# Robust Task Scheduling for Multi-Operator Supervisory Control Missions

Jeffrey R. Peters [1]
*University of California, Santa Barbara,*

*Santa Barbara, CA, 93106-5070, USA*

Luca F. Bertuccelli [2]
*United Technologies Research Center, East Hartford, CT, 06118-1127, USA*

**In future human-UAV collaborative missions, especially those involving multiple human operators, the issue of resource allocation will be a crucial component to system success. Traditional deterministic strategies for task allocation and scheduling, such as those designed for job-shop applications, can often lead to poor performance in human-centered systems, since these strategies fail to account for operator cognitive requirements or for the large amounts of uncertainty in human behavior. In light of this, we present a flexible framework that can potentially address both of these issues in finite horizon scheduling applications involving multiple operators. Specifically, we illustrate how operator task load constraints can be formulated as a part of a mixed-integer linear program scheduling framework, which also incorporates robustness to uncertain processing times through the use of scenarios. We then explore the utility and modularity of this framework through the introduction of adaptive components that can further mitigate uncertainty and potentially boost performance. Finally, we propose a heuristic, auction-based strategy for task allocation in order to reduce computation time to reasonable levels. Throughout our discussion, we use numerical examples to discuss the functionality of these algorithms, as well as discuss various considerations for future practical implementation.**

---

[1] Graduate Student Researcher; Department of Mechanical Engineering; jrpeters@engineering.ucsb.edu
[2] Staff Research Engineer; Systems Department; bertuclf@utrc.utc.com

**Nomenclature**

*Mathematical Operators*

$\lceil a \rceil$ The least integral upper bound of $a \in \mathbb{R}$, i.e., $\lceil a \rceil \coloneqq \min\{b \in \mathbb{Z} : b \geq a\}$

$\sqcup_{i=1}^{A} A_i$ Disjoint union of the sets $A_i$, i.e., $\sqcup_{i=1}^{A} A_i \coloneqq \{(a, i) : i \in \{1, \ldots, A\}, a \in A_i\}$

*Indices*

$K$ The number of operators

$N$ The number of (non-null) tasks

$M$ The number of task slots

$Q$ The number of scenarios

$P$ The number of sources (Section III B)

$T$ The number of task types (Section III B)

*Parameters*

$\underline{w}, \overline{w}$ Lower and upper task load bounds

$\zeta$ The length of the null-task

$T_H$ The (finite) time horizon

$p_\alpha$ A MILP parameter used to penalize schedules that are predicted to exceed $T_H$

$p_\beta, p_\gamma$ MILP parameters used to penalize schedules that are predicted to violate task load bounds

*Task Parameters*

$\mathcal{T}$ The set of tasks in the task pool (including the null-task), i.e., $\mathcal{T} \coloneqq \{T_i\}_{i \in \{1,\ldots,N+1\}}$

$\delta w_i$ The task load increment associated with performing task $T_i$

$r_i$ The reward obtained upon successful completion of task $T_i$

$s_i$ The (global) time at which the $i$-th task becomes available

$t_i^m$ The processing time of task $T_i$ according to the $m$-th scenario

$\phi_i$ The source of task $T_i$ (Section III B)

$\tau_i$ The type of task $T_i$ (Section III B)

*Optimization and Assignment Variables*

$x_{i,j,k}$ Binary decision variable; indicates if $T_i$ is performed in the $j$-th slot of operator $k$'s sequence

$W_{0,k}$ Operator $k$'s initial task load

$\alpha, \beta, \gamma$ Decision variables that quantify the amount that horizon and task load bounds are exceeded.

$B_{j,k}^m$ The start time of the $j$-th task in operator $k$'s sequence in the $m$-th scenario

$C_{j,k}^m$ The completion time of the $j$-th task in operator $k$'s sequence in the $m$-th scenario

$W_{j,k}^m$ Operator $k$'s task load level after processing the $j$-th task in the sequence in the $m$-th scenario

$\lambda$ Discount parameter (contained in the set $[0, 1]$) used in heuristic task assignment (Section IV)

$\bar{\epsilon}_i, \underline{\epsilon}_i, \epsilon_i$ Process variables used in the heuristic assignment of Section IV

$\pi_i$ The score assigned to $T_i$ in the heuristic assignment of Section IV

## I. Introduction

In futuristic mission scenarios involving collaboration between humans and unmanned aerial vehicles (UAVs), human operators are often charged with sequential processing of tasks that are generated by their UAV partners. In fact, in many cases, there is not one, but many human operators that supervise a single UAV or team of UAVs. For example, the multi-sensor system Gorgon Stare utilizes a team of intelligence analysts, distributed across various workstations, to process large databases of imagery generated by unmanned vehicles [1]. Proper scheduling of the tasks generated by UAVs in supervisory missions can have a profound impact on both operator and mission performance, since mission planning decisions are usually intimately linked to operators' abilities to process tasks quickly and accurately [2, 3]. In the presence of multiple operators, it is necessary to simultaneously allocate tasks and determine the order in which tasks should be processed, leading to a discrete problem instance whose optimal solution is not straightforward.

These types of discrete scheduling problems are known to be NP-hard (non-deterministic polynomial-time hard) in the general case [4]. Despite this, the traditional deterministic scheduling problem and its relevant variations have been considered for a number of years in the context of job-shop applications, that is, applications involving the coordination and scheduling of jobs or tasks that each require specific resources and have specific timing constraints (e.g., [5]), and a variety of high-quality heuristic methods exist for constructing effective solutions. Some common strategies involve the use of integer programming [6], disjunctive graphs [7], and various heuristics [8]. Although these strategies are well-established in deterministic settings, they are often ill-suited for human-centered applications. The main reason for this is due to the large amounts of uncertainty that are involved in human behavior. For example, virtually all models of human cognitive processing in visual search are stochastic in nature, and thus task processing times will usually carry significant uncertainty. Although some strategies do exist for multiple resource allocation and discrete scheduling in uncertain or dynamic environments (e.g., [9]), and robust optimization methods are feasible in some circumstances [10], other exogenous factors that are generally unique to human-centered systems can cause even these strategies to fail. For example, factors such as cognitive load, fatigue, memory retention, among others, all have some effect on operator performance in persistent task execution missions. In general, the particular effect of any one of the aforementioned factors is task and individual dependent. However, there are some generally accepted relationships that are often useful in system design. For example, the Yerkes-Dodson law generally implies that moderate amounts of operator *arousal* or *stress* result in the best performance [11, 12]. Although, in full generality, operator stress is an abstract construct that is dependent on the nature of and the relationship between a number of diverse stressors [13, 14], the relationship suggested by the Yerkes-Dodson law is often used by practitioners as the basis for moderating more controllable system metrics, such as the operator *task load*, under the assumption that these metrics are closely related to operator stress and can thus be used as a means of improving primary task performance.

In what follows, we develop a mixed-integer linear program (MILP) framework for constructing solutions to finite horizon task allocation and scheduling problems involving multiple human operators. Our method seeks to maximize the total achieved reward in the presence of processing time

uncertainties, while simultaneously mediating each operator's task load by penalizing schedules that are likely to cause task load levels to fall outside of a pre-specified regime. Note that the notion of *task load* is generally different from the notion of *workload*. Indeed, the two constructs can be distinguished as follows: task load is defined as "a measurement of human performance that broadly refers to the levels of difficulty an individual encounters when executing a task" [15]. Workload, on the other hand, "results from the demands a task imposes on the operator's limited resources and is determined by the relationship between resource supply and task demand" [16]. We choose to focus on task load, as opposed to other exogenous human factors issues, as a means of mediating and enhancing operator performance since it has well-established trends and links to performance that can feasibly be exploited by mission planners [17]. After establishing the base case, we illustrate how practical solution schemes can be constructed using our framework, which can potentially both boost performance and address computational issues. We note that the present work is an extension of our preliminary work [18], and is intended to explore the functionality of MILP frameworks in multiple operator scenarios.

Specifically, our contributions are as follows. We start by building an MILP framework for a multiple operator task-scheduling problem that can incorporate task load considerations using simple, trend-based models. We then illustrate how robustness to task processing time uncertainty can be added into this formulation through the use of scenarios. We also demonstrate how this strategy allows system designers flexibility to choose both the desired degree of robustness and the degree that task load is taken into account. Next, we illustrate how, in certain situations, general performance can be improved through the use of adaptive schemes, which employ both strategic re-planning and estimation. We first develop a basic receding horizon re-planning strategy for single operator scheduling, and subsequently expand our problem formulation to demonstrate a common scenario in which additional estimation is useful. We then discuss adaptations to these algorithms for use with multiple operators. Finally, we show through simulation how increased computation times due to such extensions may, in some scenarios, make some of the presented schemes intractable for practical use in their raw form. As an alternative, we propose a heuristic for task assignment in the multiple operator case, and show that we can achieve vast computational advantages at small,

if any, performance expense. Throughout our discussion, we also make a variety of suggestions and discuss issues that could arise in practical implementation of our proposed framework. Note that our problem setup and assumptions are primarily motivated by multi-UAV applications; however, there are a number of other situations in which this technology could be applied, such as manufacturing systems, healthcare applications, and maintenance roles.

## II. Multiple Operator Scheduling

### A. Scheduling Objective

Suppose that, at some time, there are $N \in \mathbb{N}$ heterogenous tasks stacked in a queue awaiting the attention of any one of $K \in \mathbb{N}$ operators. We assume that each task $T_i$, $i \in \{1, \ldots, N\}$, has an associated processing time $t_i \in \mathbb{R}_{>0}$, which defines how long the task will take to complete, an availability time $s_i \in \mathbb{R}_{>0}$, which defines the global time at which the task becomes available for processing, and an associated reward $r_i \in \mathbb{R}_{\geq 0}$, which is awarded upon successful task completion. Assume, for the moment, that $t_i$ is known *a priori* (we relax this assumption later). Assume that the parameters $t_i$, $s_i$, and $r_i$ are independent of which operator processes the task. Further, assume that all operators are aware of which tasks are available, and which tasks have already been processed at any time. We consider an "all or nothing" reward distribution scheme, in which an operator receives the full reward $r_i$ if the task is completed, and no reward otherwise, i.e., there is no pre-emption. Further, we assume that the reward for a task is only obtained if some operator completes the task within a pre-specified time horizon $T_H \in \mathbb{R}_{>0}$.

With this framework, we seek to find an optimal multi-operator schedule, i.e., ordered sequence of tasks for each operator, that maximizes the total reward accumulated across all operators. Note that, given any ordered sequence of tasks for any single operator, one can easily define appropriate starting times for each element of the sequence so that the availability time constraints $s_i$ are not violated. Indeed, the time at which the operator should start each task in the sequence can be taken as the maximum of the completion time of the previous task in the sequence, and the appropriate start time constraint. For completeness, we can formally characterize the problem statement as follows: define a *schedule* as a set of $K$ sequences $\{S_k \coloneqq \{T_{\sigma(1,k)}, T_{\sigma(2,k)}, \ldots, T_{\sigma(M_k,k)}\}\}_{k \in \{1,\ldots,K\}}$,

where $\sum_{i=1}^{K} M_k \leq N$ and $\sigma : \sqcup_{k=1}^{K}\{1,\ldots,M_k\} \to \{1,\ldots,N\}$ is some injective mapping (here, $\sqcup\cdot$ denotes the disjoint union operation). Let $\mathcal{S}$ denote the set of all possible schedules. We seek to find $S^* \in \mathcal{S}$, so that if each operator $k$ were to start the first task in the sequence at time $s_{\sigma(1,k)}$, and the $j$-th task in the sequence at time $\max\{C_{j-1,k}, s_{\sigma(j,k)}\}$ where $j \in \{2,\ldots,M\}$ and $C_{j-1,k}$ represents the time at which the $j-1$-st task in operator $k$'s sequence is completed , then the total accumulated reward across all operators is maximized.

### B. Task Load Constraints

Since we consider the problem primarily in the context of human-UAV collaborative teams, the effectiveness of a given schedule also hinges upon the cognitive states of the operators. Therefore, it is desirable to construct a schedule that allows each operator to maintain his/her cognitive state in a regime that is most amenable to high levels of performance. In particular, we focus our attention on mediating the relationship between the chosen schedule and the resulting operator task loads. Although there are many different ways of modeling task load, e.g., utilization ratio [19], multi-dimensional load space abstractions [20], among others, we model task load via an incremental, discrete process, which is driven by the task processing order and task processing times.

Our chosen model is based on the simple observation that, in most situations, when operators are executing tasks, their task load level increases, and when operators are idle, their task load decreases. In order to capture this simple dynamic evolution, to each task $T_i$, we associate a task load increment $\delta w_i$, which represents the amount that the operator's task load increases upon working on the task for time $t_i$. Further, we assume that task load will decrease by a certain amount when the operator is idle, proportional to idle time (discussed further in subsequent sections). With this model, we seek to solve the nominal scheduling problem under the additional constraint that, if possible, each operator's task load must remain within a desired regime $[\underline{w}, \overline{w}] \subset \mathbb{R}$ at any point $0 \leq t \leq T_H$. Although this model may be simplistic, it captures the essence of task load evolution during sequential task processing. Indeed, many widely accepted task load evolution models are deterministic processes that augment task load during busy times and degrade task load during idle times (e.g., [19]). Further, in the sequel, we will treat processing times as pre-

determined parameters for the optimization (even in the scenario-based formulation of Section II D). Therefore, if desired, the task load increment parameters can be systematically chosen to reflect more sophisticated dynamics. For example, if $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is a function that relates time to the amount that task load increments when the operator is engaging in a general task, then we can set $\delta w_i = f(t_i)$ for all $i \in \{1, \ldots, N\}$. A similar statement can be made regarding task load decrements during idle time. Note that while there are numerous task load (and cognitive workload) models in the literature, many rely on large amounts of data to be collected or extensive physiological data to calibrate and precisely model the operator load. We have chosen a simpler path that captures the main qualitative features of such activities. As part of the development of these types of approaches, it would be important to validate whether high level decision making problems (such as the one addressed in this paper) could support such simplified models, or would necessitate more complex, higher fidelity models. Currently, this is an open question that we leave as a topic of future work.

Despite its flexibility, the incremental task load model that we consider cannot account for dynamics in which increment or decrement magnitudes are dependent upon initial conditions, since the order in which tasks will be processed is unknown *a priori*. However, for a given operator, task load evolution is usually a subjective experience anyway, and thus fine level task load models may be ill-suited if they are derived from aggregate data. Therefore, in the construction of general schemes intended to be used with many different operators, simplistic dynamics may be preferable to finer level models. If, however, the model is meant to be tuned to a specific operator or group of operators, or more precise real-time data can be leveraged to accurately predict cognitive states (e.g., neurophysiological cues [21]), then alternative models may be preferable. In the latter case, the MILP framework presented herein may not be sufficient and other options should be explored.

### C. MILP Formulation

We now illustrate how the finite horizon scheduling problem can be formulated as a MILP. In the formulation, the primary decision variables are binary indicators $x_{i,j,k} \in \{0, 1\}$, which specify whether or not task $T_i$ should be executed in the $j$-th time slot of operator $k$'s output sequence (see Fig. 1). In accordance with Sections II A and II B, each task $T_i$ is fully specified by the 4-tuple

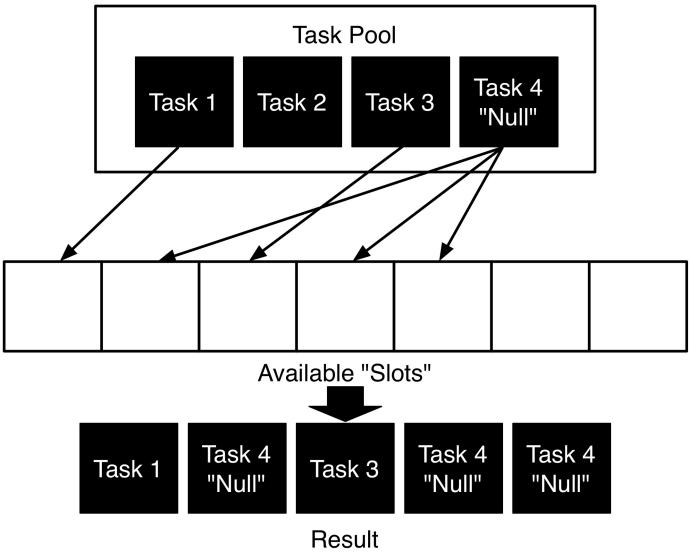



**Fig. 1 A diagram illustrating the basic MILP solution approach for an example with 4 tasks.**

$(t_i, s_i, r_i, \delta w_i)$. In addition to the tasks $\{T_1, \ldots, T_N\}$, we introduce one additional null-task $T_{N+1}$ to represent times during which the operator is idle. Specifically, we define the null task as $T_{N+1} \coloneqq (\zeta, 0, 0, -\delta w_{N+1})$, where $\delta w_{N+1} > 0$ is a constant and $\zeta \in \mathbb{R}_{\geq 0}$ is a parameter representing the length of the null-task. With the goal of capturing task load evolution within the MILP framework, we augment the set of tasks to be processed with the newly created null task, and re-define $\mathcal{T} \coloneqq \{T_1, \ldots, T_N, T_{N+1}\}$. The null-task is the only task that the operator is allowed to execute more than once. In other words, the output of our proposed method will be an *augmented schedule*, which is formally defined as a set of sequences $\{S_k \coloneqq \{T_{\sigma(1,k)}, T_{\sigma(2,k)}, \ldots, T_{\sigma(M_k,k)}\}\}$, where each index $M_k \in \mathbb{N}$ is upper bounded by the fixed parameter $M \in \mathbb{N}$, and $\sigma : \sqcup_{k=1}^{K}\{1, \ldots, M_k\} \to \{1, \ldots, N+1\}$ is some mapping such that the preimage of each singleton set $\{i\}$, where $i \in \{1, \ldots, N\}$, contains at most 1 element. This differs from the previous definition in that the null-task can appear more than once, and thus the variable $M$, which represents the maximum number of tasks that can appear in any one operator's sequence, is allowed to exceed the number of tasks $N$. In order to guarantee reasonable output augmented schedules, it is necessary to pick $M$ sufficiently large, i.e., to consider output sequences that can incorporate a sufficiently large number of terms. Setting $M = \lceil T_H/\zeta \rceil + N$ is sufficient for our purposes. For the remainder of this paper, when the distinction between a schedule and an augmented schedule is not of particular relevance, we will simply use

$$
\begin{aligned}
\underset{x_{i,j,k}\in\{0,1\}}{\text{maximize}} \quad & \left(\sum_i \sum_j \sum_k r_i x_{i,j,k}\right) - p_\beta \beta - p_\gamma \gamma & \\
\text{subject to} \quad & \sum_{i=1}^{N+1} x_{i,j,k} \leq 1, & \forall j,k \\
& \sum_{k=1}^{K} \sum_{j=1}^{M} x_{i,j,k} \leq 1, & \forall i \\
& \sum_{i=1}^{N+1} x_{i,j,k} - x_{i,j-1,k} \leq 0, & \forall j \neq 1,k \\
& \sum_{i=1}^{N+1} x_{i,j,k} s_i \leq B_{j,k}, & \forall j,k \\
& \sum_{i=1}^{N+1} x_{i,j,k} t_i = C_{j,k} - B_{j.k}, & \forall j,k \\
& C_{j,k} \leq T_H, & \forall j,k, \\
& 0 \leq B_{j,k} - C_{j-1,k} \leq \zeta, & \forall k; j \neq 1 \\
& 0 \leq B_{1,k} \leq \zeta, & \forall k \\
& W_{0,k} + \sum_{i=1}^{N+1} \delta w_1 x_{i,1,k} = W_{1,k}, & \forall k \\
& W_{j-1,k} + \sum_{i=1}^{N+1} \delta w_i x_{i,j,k} = W_{j,k}, & \forall k; j \neq 1 \\
& W_{j,k} - \overline{w} \leq \beta, & \forall j,k \\
& \underline{w} - W_{j,k} \leq \gamma, & \forall j,k \\
& \beta, \gamma \geq 0. &
\end{aligned}
\tag{1}
$$

the term "schedule," and reserve the qualifier "augmented" only for cases where the distinction is consequential to the discussion.

With this structure, we can incorporate task load into an MILP as an explicit variable that satisfies an appropriate set of constraints. As such, the scheduling problem reduces to finding the optimal augmented schedule based on the set $\mathcal{T}$. Consequently, the MILP takes the form (1). We note that this formulation is a natural extension of that in our previous work [18], which considers an analogous problem for a single operator scenario. Here, we assume $i \in \{1,\ldots,N\}$, $j \in \{1,\ldots,M\}$, and $k \in \{1,\ldots,K\}$ unless otherwise stated. In (1), the first set of constraints ensure that the solution to the MILP corresponds to a feasible solution to the original scheduling problem. That is, these constraints specify that each time slot, with respect to each operator's sequence order, can

contain at most one task. Similarly, the second set of constraints guarantees that each task can only be assigned to at most a single location in the sequence, with the exception of the null task, which can be performed multiple times (note that by convention, $i \leq N$ and thus the constraint does not include the null-task). The third set of constraints guarantee that each operator's task "slots" are filled successively. That is, if some operator's task slot is filled with a task, then all of the previous slots must be filled as well. This ensures that the output solutions correspond to valid augmented schedules, according to the rigorous definition given. The fourth through the eighth sets of constraints deal with issues relating to task availability, start times, and completion times. Here, $B_{j,k}$ and $C_{j,k}$ denote the start time and the completion time of the $j$-th task in operator $k$'s sequence, respectively. Accordingly, the fourth set of constraints guarantees that no task is started before the specified availability time $s_i$. The fifth set of constraints guarantees that the start times and completion times are defined in a reasonable way. The sixth constraint specifies that rewards are only attained for those tasks that are completed prior to the time horizon $T_H$. The contributions of the seventh and eighth sets of constraints are two-fold: First, the lower bounds specify that, in any individual operator's sequence, no task can begin before the previous task ends. The upper bounds arise from the fact that we have discretized operator idle time into discrete sets of length $\zeta$. These constraints limit "gaps" that may be present in the output schedule (see Remark 1).

The remaining constraints deal with moderating task load. With our incremental definition, it is necessary to ensure that operator $k$'s task load level, $W_{j,k}$, after the $j$-th task is defined precisely as the task load $W_{j-1,k}$ after execution of the previous task plus the increment defined in the task definition. This is captured by the ninth and tenth sets of constraints in (1). Here, $W_{0,k}$ denotes operator $k$'s initial task load. Notice that, since we have included the null-task in the pool of available tasks, this set of constraints also defines the appropriate change in task load due to idle time. The eleventh, twelfth, and thirteenth sets of constraints state that the task load levels need to remain within the pre-specified bounds, buffered by the decision variables $\beta$ and $\gamma$. These buffer variables $\beta$ and $\gamma$ enter into the objective function as linear penalties for violating the task load constraint, proportional to the parameters $p_\beta, p_\gamma > 0$. Enforcing the task load bounds in this manner, rather than as a strict constraint, is beneficial for a variety of reasons. First, exact bounds

$\underline{w}, \overline{w}$ are usually not known beforehand, and thus enforcement of a strict bound may be ill-advised. Second, the variables $p_\beta$ and $p_\gamma$ provide the system designer with a means to tune the degree of enforcement of task load bounds. Indeed, higher values of $p_\beta$ and $p_\gamma$ lead to higher penalties in the objective function for bound violations. Finally, the enforcement of task load penalties as a soft constraint ensures feasibility of at least 1 non-degenerate solution, assuming that the time horizon $T_H$ is sufficiently long. This leads to the following observation.

**Lemma 1** (Feasibility)**.** *The optimization* (1) *is feasible. If, in addition, there exists* $i \in \{1, \ldots, N\}$ *such that* $t_i + s_i < T_H$*, then there exists a non-degenerate feasible point, i.e., a point in which there is at least* 1 *task in the output augmented schedule associated with the point in question.*

*Proof.* The zero vector is feasible, implying nominal feasibility. If there exists $i \in \{1, \ldots, N\}$ such that $t_i + s_i < T_H$, then we can construct a decision vector $\hat{x}$ such that $x_{i,1,1} = 1$, all other binary decision variables are equal to zero, $B_{1,1} = s_i$, $C_{1,1} = s_i + t_i$, and $\gamma = \beta = C$, where $C > \max\{|W_{0,1} + T_H \delta w_i|, |W_{0,k} - T_H \delta w_{N+1}|\}$. Simple substitution verifies that the vector $\hat{x}$ is feasible. □

Most formulations of the optimal scheduling problem are known to be NP-hard. The MILP (1) presents a similar set of difficulties. Despite difficulties in finding global optima for all problem types, effective heuristics and reasonable methods exist for finding high quality solutions to MILPs [22]. Such methods include rounding schemes, branch and bound search strategies, genetic algorithms, simulated annealing schemes, and many others [23, 24]. Many heuristics are implemented in an efficient way and are included in a variety of software packages, including the Matlab optimization toolbox [25] and the cvx software package [26, 27]. Therefore, for applications where strict global optima are not of primary concern, the MILP formulation (1) may provide a viable solution.

**Remark 1** (Null-Task Granularity)**.** *Recall that* (1) *relies on the creation of a discrete "null-task" with length* $\zeta$*. Once a solution to the MILP has been found and an augmented schedule is extracted, tasks are executed by each operator in a sequential manner. Due to task availability constraints, however, it may not be possible for the operator to start a new task immediately after the previous task has been completed. Further, the solver may also introduce artificial delays between successive null-tasks. Therefore, if the time-profile of the operator's task execution is mapped over the horizon*

*$T_H$, there may be "gaps", i.e., times in which no task (even a null-task) is being executed. Task load effects due to these gaps are not explicitly taken into account in* (1)*. However, a result of the seventh and eighth sets of constraints is that the length of any possible gaps shrink to* 0 *as* $\zeta \to 0$*. Therefore, if* $\zeta$ *is taken small enough, then task load effects due to unaccounted gaps become negligible. Of course, shrinking* $\zeta$ *increases computational complexity, and is thus a tradeoff that must be addressed by system designers.*

### D. Incorporating uncertainty

The major theoretical downfall of the scheduling formulation of Section II C is that it does not directly incorporate uncertainty in system parameters. As mentioned, uncertainty in behavior is inherent to mixed human-machine teaming applications, and careful considerations should be taken to design a system that takes this issue into account. In response, we focus our attention in this section on designing systems that are robust to uncertainty in task processing times, which is usually a significant source of uncertainty in persistent task analysis missions. As such, we assume that, for each task $T_i$, the processing time $t_i$ is a random variable, which is distributed according to a probability density function $f_i$. We assume for now that each distribution $f_i$ is known with complete certainty *a priori*. For simplicity, we assume that the distributions $f_i$ are operator independent; however, we note that, if desired, operator-dependent processing time distributions can be added to our formulation through straightforward extension (see Remark 5). Generally, the choice of appropriate function $f_i$ is dependent upon the nature of the task being processed. However, in some circumstances, standard distributions, such as log-normal distributions, have been shown to be effective in the context of collaborative human-UAV missions [28, 29].

We adopt a scenario-based approach to incorporating uncertainty into the system. According to this approach, the processing time distributions of each task are sampled to generate a set of *scenarios*, or possible task processing times. These scenarios are then used to find an augmented schedule, which remains feasible regardless of the scenario that is chosen to represent the true task processing times. If the underlying distributions are accurate, then using large numbers of scenarios to generate a task processing schedule enables the result to be robust to particular realizations of the

processing time variables in actuality. That is, by following the schedule obtained, a wide variety of task processing times will still produce rewards that are above the predicted lower bound. Note that the optimal choice for the number of scenarios will generally depend on application goals. A thorough study of the optimal number of scenarios with respect to a given performance metric is an open research problem that we do not consider here. For our purposes, we choose values for this parameter that allow us to illustrate qualitative effects on the resultant solution.

More formally, suppose that for each task $T_i$, we generate a set of $Q \in \mathbb{N}$ possible processing times, $(t_i^1, t_i^2, \ldots, t_i^Q)$, where $t_i^m \sim f_i$ for all $m$. For each $m \in \{1, \ldots, Q\}$, the set $\{t_1^m, \ldots, t_N^m, \zeta\}$ defines a *scenario*, since it represents a possible realization of task processing times (note that the null task is included). Given the set of $Q$ scenarios, we expand the MILP (1) through the additional requirement that any constraint must be satisfied for *all* of the generated scenarios. The resulting MILP is expressed in (2). In (2), it is assumed that $k \in \{1, \ldots, K\}$, $i \in \{1, \ldots, N\}$, $j \in \{1, \ldots, M\}$, and $m \in \{1, \ldots, Q\}$ unless otherwise stated. Note that (2) is largely the same as (1); however, there are a few key differences. First, note that, for each $m$, there are unique sets of decision variables $\{B_{j,k}^m\}, \{C_{j,k}^m\}$, and $\{W_{j,k}^m\}$, where $j \in \{1, \ldots, M\}$ and $k \in \{1, \ldots, K\}$, yet there is only one set of indicator variables $\{x_{i,j,k}\}$ which serves all scenarios. Thus, starting times, ending times, and task load values are time or scenario-dependent, while the resulting schedule is not dependent on these parameters. This allows for the unambiguous extraction of an augmented schedule from the solution to the optimization. The result will then (ideally) be robust to uncertainty in processing times. Second, an additional decision variable $\alpha$ has been added to bound the amount that the task completion times can exceed the time horizon $T_H$. This variable $\alpha$ is then factored into the objective function by means of a linear penalty. As with task load, this "soft" enforcement of the time horizon constraint is advantageous because it provides system designers with an additional "tuning" parameter to control the degree of robustness in the system design. Indeed, by increasing the value of $p_\alpha$, more penalty is incurred for generating schedules whose completion times are likely to exceed $T_H$. These additional parameters also serve to prevent the optimization from returning degenerate solutions in the case of highly skewed processing time distributions. This discussion readily leads to the following lemma, whose proof is omitted.

$$
\begin{aligned}
\underset{x_{i,j,k}\in\{0,1\}}{\text{maximize}} \quad & \left(\sum_i \sum_j \sum_k r_i x_{i,j,k}\right) - p_\alpha \alpha - p_\beta \beta - p_\gamma \gamma & \\
\text{subject to} \quad & \sum_{i=1}^{N+1} x_{i,j,k} \leq 1, & \forall j,k \\
& \sum_{k=1}^{K} \sum_{j=1}^{M} x_{i,j,k} \leq 1, & \forall i \\
& \sum_{i=1}^{N+1} x_{i,j,k} - x_{i,j-1,k} \leq 0, & \forall j \neq 1,k \\
& \sum_{i=1}^{N+1} x_{i,j,k} s_i \leq B_{j,k}^m, & \forall j,k,m \\
& \sum_{i=1}^{N+1} x_{i,j,k} t_i^m = C_{j,k}^m - B_{j.k}^m, & \forall j,k,m \\
& C_{j,k}^m - T_H \leq \alpha, & \forall j,k,m \\
& 0 \leq B_{j,k}^m - C_{j-1,k}^m \leq \zeta, & \forall k,m; j \neq 1 \\
& 0 \leq B_{1,k}^m \leq \zeta, & \forall k,m \\
& W_{0,k}^m + \sum_{i=1}^{N+1} \delta w_1 x_{i,1,k} = W_{1,k}^m, & \forall k,m \\
& W_{j-1,k}^m + \sum_{i=1}^{N+1} \delta w_i x_{i,j,k} = W_{j,k}^m, & \forall k,m; j \neq 1 \\
& W_{j,k}^m - \overline{w} \leq \beta, & \forall j,k,m \\
& \underline{w} - W_{j,k}^m \leq \gamma, & \forall j,k,m \\
& \alpha, \beta, \gamma \geq 0. &
\end{aligned}
\tag{2}
$$

**Lemma 2.** *There always exists at least* 1 *non-degenerate solution to the optimization* (2)*.*

**Remark 2** (Robust Optimization)**.** *Many schemes exist for solving optimization problems in the presence of uncertainty. In particular, the area of robust optimization provides tools for finding solutions that perform well in the worst case, given that the underlying uncertainty sets can be accurately bounded [10]. Scenario-based approaches to handling uncertainty can, in some sense, be viewed as a "naive" approach to robust optimization since they rely on an expansion of the constraint set through Monte Carlo sampling. Despite this, scenario-based approaches lend themselves well to a variety of applications due to their simplicity and effectiveness. We chose to adopt such an approach for the following reasons: First, scenario-based optimization is simple and intuitive, making it an attractive option for practitioners and theoreticians alike. Second, most other approaches to robust*

*optimization rely on the presence of bounded uncertainties, or require approximations that enforce artificial bounds. The scenario-based approach does not require such bounds. Third, scenario-based approaches allow system designers to easily tune the problem to fit with a desired degree of robustness. Finally, given a "regularly" shaped distribution of uncertain parameters, scenario-based approaches usually provide reasonable performance with only a modest number of samples.*

**Remark 3** (Scenario Generation)**.** *Scenarios are generated by sampling the functions $f_i$. Naive Monte-Carlo sampling usually produces samples that most accurately reflect the underlying distribution in the limit as the number of samples tends to infinity. However, scenarios can be generated by virtually any reasonable sampling method and, in certain situations, design goals may be better served by these alternative means. For example, processing time distributions may have semi-infinite support, and thus Monte-Carlo sampling can produce a few scenarios with excessively long processing time durations in comparison to the horizon length. In most cases, excessively long durations will lead to degenerate, or excessively conservative solutions, which are not usually helpful. Therefore sampling techniques that restrict sample processing times may be beneficial.*

### E. Numerical Examples

Fig. 2 shows a summary of scheduling solutions for an abstracted and simplified mission with 10 tasks, 2 operators, a total horizon length of 30 (dimensionless units), and log-normally distributed task processing times. The solution was calculated using the formulation (2). For this simulation, we chose $p_\alpha = 10$, $p_\beta = p_\gamma = 15$, $\underline{w} = 0.3$, $\overline{w} = 0.7$, and $W_{0,1} = W_{0,2} = 0.5$. In Fig. 2(a) and Fig. 2(c), the problem setup is portrayed using the bars in the top portion of the figures. For each task $T_i$, the appropriate bar starts at the task availability time $s_i$, and extends for a length corresponding to the expected task execution time. The number inside the bar represents the reward obtained for successful completion of the task. The gray bars in the lower portion of the plot represent a simulated instance of the operator task execution for each scenario condition, based on the solution to (2). The start time and length of the bar represent the time that the task was started and the task duration respectively, while the number inside the bar represents the task index, with a letter "R" indicating a null-task (or resting task). Specifically, these simulated

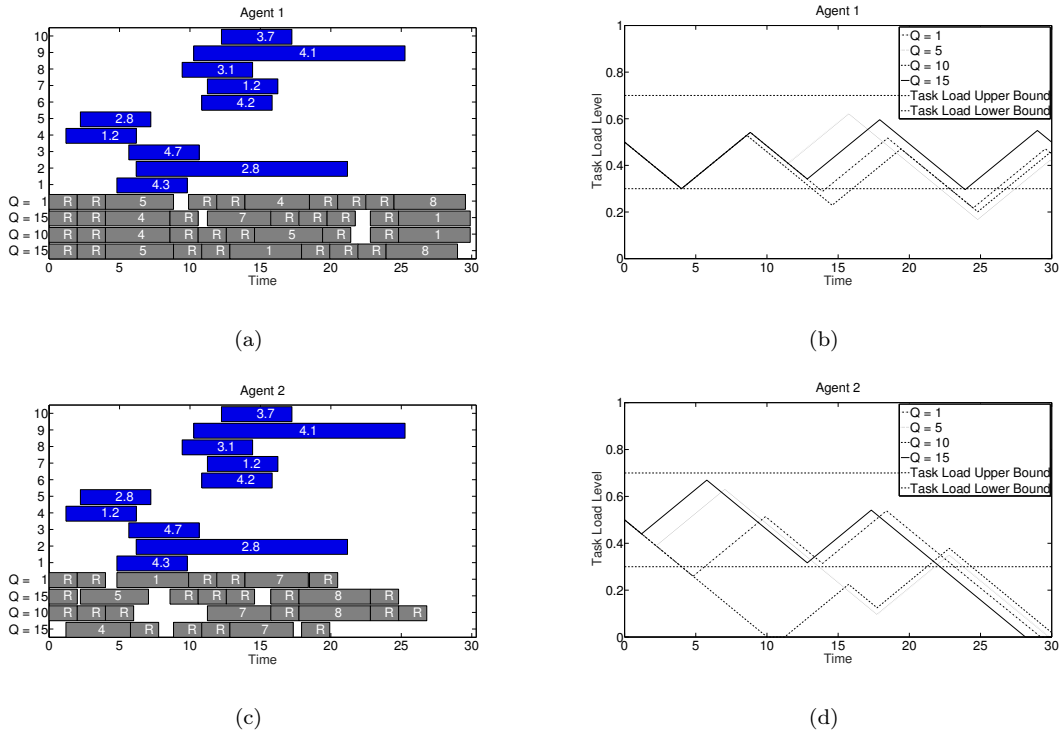



**Fig. 2 An example solution produced using** (2)**.**

schedules were calculated as follows. First, "actual" task execution times, i.e., realizations of each task processing time, were randomly sampled from the appropriate distribution. Then, for each scenario condition, scenarios were generated using naive Monte-Carlo sampling (see Remark 3), and the optimization (2) was solved using Matlab's `Intlinprog` function. Using the resultant augmented schedule, task execution was simulated by assuming that each operator processes each task in the respective sequence specified by the optimization solution and that each task has a duration according to the "actual" processing time. Each task was started at the earliest possible time (the maximum of the previous task completion time and the availability constraint for the task), and tasks were only included if their actual completion time was less than $T_H$. Note that in the actual schedule, all task executions satisfy the availability constraints, as expected, but their actual duration differs from the expected duration due to uncertainty in processing times.

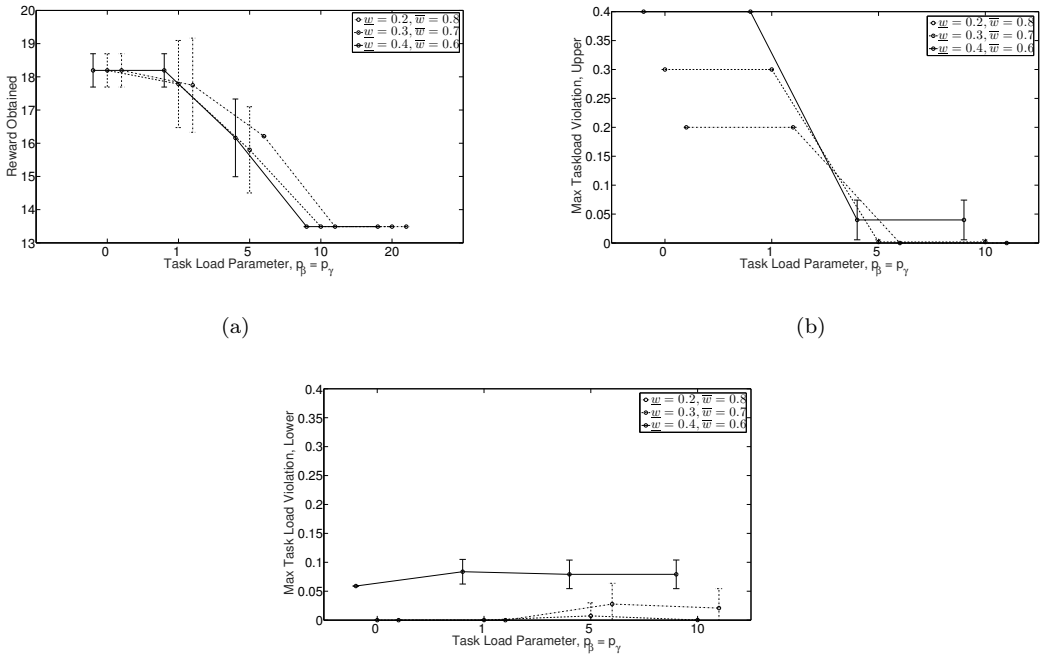
Fig. 2(b) and Fig. 2(d) show the evolution of task load corresponding to the simulated mission in Fig. 2(a) and Fig. 2(c). In the simulation, a simple linear task load model is considered: if $t \in \mathbb{R}_{\geq 0}$,

we assume operator $k$'s task load $W(t)$ evolves according to the dynamics

$$\frac{\mathrm{d}W}{\mathrm{d}t}(t) = \begin{cases} 0.05 & \text{, if executing a non-null task at time } t \text{ and } W(t) \neq 1, \\ -0.05 & \text{, if either executing a null task or no task at time } t \text{ and } W(t) \neq 0, \text{ and} \\ 0 & \text{, otherwise.} \end{cases}$$

We have chosen to constrain task load values to lie between 0 and 1 both for modeling puposes, and because this is a normalization that arises naturally in common task load models. For example, if task load were correlated with utilization, i.e., the fraction of recent time that the operator has been active, then this normalization is necessary [30]. In *a priori* planning, this cap is not taken into account by the optimization, and thus, for planning, it is assumed that $\delta w_i^m \coloneqq \pm 0.05 t_i^m$, where the sign is dependent upon whether the task under consideration is a null-task. Note here, that as the number of scenarios increases, the task load does not exceed the specified bounds.

Fig. 3 illustrates the effects of altering the task load bounds on achieved performance for a single-operator scheduling mission. The plot was generated as follows. First, an underlying task set was generated, the parameters $T_H = 30$, $W_{0,1} = 0.5$, and $p_\alpha = 10$ were initialized, and a set of 10 scenarios were generated. Then, 10 simulations runs were conducted, where in each run, 1) "actual" task processing times were sampled from the appropriate distribution, 2) an augmented schedule was created using a solution to (2) for each set of $\underline{w}, \overline{w}, p_\beta$, and $p_\gamma$ values (for the "no task load" condition, simply set these parameters to 0, i.e., no penalty is assessed for exceeding task load bounds, under the same "capped" linear task load model as before, 3) the task execution process was simulated using the resulting schedule and "actual" task times in the same manner that was used to generate the gray bars in Fig. 2(a) and Fig. 2(c), and 4) the achieved nominal reward was recorded, i.e., the sum of the $r_i$'s for tasks that could be executed within time $T_H$. Fig. 3(a) shows the mean and standard deviation of the rewards obtained over the simulation runs (the absence of error bars indicates no variation across runs). Fig.s 3(b) and 3(c) show the maximum amounts that the task load exceeded the upper task load bound and the maximum amount that the task load violated the lower task load bound during task execution, respectively (not taking into account violations that occur after all tasks in an operator's schedule have been executed). Note that, as expected, the obtained reward decreases as $p_\beta$ and $p_\gamma$ are increased and as the allowable task load




(a) (b)

(c)

**Fig. 3 An illustration of the effects of altering the parameters $\underline{w}$, $\overline{w}$, $p_\beta$, and $p_\gamma$.**

bounds widen, while the amount of task load violation decreases.

## III. Adaptive Scheduling Scheme

The use of *a priori*, scenario-based robust scheduling strategies can generate reliable lower bounds on system performance. That is, using a reasonable number of scenarios (and, of course, assuming that the chosen processing time distributions are accurate), the MILP (2) will produce a theoretical lower reward bound that will likely hold true in actuality. In missions where accurate performance guarantees are crucial to system success, such bounds may be sufficient. However, theoretical bounds produced by scenario-based robust optimizations may be very conservative, particularly when the probability distributions of uncertain parameters are highly skewed or have high variance. For example, if visual search times are modeled via log-normal distribution, then robust scheduling strategies will be most likely driven by search times occurring in the tail, which are unlikely to occur in actuality. In light of this fact, it is clear that naively following a robust schedule that is calculated using *a priori* information does not take advantage of the full knowledge at the designer's disposal during the mission, and thus may lead to poor solutions with respect to actual

realizations of uncertain parameters.

In this section, we explore adaptive methods for improving performance while still maintaining the desired robustness properties of solutions. We start by developing strategies for the single-operator case (i.e., the case where $K = 1$), and subsequently show how they can be extended for use with multiple operators.

### A. Single Operator Receding-Horizon Scheduling

Assume that a single operator is charged with processing tasks generated by the UAVs. In the present problem setup, tasks are processed sequentially by the operator. Performance under a robust scheduling strategy can potentially be improved by re-planning the mission in an online fashion as realizations of task processing times become known. This observation naturally leads to a receding-horizon robust scheduling scheme, which calculates new, robust schedules after each task is processed. Such a scheme is described by the following pseudocode:

1. collect the tasks to be processed in a set $\mathcal{T}$,
2. formulate and solve the scenario-based robust scheduling problem as Section II D, with respect to the set $\mathcal{T}$ and a total horizon length $T_H$,
3. execute the first task in the resultant augmented schedule, and observe the processing time $t$,
4. if the executed task is a not a null-task: remove it from the set $\mathcal{T}$,
5. subtract $t$ from all remaining task availability constraints, redefine $T_H = T_H - t$, and
6. repeat steps 2-5 until all tasks have been executed or until the remaining horizon $T_H \leq 0$.

The advantage of this approach is that it takes advantage of the manner in which uncertainty arises in the sequential task analysis mission. Indeed, in these missions, uncertainty is generally reduced as time progresses. Accordingly, the receding horizon scheme re-plans each time the uncertainty set is incrementally decreased, which, in the robust planning case, will generally lead to better performance. The obvious drawback to the presented receding horizon strategy is that it requires an MILP to be solved after each task is executed; thus, the total computational complexity is

significantly increased. Therefore, there is a tradeoff between solution quality and computation time that will need to be assessed on a case by case basis.

**Remark 4** (Computation time)**.** *Even though the receding-horizon robust scheduling schemes may be computationally intensive in their raw form, there are many simple steps that can be taken to adjust the schemes and fit them into a variety of computational frameworks. The simplest amendment is to reduce the number of scenarios used, which reduces complexity at the expense of robustness. A slightly more sophisticated strategy involves an adaptive approach that only re-plans when a certain criterion is met, as opposed to re-planning after every task execution. For example, one could choose to re-plan only if the observed time on the executed task is significantly different from the worst-case processing time predicted by the generated scenarios. For the sake of brevity, a thorough treatment of these types of amendments is not included here, and is left as a topic of future work.*

### B. Single Operator Receding-Horizon Robust Scheduling with Estimation

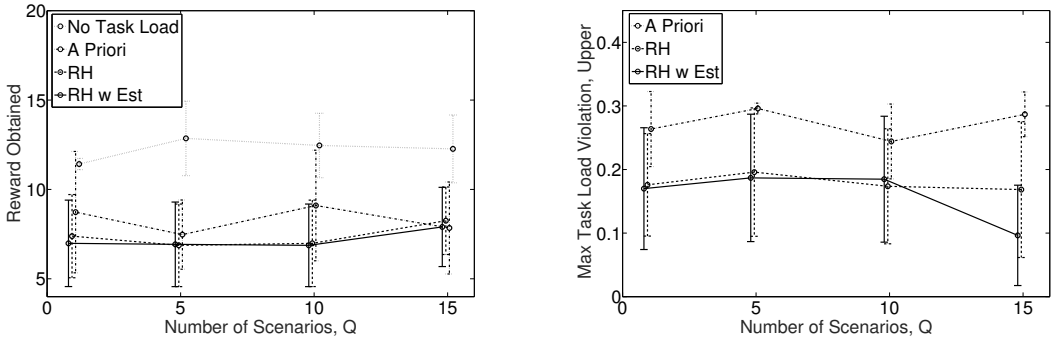
Up until this point, it has been assumed that the probability distribution $f_i$ of task processing times for each task has been known exactly. Indeed, samples are drawn assuming certain distributions in order to generate the scenarios that are used in the optimization (2). In many cases, however, the processing time distributions themselves may not be known exactly, and thus it may be beneficial to estimate them during the course of task execution. Of course, depending on problem assumptions, such estimation may not be helpful, e.g., if all tasks are assumed to have completely independent distributions $f_i$. However, given appropriate additional problem structure, online estimation of uncertain distributions can potentially further boost performance when used in conjunction with the receding-horizon MILP approach of Section III A. To illustrate, we develop a common scenario that can benefit from such estimation here.

Assume once again the presence of a single operator. Building on the formulation of the previous sections, suppose each task $T_i$ is generated by one of $P \in \mathbb{N}$ sources, e.g., different regions of interest, and each such task can be of $T \in \mathbb{N}$ possible types, e.g., easy, medium, or hard. Assuming independent sampling, suppose further that associated to each source $p \in \{1, \dots, P\}$, there exists an unknown, static probability mass function $g_p : \{1, \dots, T\} \to [0, 1]$ which captures the likelihood

that a task originating from source $p$ is of type $\ell$. That is, the probability of any single task that is generated by source $p$ being of type $\ell$ is $g_p(\ell)$. Finally, to each type $\ell$, we associate an unknown, static probability density function $f_\ell : \mathbb{R}_{\geq 0} \to \mathbb{R}$, which captures the distribution of possible operator processing times. Note that the processing time distributions $f_\ell$ are region independent. To summarize, with these additions, each task $T_i$ now consists of a 6-tuple $T_i = (t_i, s_i, r_i, \delta w_i, \phi_i, \tau_i)$, where $s_i$, $r_i$, and $\delta w_i$ are defined as before, $\phi_i \in \{1, \ldots, P\}$ represents the source which generated the task (known to the operator), $\tau_i \sim g_{\phi_i}$ is the task type (unknown to the operator), and $t_i \sim f_{\tau_i}$ is the time that it will take to process the task (unknown to the operator).

As mentioned, the underlying distributions associated with both sources and types are unknown to the operator. However, in this new setup, there are commonalities among processing time distributions that can be exploited through estimation. Therefore, we implement estimation in conjunction with the receding horizon approach as follows. For each source $p$ and each type $\ell$, let $\hat{g}_p$ and $\hat{f}_\ell$ denote estimates of the respective functions. The basic idea behind this adaptive scheduling approach is to incrementally update the estimates $\hat{g}_p$ and $\hat{f}_\ell$ as new information becomes available, i.e., as the operator processes tasks. The updated estimates are then used to perform subsequent scheduling operations. Loosely, the algorithm evolves as:

1. collect the tasks to be processed in a set $\mathcal{T}$,
2. generate new scenarios according to the estimates $\{\hat{g}_p\}_{p\in\{1,\ldots,P\}}$ and $\{\hat{f}_\ell\}_{\ell\in\{1,\ldots,T\}}$,
3. formulate and solve the scheduling problem (2),
4. execute the first task $T_i$ in the resulting sequence, and observe $\tau_i$ and $t_i$,
5. if task $T_i$ is not a null task,
   (a) update the estimates $\hat{g}_{\phi_i}$ and $\hat{f}_{\tau_i}$,
   (b) remove the executed task from the set $\mathcal{T}$,
6. subtract $t_i$ from the remaining task availability constraints, redefine $T_H = T_H - t_i$, and
7. repeat steps 2-6 until all tasks have been executed or until the remaining horizon $T_H \leq 0$.




(a) (b)

**Fig. 4 A performance comparison between different solution methodologies.**

Step 2 in the process outlined above requires explaination. Indeed, the operator does not know the task type of any unprocessed task with complete certainty, and thus it is necessary to make a decision about which distribution should be sampled for the purpose of generating scenarios. This choice can be made in many different ways. For example, a logical choice is a "maximum likelihood" method, where each timing parameter $t_i^m$ for the $m$-th scenario is selected by first finding $\tau_i^* \coloneqq \arg\max_\ell \hat{g}_{\phi_i}(\ell)$ and subsequently sampling the distribution $f_{\tau_i^*}$. That is, each scenario is generated by sampling from the processing time distribution corresponding to the most likely type for task $T_i$, according to $\hat{g}_{\phi_i}$. We exploit the "maximum likelihood" sampling process in our remaining simulations. Once the sampling method has been established, it remains to choose a process for updating the distributions $\hat{g}_p$ and $\hat{f}_\ell$. The appropriate update method will generally be governed by the problem assumptions. To illustrate the functionality of the receding-horizon robust scheduling scheme with estimation, we assume that each element in the set $\{f_\ell\}_{\ell\in\{1,\ldots,T\}} \coloneqq (\mu_\ell, \sigma_\ell)$ is log-normally distributed, where $\mu_\ell, \sigma_\ell$ are the standard log-normal distribution parameters. We also assume that some previous information is available regarding each distribution $g_p$ and $f_\ell$ in the form of a set of previous samples, accumulated prior to the current scheduling mission. With this information, upon completion of task $T_i$, the appropriate distributions $\hat{g}_{\phi_i}$ and $\hat{f}_{\tau_i}$ are updated using standard maximum likelihood estimation.

Fig. 4 presents a comparison between the various solution methods discussed thus far for a sample mission involving 10 tasks, 2 possible task sources, and 3 possible task types. Here, it is

assumed that there is uncertainty in task processing times, as well as in the task distributions $g_p$ and the distributions $f_\ell$. Each distribution $g_p$ was generated randomly, while the distributions $f_\ell$ were generated by creating a set of log-normal distributions, each with identical $\sigma$ parameters, and whose medians ($e^\mu$) were spaced equally across the interval $[0, 0.5T_H]$. It is assumed that 10 prior samples from each source distribution $g_p$ are available *a priori*, and 5 prior samples are available from each distribution $f_\ell$. The prior samples were generated from sampling the appropriate "actual" distributions. The estimates $\hat{g}_p$ and $\hat{f}_\ell$ were then generated using standard maximum likelihood estimation. For the methods with the estimation step, these distribution estimates were re-evaluated each time the operator executed a task. In all cases, scenarios were generated using the "maximum likelihood" scheme. Finally, we chose $p_\alpha = 10$, $p_\beta = p_\gamma = 15$, $W_0 = 0.5$, $\underline{w} = 0.3$, $\overline{w} - 0.7$, and $T_H = 30$. Fig. 4(a) shows the nominal rewards obtained, i.e., the sum of the the rewards $r_i$ associated with tasks that were executed, averaged over 10 simulated task execution processes. The same setup and prior information was used in each run, but each run had different realizations of the processing times, sampled from the underlying distributions. Identical scenarios were used across experimental conditions (i.e., solution methods considered). As expected, the no task load consideration condition resulted in the highest achieved rewards for all cases, and the rewards showed a slight downward trend as the number of scenarios increases, since higher numbers of scenarios provide increased robustness at the expense of lower expected rewards. Also note the large variances due to the high amounts of uncertainty in the underlying problem. Fig. 4(b) shows the maximum amount that the upper task load bound was violated during the course of task execution (the lower bound was almost never violated in any of the conditions, so we have chosen to omit this case). Notice that, qualitatively, the receding horizon scheme with estimation resulted in lower task load violations than the other methods, while still maintaining a similar level of nominal reward, particularly as the number of scenarios increased. Although the obtained benefit is quantitatively inconclusive in this example, we hypothesize that the adaptive scheme with estimation would likely produce a more pronounced benefit in situations with larger numbers of tasks. Rigorous analysis of this hypothesis and other effects due to the underlying problem structure on the utility of adaptive schemes is another interesting avenue of future research.

### C. Multiple-Operator Receding-Horizon Schemes

On the surface, it may seem straightforward to extend the receding-horizon schemes of the previous two sections for use with multiple operators. However, upon closer examination, certain aspects of the algorithms presented in the previous sections become unclear in the presence of multiple operators that process tasks simultaneously. For example, in calculating schedules in the receding horizon framework for a single operator (as in Section III A), it is abundantly clear when re-planning operations are appropriate, e.g., when the operator finishes processing a task. However, with multiple operators, each operator will finish his/her assigned tasks at different times, and therefore a decision must be made as to when it is appropriate re-plan. We explore these and other issues relating to multiple operator adaptive schemes in this section.

Assume the presence of $K > 1$ operators and the task structure of Section III B. As before, we assume that all task-related quantities are operator independent. That is, each task $T_i$ is defined as before, with the parameters $t_i, s_i, r_i, \delta w_i, \phi_i, \tau_i$ being independent of which operator processes the task. Further, the distributions $g_p$ associated with sources and the distributions $f_\ell$ associated with each task type are operator independent. Under these assumptions, the formulation (2) is appropriate for *a priori* planning of schedules. However, to develop a receding horizon scheme for multiple operators (analogous to Section III A) some additional work is necessary. In particular, different operators will process and finish tasks at different times, and therefore an appropriate re-plan strategy must be formulated. In order to keep a close connection to the single cycle case, we develop a strategy in which we re-plan each time a non-null task is completed. This is explained by the following pseudocode.

1. collect the tasks to be processed in a set $\mathcal{T}$,
2. generate new scenarios,
3. formulate and solve the scheduling problem (2),
4. instruct all operators to begin executing their respective schedules,
5. if some operator finishes processing a task $T_i$ after time $t$ has elapsed, subtract time $t$ from the horizon length $T_H$ and from the remaining task availability constraints,

6. if task $T_i$ is non-null, remove it from the task list $T$,

7. formulate and solve a multiple operator scheduling problem, constraining tasks that are already in progress to be completed first by the appropriate operator,

8. instruct operators to begin executing their new schedules (if an operator has a task already in progress, the completion of this task will be the first task in their new schedule), and

9. repeat steps $4-8$ until all tasks are executed or until the remaining horizon length $T_H \leq 0$.

Notice that step 7 of the above algorithm requires the ability to plan a new multiple operator schedule while simultaneously constraining the tasks that are currently in progress to be completed first. This will require an additional constraint to be added to (2). For example, if task $T_i$ is in progress by operator $k$ at the time of re-plan, then the additional constraint to be added would take the form $x_{i,1,k} = 1$. Indeed, setting $x_{i,1,k} = 1$ indicates that the first task in operator $k$'s new task sequence is task $T_i$. Recall, however, that at the time of re-plan, the actual processing time of any task in progress is still unknown. Therefore, in order for the MILP to accurately reflect the current state of the task execution process, prior to solving the MILP (2) (with the additional constraint just mentioned), it is also necessary to subtract off the time that each task has been in progress from the generated scenarios.

In summary, for the previous algorithm, step 7 will take the form:

1. Generate new scenarios for the remaining tasks in $\mathcal{T}$

2. For each task in progress $T_i$,

   (a) Find the difference $\bar{t}$ between the current time and the time task $T_i$ was started

   (b) For all $m$, set $t_i^m = \max\{t_i^m - \bar{t}, 0\}$.

   (c) Find the agent $k$ that is working on task $i$. Add an additional constraint $x_{i,1,k} = 1$ to (2).

3. Formulate and solve (2), including the newly added constraints.

If desired, an estimation step can be added in a way analogous to that presented in Section III B. This addition is straightforward, and thus we omit further discussion of this case.

**Remark 5** (Generality)**.** *Although we have made a variety of assumptions in formulating* (2)*, additional degrees of generality can be achieved by relaxing some of these assumptions and amending the MILP in a natural way. For example, task processing times can be made operator dependent by generating scenarios for each operator, and introducing an additional index k into the task processing time. With this setup, the variables* $t_i^m$ *would become* $t_{i,k}^m$ *to represent the time required for operator k to process task i according to the m-th scenario. Other generalizations can be made in a similar fashion. These additional complexities are, of course, at the expense of additional computation.*

**Remark 6** (Re-plan Schemes)**.** *In the presented receding horizon scheme for multiple operators, we have chosen to re-plan each time a non-null task is completed by one of the operators. This is certainly not the only strategy. For example, an alternative strategy would be to re-plan only if a non-null task is finished and the actual task duration is significantly different than expected. Exploration of different re-plan strategies should be assessed in future work.*

**Remark 7** (Task Pool Evolution)**.** *A natural question that arises from the presented multiple-operator, receding horizon strategy is whether it is possible to simply omit tasks that are already in progress from the re-plan MILP, rather than altering scenarios and including additional constraints that require the completion of such tasks. This alternative strategy is not possible given the presented MILP framework, as the actual task end-times are still unknown at the time of re-planning.*

## IV. Heuristic Approach

The multiple operator formulations of the previous sections may be appropriate for small scheduling problems; however, the introduction of additional variables, which are necessary to move from a single to a multiple operator framework, and the additional planning instances that are required by adaptive approaches can significantly increase computation time, as shown in Fig. 5. This figure was generated for an example problem involving $N = 10$ tasks, $T_H = 30$, $\zeta = 2$, $\underline{w} = 0.3$, $\overline{w} = 0.7$, $W_{0,k} = 0.3$ for all $k$, $p_\alpha = 10$, and $p_\beta = p_\gamma = 15$. Note that even with a moderate number of tasks and operators, the inflation of the state space caused by the introduction of new variables and adaptive strategies significantly increases computation time. We remark that the plot in Fig. 5 is only meant to give the reader a flavor for the usual trends that are observed with respect to

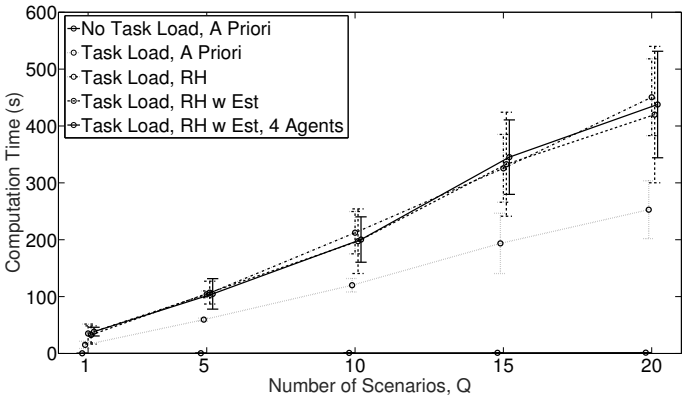



**Fig. 5 Observed computation times for different solution methodologies.**

computation time, and is not meant to portray a strict relationship to be observed in every problem instance. Indeed, due to the use of heuristic solvers (namely, Matlab's `intlinprog` solver), observed computation times rely heavily on the particular problem setup and the parameters used to initialize solvers. For instance, there may be some cases in which moving from a single agent setup to a 2 agent setup actually *reduces* computation time due to the nature of the state space. Despite this, the general trend remains in that computation time diverges quickly to an impractical level as the number of agents grows and additional problem complexities are added.

This discussion motivates the need for alternative means for solving the multiple operator problem. In this section, we introduce one such method, and illustrate how this alternative method can still achieve adequate performance, while using only a fraction of the computation time necessary for solving the problem with the naive approaches of Sections II D, III A, and III B. The idea behind this alternative approach is straightforward: instead of solving a single optimization across all operators, we use a heuristic to assign tasks to specific operators beforehand, and subsequently solve the resulting single operator problems. In this section, we restrict our attention to *a priori* planning strategies, although one could easily construct an analogous approach that re-plans in a receding-horizon or other adaptive fashion.

Consider first the case of a single scenario, i.e., $Q = 1$. Our proposed assignment strategy proceeds in a methodical fashion, in which agents are cyclically selected one at a time according to a pre-defined order. When an agent is selected, a single task is assigned to the selected agent out of

the pool of remaining tasks, based on scores that are assigned to each task. The algorithm evolves in the following manner. In this algorithm, the pool of available tasks includes the null-task.

1. Initialize statistics $\alpha = \beta = \gamma = 0$, select $\lambda \in (0, 1]$.

2. For each operator, initialize statistics $\theta_k = 0$, $\xi_k = W_{0,k}$.

3. Select a permutation of the set $\{1, \ldots, K\}$ to act as the selection order

4. Until all tasks have been assigned, do the following (cycling through the selection order):

   (a) For the next operator in the selection order, say operator $k$, calculate the theoretical times $t_i^{\text{end}}$ that operator $k$ could finish each task $T_i$, assuming that $\theta_k$ is the current time and that the times prescribed by the scenario are the true task processing times.

   (b) Calculate theoretical task load values $w_i^{\text{end}}$ that would be attained if each task $T_i$ were to be processed next, under the same assumptions of the previous step, along with the additional assumption that $\xi_k$ is operator $k$'s current task load level.

   (c) For each $i$, if $w_i^{\text{end}} - \overline{w} > \beta$ set $\overline{\epsilon}_i = w_i^{\text{end}} - \overline{w} - \beta$ and $\beta = w_i^{\text{end}} - \overline{w}$ , otherwise set $\overline{\epsilon}_i = 0$. If $-w_i^{\text{end}} + \underline{w} > \gamma$ set $\underline{\epsilon}_i = -w_i^{\text{end}} + \underline{w} - \gamma$, otherwise set $\underline{\epsilon}_i = 0$ and $\gamma = -w_i^{\text{end}} + \underline{w}$.

   (d) For each $i$, if $t_i^{\text{end}} - T_H > \alpha$, set $\epsilon_i = t_i^{\text{end}} - T_H - \alpha$ and $\gamma = t_i^{\text{end}} - T_H$, otherwise set $\epsilon_i = 0$.

   (e) For each $i$, calculate a score $\pi_i = (r_i - p_\alpha \epsilon_i - \beta \overline{\epsilon}_i - \gamma \underline{\epsilon}_i) \lambda^{t_i^{\text{end}} - \theta_i}$.

   (f) Find $i^* = \arg\max_i \pi_i$ and, if task $T_{i^*}$ is not a null-task, assign task $T_{i^*}$ to operator $k$.

   (g) Set $\theta_k = t_{i^*}^{\text{end}}$, $\xi_k = w_{i^*}^{\text{end}}$ and, if task $T_{i^*}$ is not a null-task, remove task $T_{i^*}$ from $\mathcal{T}$.

The idea behind this algorithm is to quickly and incrementally simulate a task-execution process for each operator, based on local information, and use the result as a basis for task assignment. Each task is assigned a score based on how much the objective function of (2) would increment if that task were to be executed next in the considered operator's task sequence. The score for each task is discounted by a factor $\lambda^{t_i^{\text{end}} - \theta_i}$, which is proportional to the amount of time that it will take the operator to process the task (assuming that the time indicated by the scenario is accurate). The

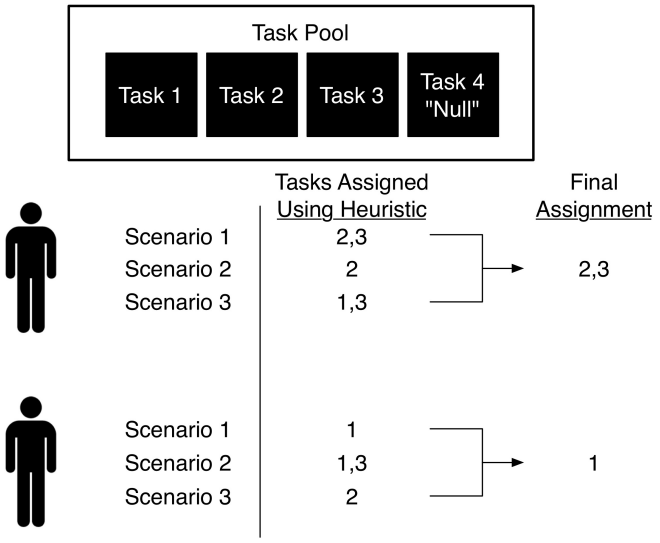



**Fig. 6 An illustration of the proposed task assignment process for a sample mission.**

task with maximum score is assigned to the operator under consideration, and this task is "executed" next in the respective operator's simulated schedule.

Since we have only used one scenario thus far, the previously discussed algorithm suffers from the same deficiency as traditional deterministic scheduling schemes in the presence of uncertain processing times. Namely, we have performed task assignments based on the assumption that the processing times for a single scenario are true. To incorporate some degree of robustness into the heuristic task assignment scheme, we propose the following algorithm (see Fig. 6).

1. Generate a set of $Q$ scenarios, and initialize a matrix `counts` as a $N \times Q$ matrix of zeros.

2. For each scenario $m$:

   (a) Use the previous algorithm to obtain a target-task assignment.

   (b) For each operator $k$, and each task $T_i$ assigned to operator $k$, increment the $(i,k)$-th element of the matrix `counts` by 1.

3. To obtain the final target task assignment, assign each task $T_i$ to agent $k* := \arg\max_i \texttt{counts}(i,\cdot)$.

This new algorithm has the added feature that it bases the final target assignment on a series of heuristic target assignment operations, and assigns to each task the agent that is most likely to

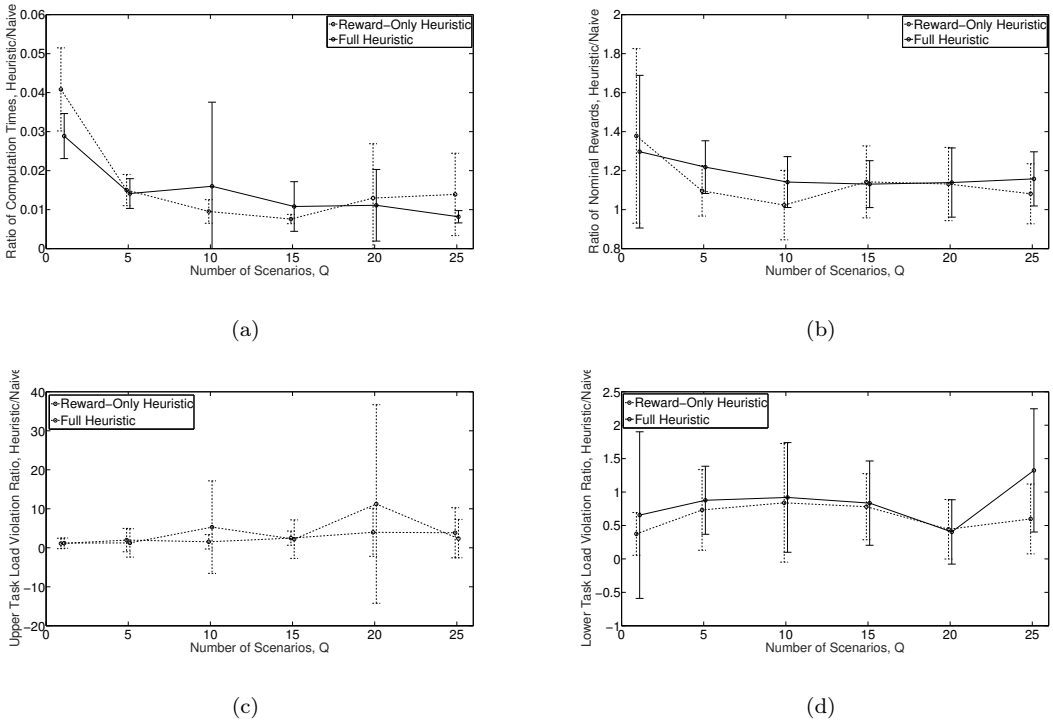



**Fig. 7 Performance of heuristic task assignment methods in comparison to "naive" scheduling.**

comprise the best task-operator assignment.

One other small detail that must be addressed is the procedure to be used if there is a "tie" between two or more operators when executing the final step of the algorithm. One possibility is to randomly select one of the operators as the final assignment. Although valid, this strategy does not take into account the tasks that may already be assigned to the set of operators that are tied. A potentially better approach in the case of ties is to assign the task according to either the global end times of the task, or the number of tasks that are already in the various operator's assignment. For example, if there is a tie between operators 1 and 2, but operator 1 has already been assigned 5 tasks and operator 2 has not been assigned any task, then it may be beneficial to give the task to operator 2. Thorough comparisons between these variations should be assessed in future research.

Fig. 7 shows a performance comparison between 3 solution methods for a sample 3-operator scheduling problem, using a "least number of tasks" tie-breaking procedure. The solution methods used are 1) the "naive" approach of Section II D, 2) the full heuristic assignment and solution approach of the present section, and 3) a "reward-only" heuristic assignment strategy that is analogous to the full heuristic strategy, except scores are assigned to tasks solely based on the rewards $r_i$ during

assignment, i.e., the strategy is the same as that discussed above, except the value of $\underline{\epsilon_i}$, $\overline{\epsilon_i}$, and $\epsilon_i$ are always taken to be 0. The plots illustrate the average over 10 simulation runs of the ratio between the computation times, obtained nominal rewards, and the task load violations for the heuristic strategies to the respective variables for the naive strategy. Notice that the computation times for the heuristic strategies is only a small fraction of that necessary for the naive strategy, while the performance remains roughly the same. In fact, the heuristic strategies even out-performed the naive strategy in some cases. The reason that this is possible is again due to the nature of the heuristic solvers used. To prevent excessively long computation times, a bound was placed on number of nodes that `intlinprog` solver was permitted to explore during its implicit branch and bound solution phase ($10^5$). This bound sometimes limited the quality of the solutions produced by the naive strategy, allowing it to be out-performed by the heuristic method. This result only serves to further make the case that heuristic methods may be preferable to the naive approach.

**Remark 8** (Task Assignment)**.** *We have chosen to assign tasks to operators on the basis of which operator was assigned a given task most often upon iterating through the various scenarios. Loosely speaking, this is a sort of "maximum likelihood" approach in the sense that it selects, for each task, the operator that is most likely to be assigned to that task, given a sample of the multi-variate processing time distribution that generated each scenario. As before, many other methods can be used instead, as long as the chosen strategy selects exactly* 1 *operator for each scenario. For example, it is conceivable that choosing the "least likely" operator may, in some cases, lead to increased robustness. Once again, we leave a thorough study of these concepts as a topic of future research.*

## V. Conclusion

In the context of human-UAV collaborative missions, it is crucial to develop task scheduling methodologies that 1) take human cognitive requirements into account, 2) are robust to uncertainty in system and modeling parameters, 3) can incorporate a wide range of design goals, and 4) are simple enough to practically implement. Our presented MILP framework can potentially accomplish all of these goals. Indeed, we have illustrated how this framework can incorporate task load, introduce robustness, and expand to handle a number of additional layers of complexity, while staying within

a straightforward framework that is familiar to theoreticians and practitioners alike.

With this preliminary work in hand, future work should focus on verifying the utility of these scheduling frameworks through human subjects experiments. In particular, these studies should seek to 1) verify that performance actually does, in fact, improve with the introduction of such scheduling systems, 2) investigate whether simple task load models are sufficient for these applications or if more elaborate models should be considered, and 3) identify any other unforeseen practical or performance issues that may arise. With this knowledge in hand, additional theoretical work can be explored on how to incorporate additional layers of complexity and tailor system designs to particular human-UAV missions. Furthermore, we envision that this kind of technology could be applied to a multitude of other domains, including healthcare and manufacturing applications. In addition to the suggestions throughout the text, other interesting future research avenues include a thorough investigation of how to choose optimal parameters for a given application domain or set of design goals, a study of the domain of applicability of the various solution methods, e.g., a study of the conditions under which estimation is useful, and a comparison between different heuristics for task assignment, given the timescales on which a particular mission operates.

**Acknowledgements**

This paper is based on work supported by the US Air Force Research Lab (AFRL), Air Force Office of Scientific Research, under contract FA9550-14-C-0022.

**References**

[1] Hoffman, M., "New Reaper sensors offer a bigger picture," *Air force times*, Vol. 19, 2009.

[2] Cummings, M. L. and Mitchell, P., "Automated scheduling decision support for supervisory control of multiple UAVs," *Journal of Aerospace Computing, Information, and Communication*, Vol. 3, No. 6, 2006, pp. 294–308.

[3] Freed, M., Harris, R., and Shafto, M., "Human-interaction challenges in UAV-based autonomous surveillance," *Proceedings of the 2004 Spring Symposium on Interactions Between Humans and Autonomous Systems Over Extended Operations*, 2004.

[4] Lawler, E., Lenstra, J. K., Kan, A. H. R., and Shmoys, D. B., "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, Vol. 4, 1993, pp. 445–522.

[5] Yamada, T. and Nakano, R., "Job shop scheduling," *IEE control Engineering series*, 1997, pp. 134–134.

[6] Alidaee, B. and Li, H., "Parallel machine selection and job scheduling to minimize sum of machine holding cost, total machine time costs, and total tardiness costs," *Automation Science and Engineering, IEEE Transactions on*, Vol. 11, No. 1, 2014, pp. 294–301.

[7] Mason, S. and Oey, K., "Scheduling complex job shops using disjunctive graphs: A cycle elimination procedure," *International Journal of Production Research*, Vol. 41, No. 5, 2003, pp. 981–994.

[8] Cheng, R., Gen, M., and Tsujimura, Y., "A tutorial survey of job-shop scheduling problems using genetic algorithms-I. Representation," *Computers & Industrial Engineering*, Vol. 30, No. 4, 1996, pp. 983–997.

[9] Ouelhadj, D. and Petrovic, S., "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, Vol. 12, No. 4, 2009, pp. 417–431.

[10] Ben-Tal, A., El Ghaoui, L., and Nemirovski, A., *Robust Optimization*, Princeton University Press, 2009.

[11] Westman, M. and Eden, D., "The inverted-U relationship between stress and performance: A field study," *Work & Stress*, Vol. 10, No. 2, 1996, pp. 165–173.

[12] Teigen, K. H., "Yerkes-Dodson: A law for all seasons," *Theory & Psychology*, Vol. 4, No. 4, 1994, pp. 525–547.

[13] Broadbent, D., "Differences and interactions between stresses," *Quarterly Journal of Experimental Psychology*, Vol. 15, No. 3, 1963, pp. 205–211.

[14] Hancock, P. A., "A dynamic model of stress and sustained attention," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, Vol. 31, No. 5, 1989, pp. 519–537.

[15] Zimmerman, M., "Task Load," *Encyclopedia of Clinical Neuropsychology*, edited by J. Kreutzer, J. DeLuca, and B. Kaplan, Springer, 2011, pp. 2469–2470.

[16] Donmez, B., Nehme, C., and Cummings, M., "Modeling workload impact in multiple unmanned vehicle supervisory control," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions On*, Vol. 40, No. 6, 2010, pp. 1180–1190.

[17] Cummings, M. and Mitchell, P., "Operator scheduling strategies in supervisory control of multiple UAVs," *Aerospace Science and Technology*, Vol. 11, No. 4, 2007, pp. 339–348.

[18] Peters, J. R. and Bertuccelli, L. F., "Robust Scheduling Strategies for Collaborative Human-UAV Missions," *2016 American Control Conference (ACC)*, AACC, 2015, Submitted.

[19] Savla, K. and Frazzoli, E., "A dynamical queue approach to intelligent task management for human operators," *Proceedings of the IEEE*, Vol. 100, No. 3, 2012, pp. 672–686.

[20] Neerincx, M., "Cognitive task load design: Model, methods and examples," *Handbook of cognitive task design*, 2003, pp. 283–305.

[21] Tsai, Y.-F., Viirre, E., Strychacz, C., Chase, B., and Jung, T.-P., "Task performance and eye activity: Predicting behavior relating to cognitive workload," *Aviation, Space, and Environmental Medicine*, Vol. 78, No. Supplement 1, 2007, pp. B176–B185.

[22] Bertsimas, D. and Tsitsiklis, J. N., *Introduction to Linear Optimization*, Vol. 6, Athena Scientific, Belmont, MA, 1997.

[23] Lawler, E. and Wood, D., "Branch-and-bound methods: A survey," *Operations Research*, Vol. 14, No. 4, 1966, pp. 699–719.

[24] Achterberg, T., Berthold, T., and Hendel, G., "Rounding and propagation heuristics for mixed integer programming," *Operations Research Proceedings 2011*, Springer, 2012, pp. 71–76.

[25] MathWorks, "Linear Programming and Mixed-Integer Linear Programming," `http://www.mathworks.com/help/optim/linear-programming-and-mixed-integer-linear-programming.html`, Accessed: 2015-08-29.

[26] Grant, M. and Boyd, S., "Graph implementations for nonsmooth convex programs," *Recent Advances in Learning and Control*, edited by V. Blondel, S. Boyd, and H. Kimura, Lecture Notes in Control and Information Sciences, Springer-Verlag Limited, 2008, pp. 95–110, `http://stanford.edu/~boyd/graph_dcp.html`.

[27] Grant, M. and Boyd, S., "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," `http://cvxr.com/cvx`, March 2014.

[28] Southern, D., "Human-guided management of collaborating unmanned vehicles in degraded communication environments," Tech. rep., DTIC Document, 2010.

[29] Bertuccelli, L., Pellegrino, N., and Cummings, M., "Choice modeling of relook tasks for UAV search missions," *American Control Conference (ACC), 2010*, IEEE, 2010, pp. 2410–2415.

[30] Nehme, C. E., *Modeling human supervisory control in heterogeneous unmanned vehicle systems*, Ph.D. thesis, Massachussetts Institute of Technology, 2009.