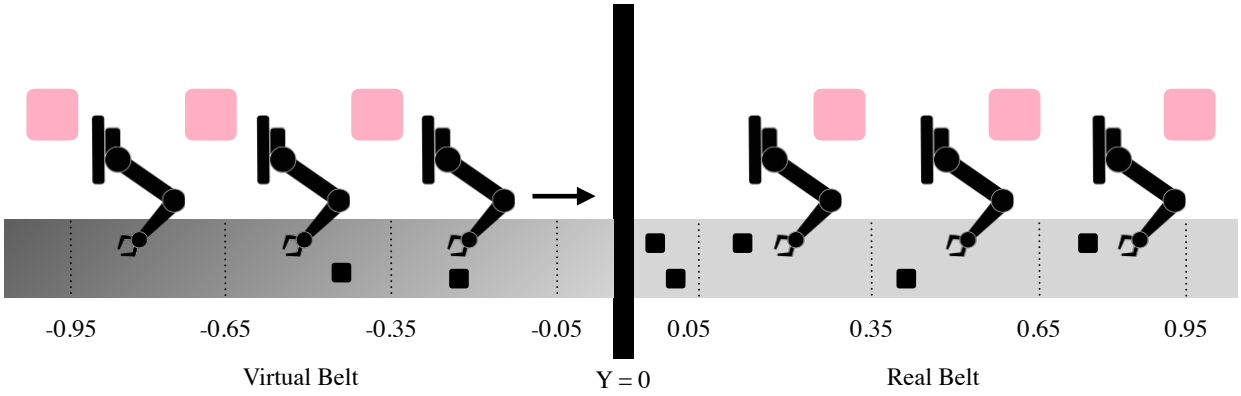


Fig. 7: The task generation process for the conveyor belt domain is defined by reflecting the real setup in space and time to obtain a virtual setup.

A', derived from a conflict in the solution that A' represents. Second, the total number of possible constraints is finite. Specifically, the maximum possible number of constraints is the number of ways K tasks can be distributed across N agents (obtained from standard combinatorics results) multiplied by the time horizon, i.e. $\frac{(K+N-1)!}{K!(N-1)!} \cdot T$. Therefore, the finite number of possible constraints and the addition of at least one new constraint per new node implies that there is a finite number of nodes in the constraint tree. The high level routine in SCoBA uses systematic best-first search over the constraint tree, whose expanded nodes have monotonically non-decreasing cost (by construction). 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. □

Additional Evaluation Details

Conveyor Belt Task Generation Process

We discuss in more detail the conveyor belt task generation process mentioned in Section V-B of the main text. Figure 7 provides a supporting illustration for the same. We reflect the belt setup in both space and time and create a virtual assembly line (on the left side of the figure) where arms pick up objects from bins and place them on the virtual belt. When the virtual belt crosses over, the virtual objects appear as new real objects.

Recall that the above task generation process, by construction, implies the existence of at least one allocation strategy that is perfect, i.e., that successfully executes all tasks when there is no uncertainty. One perfect allocation strategy is that which reflects the virtual setup that generates the tasks. By this strategy, for instance, the first arm, i.e., the arm closest to the belt origin, is allocated to pick up those real objects whose corresponding virtual forms were placed on the belt by the arm's reflection. The attempt location is the reflection (on the real belt) of the point in the virtual arm's workspace where it drops the virtual object.

In practice, of course, we would require complete access to the virtual generator in order to obtain the perfect allocation strategy (assuming no uncertainty) for a given problem instance.

TABLE IV: Timing comparisons – Conveyer Belt.

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

However, note that evaluating the competitive performance of an online algorithm with respect to an oracle (as we do in the main text) only requires us to know the performance of an oracle, which is an upper bound on the performance of any other method. The existence of a perfect allocation strategy implies that an oracle with access to all future information would have full success rate for any task sequence generated by our process. We can thus effectively compute the competitive performance of SCoBA (or any other approach) simply by evaluating the fraction of unsuccessful tasks (missed boxes) of that approach.

Baseline Implementation Details

Like SCoBA, all baselines were implemented in the Julia language [53]. For both MCTS and Q-Learning, we used the implementations in the POMDPs.jl framework [57] for modeling and solving Markov Decision Processes. For the rollout policy of MCTS, we used the EDD heuristic itself, which was better than random rollout. For Q-Learning, we used ϵ-greedy exploration with ϵ scaled linearly with a decay rate of 0.9995. The learning rate was 0.01 and the policy was trained for 100000 steps.

Baseline Computation Time Results

As we had flagged in the main text, we now report the computation times of some baseline approaches in our two simulation domains. The comparison is not apples-to-apples, because the baselines have different input interfaces that impose different restrictions on the full problem size, depending on the domain (for instance, the MDP approaches in the conveyor belt domain are unaffected by the number of tasks due to the action space being the belt slots obtained by discretizing it). Furthermore, computation time for us is not

TABLE V: Timing comparisons – Drone Delivery. All times are in seconds.

	20 Requests			50 Requests			100 Requests		
(Depots,Drones)	SCoBA	Hungarian	MCTS	SCoBA	Hungarian	MCTS	SCoBA	Hungarian	MCTS
$(3, 18)$	0.02	8.4×10^{-5}	0.007	1.48	1×10^{-4}	0.008	2.55	2×10^{-4}	0.009
$(5, 15)$	0.06	8.3×10^{-5}	0.006	2.13	1×10^{-4}	0.007	5.42	2×10^{-4}	0.009
$(5, 30)$	0.17	2×10^{-4}	0.01	1.76	2×10^{-4}	0.013	7.08	3×10^{-4}	0.016

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 (which they are, as we discuss in the main text).

Conveyor Belt: In Table IV we compare the computation time of the Hungarian baselines with that of SCoBA's policy tree search, with varying objects. The times in the SCoBA column are copied over from Table II. Clearly, Hungarian approach is much faster than SCoBA, but unlike SCoBA it does not plan sequentially and only matches an arm to objects within the arm workspace (so the effective number of objects considered per arm is about a third of the total number, for each of the three arms). For MCTS, the action computation time is independent of the number of the objects because the action space is the discretization of the conveyor belt into 15 slots per agent. With 100 MCTS iterations and a search depth of 20, the average action computation time is 0.1 s.

The computation times for EDD and Q-Learning are not informative; EDD is a simple heuristic that does not plan jointly for all agents, and in Q-learning the tabular policy is precomputed offline and simply looked up online (and like MCTS the action space is a discretization of belt slots and does not depend on the number of objects).

Drone Delivery: In Table V, we compare mean computation times of the Hungarian and MCTS baselines to those of the full SCoBA algorithm, with varying depots, drones, and requests considered. The values in the SCoBA sub-columns are copied over from the corresponding entries in Table III (the first quantity of the mean and standard error tuple). As expected, Hungarian is orders of magnitude faster than SCoBA. For the MCTS baseline, the action space is now directly proportional to the number of tasks, i.e., requests considered, and the computation times do vary accordingly. The absolute numbers are much lower than SCoBA primarily due to the MCTS baseline circumventing the complexity of multi-agent coordination by imposing an arbitrary ordering on the assignment of agents.