Machine Learning for Efficient Sampling-based Algorithms in Robust Multi-Agent Planning under Uncertainty

John F. Quindlen[*]
Massachusetts Institute of Technology, Cambridge, MA, 02139

Jonathan P. How[†]
Massachusetts Institute of Technology, Cambridge, MA, 02139

Robust multi-agent planning algorithms have been developed to assign tasks to cooperative teams of robots operating under various uncertainties. Often, it is difficult to evaluate the robustness of potential task assignments analytically, so sampling-based approximations are used instead. In many applications, not only are sampling-based approximations the only solution, but these samples are computationally-burdensome to obtain. This paper presents a machine learning procedure for sampling-based approximations that actively selects samples in order to maximize the accuracy of the approximation with a limited number of samples. Gaussian process regression models are constructed from a small set of training samples and used to approximate the robustness evaluation. Active learning is then used to iteratively select samples that most improve this evaluation. Three example problems demonstrate that the new procedure achieves a similar level of accuracy as the existing sample-inefficient procedures, but with a significant reduction in the number of samples.

I. Introduction

Robotic vehicles such as unmanned aerial vehicles (UAVs) are currently used to complete dull, dirty, or dangerous tasks and will be increasingly relied upon to conduct complex missions with little-to-no human supervision. This has spurred the development of autonomous task allocation and planning algorithms to coordinate teams of cooperative robotic vehicles. In order to more effectively pair available agents (i.e. robots, UAVs) with uncompleted tasks (mission objectives), these task planning approaches are often formulated as combinatorial optimization problems.[1–3] The agents are each assigned an ordered sequence of tasks to complete within a specified time window and are "rewarded" for successful completion of each task in that window. The optimization problem then attempts to maximize the cumulative reward through different allocations of tasks to the agents.

In real-life scenarios, the execution of the task assignments from the task allocation method is complicated by the presence of numerous uncertainties. For instance, teams of autonomous robots typically operate in dynamic and poorly-modeled environments. In many applications, these uncertainties in the true environment can be captured as parametric uncertainties in the underlying system models. The parametric uncertainties can affect various portions of the planning model such as the vehicle dynamics, task execution times, or the rewards for completing tasks and therefore must be carefully examined to identify their impact. For example, uncertain wind speed and direction can drastically affect UAV travel time, fuel burn rate, and duration of surveillance tasks when coordinating teams with UAVs. In the worst case, a particular task allocation strategy may become completely infeasible for certain parameter settings. The issue with planning under parametric uncertainties is that the exact values of the parameters during real-world execution is unknown; therefore, the realized planning performance score after execution can take any number of values.

[*]Graduate Student, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology
[†]Richard C. Maclaurin Professor of Aeronautics and Astronautics, Aerospace Controls Laboratory, MIT

Thus, the potential impact on the mission performance motivates the need for robust planning algorithms that account for these uncertainties.

Robust planning algorithms were developed to select the best task assignment in order to minimize the effect of uncertainties on the final mission score. While standard task allocation methods will produce a single task assignment optimized for a single vector of assumed values of the parameters, robust algorithms must return a single task assignment that works over a range of possible parameter values. Robust score metrics[1,4,5] such as the expectation of the score or chance-constraints are used to quantify the effect of uncertainties upon the realized scores of a particular task assignment. Robust planning methods then determine an appropriate task allocation from these robust scores given the uncertainties likely to be encountered. An issue that arises in many applications, is the difficulty required to analytically compute these robust scores.[6] The underlying mapping from uncertain parameters to score is usually unknown and can be highly nonlinear. In those cases, approximate scores can be more easily computed using sampling-based procedures that obtain the weighted score from various realizations of the uncertain parameters. While they lack the exactness of analytical solutions, these statistical scores are more tractable and can become arbitrarily close to the correct, analytical ones after as the number of samples increases.

The work in this paper focuses on robust planning problems where it is computationally-intensive to obtain even a single sample of the score. The high associated with each sample limits the utility of sampling-based procedures, even if they are the only available method, because of the large numbers of samples required to produce a solution. In particular, one motivating example is an aerial forest firefighting mission where UAVs are assigned to complete specific fire monitoring tasks related to the spread of the forest fire. As the spread of the forest fire is affected by uncertain parameters like wind speed, wind direction, and vegetation type, these parameters will affect the completion of firefighting tasks and must be incorporated into the planning stage. Fire simulation models[7,8] can be used to accurately predict the fire spread given a set of these uncertain parameters; however these simulators are typically resource-intensive. For instance, a simplified Matlab-based simulator derived from these models can take 50 seconds to complete a single sample on a quad-cored Ubuntu machine with 8GB of RAM. Therefore, it is desirable to limit the number of fire simulations required to compute the robust task assignment. Similar example problems can be seen across a wide variety of sampling-dependent applications.

Machine learning approaches offer the potential to more efficiently sample the uncertainty space with fewer number of samples, but simultaneously minimize the impact on the accuracy of the robust score metrics associated with fewer samples. Bayesian nonparametric modeling techniques[9] have been used to form accurate approximations of complex functions using a small number of samples and the resulting models can also be used to predict the function behavior over unobserved regions of the feasible space. In particular, Gaussian process (GP) regression models[9] return mean and covariance functions that both predict the unknown true function's response and quantify the confidence in those predictions. Another machine learning technique, active learning,[10,11] can be used alongside regression models to actively select informative samples. Since the number of samples will be limited, it is desirable to pick the "best" samples to achieve the most accurate approximation. Here, "best" refers to the samples which have the greatest impact on the approximation error. Active learning has also been proven to improve score-like approximations in similar applications.[12–14] Consequently, GP prediction models coupled with active learning offers a potential solution to accurately approximate the robustness metrics using limited computational resources.

This paper draws upon machine learning techniques to develop a new sampling-based procedure for robust multi-agent planning algorithms. In particular, this new procedure is used to replace existing sampling procedures in the robust consensus-based bundle algorithm (CBBA) architecture,[5,6,15] a distributed planning framework for networked multi-agent teams. The technique will also apply to other centralized planning architectures with little modifications, but they are not explicitly addressed in this paper. The new sampling procedure combines GP regression models with active learning in order to effectively compute the robust score with a limited number of samples. These samples are chosen in order to minimize the entropy of the expected score, which consequently improves the accuracy of the robust score approximation. The entire process is demonstrated on multiple planning under uncertainty problems, including the aforementioned aerial firefighting example. These examples show the new procedure has similar performance to the existing, more computationally-intensive procedures, but with significantly fewer samples.

II. Problem Description

The following section describes the multi-agent task planning problem for systems subject to parametric uncertainties. A more-detailed description of the robust multi-agent planning problem can be found in these previous works.[5,6]

A. Problem Formulation

Given N_a number of agents and N_t number of tasks to complete, the multi-agent task allocation problem attempts to maximize the mission performance. This mission performance is defined by a global objective function that captures the costs or rewards associated with the assignment of agents to tasks. In many applications, these tasks can only be completed by one agent at a time. This paper also makes that standard assumption and therefore the global objective function can be rewritten as the sum of local objective functions defined by each agent and their assigned task(s). As a result, the global task allocation problem can be written as a mixed-integer nonlinear optimization program

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{i=1}^{N_a} \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}, \boldsymbol{\theta}) x_{ij} \\ \text{s.t.} \quad & \mathbf{G}(\mathbf{x}, \boldsymbol{\theta}) \leq \mathbf{b} \\ & \mathbf{x} \in \mathcal{X} \end{aligned} \tag{1}$$

where design vector $\mathbf{x} \in \mathcal{X}$ is the assignment of all agent-task pairings, with x_{ij} denoting whether agent i is assigned to task j, i.e. $\mathcal{X} = \{0, 1\}^{N_a \times N_t}$. Vector $\boldsymbol{\theta}$ consists of the planning parameters that may influence the objective cost c_{ij}. This cost function maps the cost or reward obtained by agent i for completing task j to the set of global task assignments $\mathbf{x}$ and planning parameters $\boldsymbol{\theta}$. In order to capture vehicle dynamics and other limitations, the nonlinear constraints $\mathbf{G}$ and $\mathbf{b}$ are placed on the optimization problem.

Additionally, in many real-world problems of interest, the rewards for completing tasks vary explicitly with time. For instance, time-varying rewards enable the possibility of time windows within which the task can only be performed or time-critical tasks that should be performed sooner rather than later. For example, firefighting UAVs must consider the movement of a fire front over time during the assignment of fire monitoring tasks. The agent would not receive any reward for completing a fire-related task either before or after the fire front has moved through the area. In short, it is important to consider not only *if* a task is assigned to a particular agent, but also *when* that agent completes the task. The mixed-integer problem in Eq. 1 can be reformulated to include task service times in addition to task assignments

$$\begin{aligned} \max_{\mathbf{x}, \boldsymbol{\tau}} \quad & \sum_{i=1}^{N_a} \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\theta}) x_{ij} \\ \text{s.t.} \quad & \mathbf{G}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\theta}) \leq \mathbf{b} \\ & \mathbf{x} \in \mathcal{X} \\ & \boldsymbol{\tau} \in \mathcal{T} \end{aligned} \tag{2}$$

where decision variable $\boldsymbol{\tau} \in \mathcal{T}$ is the execution sequence. Scalar term τ_{ij} denotes the time when agent i will execute the assigned task j or $\tau_{ij} = \emptyset$ if task j is not assigned to agent i. As a result, the costs c_{ij} and constraints $\mathbf{G}$ are functions of the task assignment $\mathbf{x}$, execution sequence $\boldsymbol{\tau}$, and planning parameters $\boldsymbol{\theta}$.

B. Planning under Uncertainty

As mentioned earlier, one important issue that can arise during the planning and execution stages is uncertainty in the underlying system model. The planning algorithm might utilize one realization of the system model while the true system model is different. As the discrepancies between models increase, the mission performance typically degrades as the tasks are no longer optimally assigned. In particular, this work addresses parametric uncertainties where the system model correctly captures the coupling of the planning parameters $\boldsymbol{\theta}$ to the objective and constraint functions, but the exact value of the true $\boldsymbol{\theta}$ is unknown. While the true value of $\boldsymbol{\theta}$ is unknown, it is assumed that a likelihood model of the uncertainty is known ahead of time. This model consists of the complete set of feasible parameter values Θ, where all $\boldsymbol{\theta} \in \Theta$, and a probability distribution over set Θ, labeled as $\mathbb{P}(\boldsymbol{\theta})$. For instance, wind direction is typically an uncertain model parameter with known upper and lower bounds. During actual real-world implementation, rough estimates of wind direction can be obtained from noisy meteorological sensors, resulting in a distribution over likely wind directions. This knowledge can then be incorporated into the planning algorithm in order to improve the mission score.

While the likelihood model of the uncertainty should be incorporated into the planning approach, it is difficult to predict the impact of the parametric uncertainties upon the total mission score due to the nonlinearities, coupling, and interdependencies between $\mathbf{x}$, $\boldsymbol{\tau}$, and $\boldsymbol{\theta}$ in functions c_{ij} and $\mathbf{G}$ from Eq. 2. For example, uncertainty in the duration of one task would affect the service times of subsequent tasks and possibly cause those tasks to be dropped or completed with a diminished reward. Robust multi-agent task allocation algorithms explicitly account for this variability using the known uncertainty model. More specifically, they employ robustness metrics $\mathbb{M}_\theta$ to quantify the effect of the uncertainties on the mission execution. For planning under uncertainty, the objective function in Eq. 2 is modified with a robustness metric $\mathbb{M}_\theta$,

$$\max_{\mathbf{x},\boldsymbol{\tau}} \mathbb{M}_\theta \Big\{ \sum_{i=1}^{N_a} \sum_{j=1}^{N_t} c_{ij}(\mathbf{x},\boldsymbol{\tau},\boldsymbol{\theta}) x_{ij} \Big\} \tag{3}$$

where this metric can be chosen from a number of suitable approaches such as expected value, worst-case value, or chance-constraints.[6] Discussed in the next section, this paper will use the expected value robustness metric, although the work could easily be adapted for use with the other metrics.

III. Robust Consensus-Based Bundle Algorithm for Multi-agent Planning

This paper highlights the challenges encountered with sampling-based robust planning through explicit demonstration on the consensus-based bundle algorithm planning framework. The following section provides background material on this framework and motivates the underlying problem. Additional information on the consensus-based bundle algorithm framework can be found in earlier work.[6]

The consensus-based bundle algorithm (CBBA) is a multi-agent task allocation strategy for teams of cooperative agents operating in a distributed manner.[5] In known environments, the baseline CBBA procedure is able to guarantee conflict-free solutions and the algorithm runs in polynomial-time with good scalability for increasing numbers of agents and tasks. Robust CBBA methods[6] were developed in order to extend CBBA to uncertain environments. These extensions also demonstrated the suitability of robust CBBA methods for distributed planning[16] through flight test experiments and other hardware demonstrations. This section will summarize the important aspects of robust CBBA and highlight the challenges associated with implementation of the sampling-based algorithm.

A. Overview of the Robust Consensus-Based Bundle Algorithm

The underlying Robust CBBA algorithm is summarized in Algorithm 1. Since CBBA is a distributed framework, it consists of two planning phases: the bundle construction phase and the task consensus phase. In the bundle construction phase (Alg. 1, line 3), each agent greedily generates their own suggested task assignment bundles $\mathcal{A}_i$. These bundles consist of the set of tasks currently assigned to the agent and the path order of which they will be executed. Alongside their suggested task assignment bundles, each agent also computes bids $\mathcal{C}_i$ corresponding to each task in their assignment. This bid information contains the agent's cost/reward associated with successful completion of the tasks in the specified path order and their knowledge of other agent's assignments. The bid information $\mathcal{C}_i$ is then used in the task consensus phase (Alg. 1, lines 4-6) to converge towards a final conflict-free assignment $\mathcal{A}$ between all the distributed agents. Note that $\mathcal{I}$ is the set of all agents ($|\mathcal{I}| = N_a$) and $\mathcal{J}$ is the set of all tasks ($|\mathcal{J}| = N_t$).

While the task consensus phase is fairly straightforward once the bids are available, it is not as easy to obtain the bundles and corresponding bids due to the uncertainty in the planning parameters $\boldsymbol{\theta}$. The costs and rewards associated with the completion of tasks in a specified order will usually vary with $\boldsymbol{\theta}$; therefore, the bids $\mathcal{C}_i$ and bundle assignments $\mathcal{A}_i$ will also vary according to $\boldsymbol{\theta}$. Since the bundle construction phase can only return a single sequence of assignments and bids, these assignments should be chosen so as to be robust to uncertainties in $\boldsymbol{\theta}$.

One of the most common and straightforward approaches to obtain a robust task allocation is to optimize the task assignments with respect to the expected mission performance. In such an approach, the expected

Algorithm 1 Robust Consensus-Based Bundle Algorithm

```
1: Initial set of bundle assignments and corresponding bids: $\{\mathcal{A}_i, \mathcal{C}_i\}$, $\forall i \in \mathcal{I}$
2: while not converged do
3:     $(\mathcal{A}_i, \mathcal{C}_i) \leftarrow$ Robust-CBBA-Bundle-Construction($\mathcal{A}_i, \mathcal{C}_i, \mathcal{J}$), $\forall i \in \mathcal{I}$
4:     $\mathcal{C}_i \leftarrow$ CBBA-Communicate($\mathcal{C}_i, \mathcal{C}_{N_i}$) $\forall i \in \mathcal{I}$
5:     $(\mathcal{A}_i, \mathcal{C}_i) \leftarrow$ CBBA-Bundle-Removal($\mathcal{A}_i, \mathcal{C}_i$), $\forall i \in \mathcal{I}$
6:     converged $\leftarrow \bigwedge_{i \in \mathcal{I}}$ Check-Convergence($\mathcal{A}_i$)
7: end while
8: Combine each agent's bundle assignment: $\mathcal{A} \leftarrow \bigcup_{i \in \mathcal{I}} \mathcal{A}_i$
9: return bundle assignment $\mathcal{A}$
```

value operator is used as the robustness metric in Eq. 3.

$$\max_{\mathbf{x},\boldsymbol{\tau}} \mathbb{E}_\theta \Big\{ \sum_{i=1}^{N_a} \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\theta}) x_{ij} \Big\} \tag{4}$$

Note that this is not the same as planning with the mean parameter values. That approach effectively decouples the uncertainty from the computation of the score and will most likely result in very poor performance.[6] Leveraging the linearity of the expected value operator, the expected value of the randomly distributed (with respect to $\boldsymbol{\theta}$) objective costs in Eq. 4 is equivalent to the sum of the expected values. The objective functions can then be decoupled from the sum over agents.

$$\mathbb{E}_\theta \Big\{ \sum_{i=1}^{N_a} \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\theta}) x_{ij} \Big\} = \sum_{i=1}^{N_a} \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}, \boldsymbol{\tau}, \boldsymbol{\theta}) x_{ij} \Big\} \tag{5}$$

The centralized problem can then be decomposed into subproblems for each agent

$$\begin{aligned} \max_{\mathbf{x}_i,\boldsymbol{\tau}_i} \; & \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}_i, \boldsymbol{\tau}_i, \boldsymbol{\theta}) x_{ij} \Big\} \\ \text{s.t.} \quad & \mathbf{G}(\mathbf{x}_i, \boldsymbol{\tau}_i, \boldsymbol{\theta}) \leq \mathbf{b} \\ & \mathbf{x}_i \in \mathcal{X} \\ & \boldsymbol{\tau}_i \in \mathcal{T} \end{aligned} \tag{6}$$

and the global score is the sum over the agent-specific scores.

Since the execution times of the tasks depend on the order in which they are arranged, the path variable $\mathbf{p}_i(\mathbf{x}_i) = \mathbf{p}_i$ is introduced to define the order in which the tasks specified in $\mathbf{x}_i$ are executed. Path $\mathbf{p}_i$ is thus an ordered sequence of tasks. In general, each agent has an upper bound on the number of tasks it can feasibly execute within the time interval. As such, a maximum path length $|\mathbf{p}_i| \leq L_i$ is included to limit the number of assigned tasks. The complete solution to the task allocation problem not only optimizes the assignment of tasks but also the order in which they're executed. The optimization program in Eq. 6 can then be recast in terms of the path $\mathbf{p}_i(\mathbf{x}_i)$.

$$\max_{\mathbf{x}_i,\boldsymbol{\tau}_i} \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\mathbf{x}_i, \boldsymbol{\tau}_i, \boldsymbol{\theta}) x_{ij} \Big\} = \max_{\mathbf{p}_i,\boldsymbol{\tau}_i} \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij} \Big\} \tag{7}$$

The optimization can be solved as a two-step process,

$$\max_{\mathbf{p}_i,\boldsymbol{\tau}_i} \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij} \Big\} = \max_{\mathbf{p}_i} \Big(\max_{\boldsymbol{\tau}_i} \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij} \Big\} \Big) \tag{8}$$

where the inner portion obtains the optimal task service times and the outer portion iterates over possible paths. In uncertain environments, the computation of the optimal execution times is difficult as it may vary for different realizations of the uncertainty. For instance, given a path assignment $\mathbf{p}_i(\mathbf{x}_i)$ and realization of the uncertainty $\boldsymbol{\theta}_k$, the optimal task service times are

$$\boldsymbol{\tau}_i^* = \underset{\boldsymbol{\tau}_i}{\operatorname{argmax}} \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}_k) x_{ij} \tag{9}$$

with corresponding optimal cost $J_{\mathbf{p}_i}^k = J_{\mathbf{p}_i}(\boldsymbol{\theta}_k)$.

$$J_{\mathbf{p}_i}^k = \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}^*(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}_k) x_{ij} \tag{10}$$

However, if the realization of the uncertainty is different, $\boldsymbol{\theta}_k \neq \boldsymbol{\theta}_l$, then the optimal task service times might change and result in a different cost, $J_{\mathbf{p}_i}^k \neq J_{\mathbf{p}_i}^l$. Therefore, the total robust path score for each agent is given by the distribution of the scores over $\boldsymbol{\theta}$.

$$\begin{aligned} J_{\mathbf{p}_i} = \mathbb{E}_\theta \Big\{ \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}^*(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij} \Big\} &= \int_{\boldsymbol{\theta} \in \Theta} \Big(\sum_{j=1}^{N_t} c_{ij}(\tau_{ij}^*(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij} \Big) \mathbb{P}(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \int_{\boldsymbol{\theta} \in \Theta} J_{\mathbf{p}_i}(\boldsymbol{\theta}) \mathbb{P}(\boldsymbol{\theta}) d\boldsymbol{\theta} \end{aligned} \tag{11}$$

Algorithm 2 Robust-CBBA-Bundle-Construction($\mathcal{A}_i, \mathcal{C}_i, \mathcal{J}$) for Agent i

```
1: while $|\mathbf{p}_i| \leq L_i$ do
2:   for all available tasks $j \in \mathcal{J} \setminus \mathbf{p}_i$ do
3:     for all available positions in $\mathbf{p}_i$ do
4:       Insert task $j$ into path $\mathbf{p}_i$ at location $n \longrightarrow \mathbf{p}_{i \oplus n_j}$
5:       Compute-Expected-Score $\longrightarrow \hat{J}_{\mathbf{p}_{i \oplus n_j}}$
6:     end for
7:     Maximum score $\hat{J}_{\mathbf{p}_{i \oplus n_j^*}} = \max_{n_j} \hat{J}_{\mathbf{p}_{i \oplus n_j}}$
8:     Marginal score $\Delta J_{ij}(\mathbf{p}_i) = \hat{J}_{\mathbf{p}_{i \oplus n_j^*}} - \hat{J}_{\mathbf{p}_i}$
9:     Compute bid $s_{ij}$ for task $j$ from marginal score $\Delta J_{ij}(\mathbf{p}_i)$
10:    $h_{ij} = \mathbb{I}(s_{ij} > \text{existing bid})$
11:  end for
12:  Optimal task to add to bundle $j^* = \underset{j \notin \mathbf{p}_i}{\operatorname{argmax}} \Delta J_{ij}(\mathbf{p}_i) h_{ij}$
13:  if $(\Delta J_{ij^*}(\mathbf{p}_i) h_{ij^*} > 0)$ then
14:    Add task $j^*$ to the bundle, update path, bids, and times
15:  else
16:    Break
17:  end if
18: end while
19: return bundle $\mathcal{A}_i$ and bids $\mathcal{C}_i$
```

B. Sampling-based Approximate Score

In practice, it is very difficult, if not impossible, to compute the analytical interval from Eq. 11. Instead, an approximation of the expected score can be computed using sampling methods. These sampling methods use N samples of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ with corresponding weights w_k to compute the approximate score.

$$J_{\mathbf{p}_i} \approx \sum_{k=1}^{N} w_k J_{\mathbf{p}_i}^k \tag{12}$$

These weights w_k are functions of the probability distribution, $w_k = f(\mathbb{P}(\boldsymbol{\theta}))$. The previous approach[6] for approximating the expected score for a specific path $\mathbf{p}_i$ is given in Algorithm 3. The robust score computation from Algorithm 3 is then used within the distributed planning framework. The complete bundle construction algorithm for each agent is shown in Algorithm 2. Within Alg. 2, the sampling-based approximate score computation is repeated for each available task and path position (Alg. 2, line 5).

The difficulty with the sampling-based score approximation methods is that the approximate score computation requires a large number of simulations. For example, the previous work with robust CBBA[6] commonly uses 50,000 samples to compute the approximate score for each candidate path $\mathbf{p}_i$. This process is then repeated for each remaining available task (Alg. 2, line 2) until the path length is filled. Finally, those steps are repeated for each of the N_a agents. If the computation of the score $J^k_{\mathbf{p}_i}$ is computationally inexpensive, then this process is straightforward; however, the score is not always so easy to compute. In many applications, the score calculation is computationally expensive and it would be infeasible to run such a large number of calculations. For instance, consider the aerial firefighting problem from Section I. The assignment of UAVs to fire monitoring tasks explicitly depends on the locations of the fire at future instances of time. Given relevant parameters, namely wind direction and speed, fire simulation models[7,8] can be used to accurately predict the fire spread and determine the resulting firefighting task locations and durations. While these simulation models are necessary to compute the task assignments, they do require additional computation resources. A simplified Matlab-based fire simulator derived from these models can take 50 seconds to compute the relevant task information for a single realization of $\boldsymbol{\theta}_k$, depending on the resolution of the fire simulation grid. It would be impossible to perform thousands of fire simulations for each candidate path, but arbitrarily capping the number of simulations can also lead to poor solutions. Regardless of the application, it is desirable to minimize the number of simulations required to compute the approximate score for a given task assignment without sacrificing the accuracy of the approximation.

Algorithm 3 Compute-Expected-Score: Pure Monte Carlo

```
1: ${\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N}$ samples from $\mathbb{P}(\boldsymbol{\theta})$
2: $\{w_1, \ldots, w_N\}$ corresponding weights
3: $\{w_1, \ldots, w_N\} \longleftarrow \{w_1, \ldots, w_N\} / \sum_{k=1}^{N} w_k$
4: for $k = 1 : N$ do
5:     $\boldsymbol{\tau}_i^* = \underset{\boldsymbol{\tau}_i}{\text{argmax}} \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}^*(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij}$
6:     $J^k_{\mathbf{p}_i} = \sum_{j=1}^{N_t} c_{ij}(\tau_{ij}^*(\mathbf{p}_i(\mathbf{x}_i)), \boldsymbol{\theta}) x_{ij}$
7: end for
8: $J_{\mathbf{p}_i} = \sum_{k=1}^{N} w_k J^k_{\mathbf{p}_i}$
9: return $J_{\mathbf{p}_i}$
```

IV. Machine Learning Methods for Efficient Sampling

This work uses machine learning techniques to address the difficulties encountered with sampling-based approximations described in Section III-B. Machine learning methods are employed to construct a model from a small number of samples, predict the cost over a wider range of unseen $\boldsymbol{\theta}$ values, and select further samples from Θ in order to improve these approximations. In particular, Gaussian process models are used for regression and prediction from a small set of samples and active learning is used to select future sample points. The following section describes a new procedure to replace Algorithm 3 that relies upon fewer samples of the score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$.

A. Gaussian Process Regression Model

Gaussian process (GP) regression methods, also known as Kriging, have been used to model unknown functions from observed sample data in a variety of aerospace applications. These applications range from surrogate models for system design optimization,[17,18] efficient flight simulation models,[19,20] and nonparametric learning-based control.[21,22] In all these applications, GPs are used to create a Bayesian nonparametric model from a finite number of observed samples. The power of these nonparametric models arises from the fact they provide mean and covariance functions that can be used to both efficiently predict the response over unobserved regions of the input space and quantify the confidence in those predictions. This scalability and flexibility make GP regression models well-suited to the problem with sampling-based score approximations.

By definition, a Gaussian process is a collection of random variables, any finite subset of which are

Gaussian distributed.[9] A Gaussian process can then be viewed as a distribution over possible functions. For the robust task allocation problem, this distribution is written as

$$J(\boldsymbol{\theta}) \sim \mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')) \tag{13}$$

where $m(\boldsymbol{\theta})$ is the mean function and $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')$ is the covariance function.

$$m(\boldsymbol{\theta}) = \mathbb{E}_\theta[J(\boldsymbol{\theta})] \tag{14}$$
$$\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}') = \mathbb{E}_\theta[(J(\boldsymbol{\theta}) - m(\boldsymbol{\theta}))(J(\boldsymbol{\theta}') - m(\boldsymbol{\theta}'))] \tag{15}$$

In this work, the covariance function $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')$ uses the squared exponential.

$$\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}') = \alpha^2 \exp(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')\Lambda^{-1}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T) \tag{16}$$

The signal variance is given by scalar α^2 while matrix Λ is the diagonal matrix of the length scales for each dimension of $\boldsymbol{\theta}$. The GP hyperparameters (α^2, Λ) are selected during the training process.

1. *Training and Prediction*

Given a path $\mathbf{p}_i(\mathbf{x}_i)$, the GP regression model of the unknown, true score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ is constructed from a training set $\mathcal{S}$. This training set consists of parameter samples $\boldsymbol{\theta}_k \in \Theta$ and their corresponding true score $J^k_{\mathbf{p}_i}$. The complete set of all these parameter samples is set Θ_S and the set of their corresponding score evaluations is $J_{\mathbf{p}_i}(\Theta_S)$. The training set $\mathcal{S}$ is then used to construct a Gaussian process regression model as in Eq. 13. More detailed information on the GP training and hyperparameter selection processes can be found in Chapters 2 and 5 of Rasmussen and William's textbook.[9]

With this trained regression model, the GP can be used to predict the score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ over the entire feasible set Θ. Given a sample $\boldsymbol{\theta}_* \in \Theta$, the joint predictive distribution $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta}_*)$ is defined by a posterior mean $\mu(\boldsymbol{\theta}_*)$ and covariance function $\Sigma(\boldsymbol{\theta}_*)$ where matrix $K = \kappa(\Theta_S, \Theta_S)$.

$$\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta}_*) \sim \mathcal{N}(\mu(\boldsymbol{\theta}_*), \Sigma(\boldsymbol{\theta}_*)) \tag{17}$$

$$\mu(\boldsymbol{\theta}_*) = \kappa(\boldsymbol{\theta}_*, \Theta_S)K^{-1}J_{\mathbf{p}_i}(\Theta_S) \tag{18}$$
$$\Sigma(\boldsymbol{\theta}_*) = \kappa(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \kappa(\boldsymbol{\theta}_*, \Theta_S)K^{-1}\kappa(\Theta_S, \boldsymbol{\theta}_*) \tag{19}$$

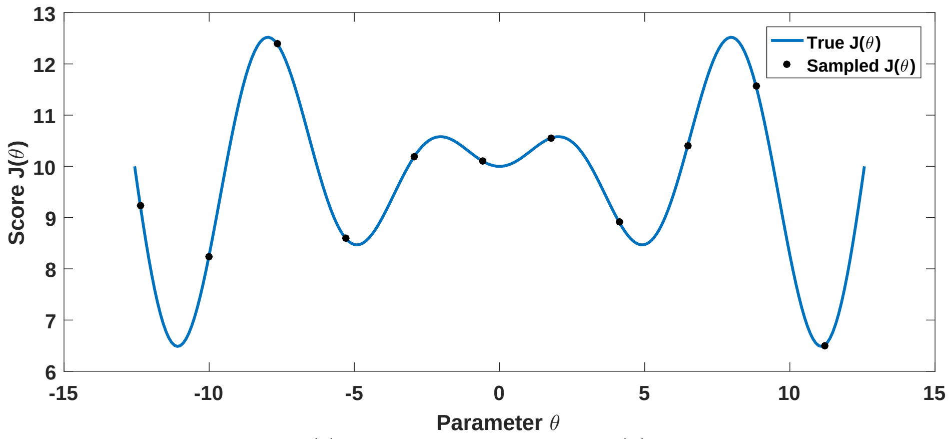
The posterior mean represents the estimated score at $\boldsymbol{\theta}_*$ while the covariance $\Sigma(\boldsymbol{\theta}_*)$ captures the confidence in that prediction. Note that since the samples $J_{\mathbf{p}_i}(\Theta_S)$ are exact values of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ rather than noisy measurements of it, the GP prediction model assumes noise-free observations. An example of the GP regression model and its predictive capabilities is shown in Figure 1. In this example, the true function is sampled at 11 equally-spaced values of $\boldsymbol{\theta}$, shown in Figure 1(a). In Figure 1(b), these samples are used form a GP regression model where the mean function approximates the true function. The plot also shows the 95% confidence intervals ($\pm 2\sigma$ bounds) on the predictions calculated with the covariance function. Notice that the covariance shrinks at each sample point from Θ_S as their is no uncertainty with a noise-free observation of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$.

The GP prediction model trained from set $\mathcal{S}$ can then be used to estimate the expected value of the score function

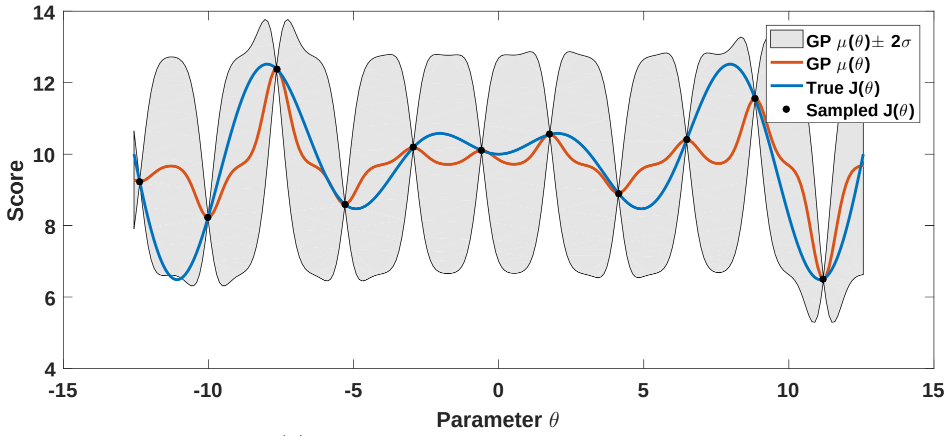
$$J_{\mathbf{p}_i} \approx \hat{J}_{\mathbf{p}_i} = \mathbb{E}_\theta[\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})] = \int_{\boldsymbol{\theta} \in \Theta} \hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{20}$$

where the actual expected value of the score was given in Eq. 11, $J_{\mathbf{p}_i} = \mathbb{E}_\theta[J_{\mathbf{p}_i}(\boldsymbol{\theta})]$. Given the training set $\mathcal{S}$, the expected value in Eq. 20 can be reduced to

$$\mathbb{E}_\theta[\hat{J}_{\mathbf{p}_i}|\mathcal{S}] = \int_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_\theta[\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})]\mathbb{P}(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{21}$$
$$= \int_{\boldsymbol{\theta} \in \Theta} \mu(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{22}$$



(a) Sampled score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$



(b) GP regression model and predictions

Figure 1. Gaussian process regression model fit to samples of the score function $J(\boldsymbol{\theta}) = J_{\mathbf{p}_i}(\boldsymbol{\theta})$. The top figure shows 11 equally-spaced samples of $\boldsymbol{\theta}$ and corresponding $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ used to form the training set $\mathcal{S}$. The lower figure shows the GP regression model trained from $\mathcal{S}$ and its prediction $\mu(\boldsymbol{\theta})$ over the entire feasible set Θ. This figure also shows the 95% confidence intervals ($\pm 2\sigma$ bounds) on the prediction.

where $\mu(\boldsymbol{\theta})$ is the posterior predictive mean function from before. Just as in Eq. 12, this analytical integral can be numerically approximated with a finite number of data points.

$$\hat{J}_{\mathbf{p}_i} \approx \sum_{k=1}^{N} w_k \mu(\boldsymbol{\theta}_k) \tag{23}$$

For this new approximation, the weights w_k are functions of the probability $\mathbb{P}(\boldsymbol{\theta})$ selected based upon the Bayesian Quadrature rule.[12–14,23] Central to this approach is the assumption that it is significantly less computationally expensive to train the GP regression model and predict than it is to simply obtain additional samples of the score function. As long as this assumption holds and training set $\mathcal{S}$ is smaller than the evaluation set, $|\mathcal{S}| << N$, it is more computationally efficient to predict the score $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ over the N samples from Θ as in Eq. 23 than it is to actually sample $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ as in Eq. 12. This approach (Algorithm 4) will then replace pure Monte Carlo estimation (Algorithm 3) for the computation of the expected score in Algorithm 2, line 5.

While it may be more computationally efficient to compute $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ and $\hat{J}_{\mathbf{p}_i}$ using the Gaussian process prediction model, it does not necessarily guarantee the GP approximation is accurate. If the training set is poorly chosen, then the GP estimate $\hat{J}_{\mathbf{p}_i}$ could differ quite dramatically from the true value $J_{\mathbf{p}_i}$. The following subsection presents an algorithm that not only computes $\hat{J}_{\mathbf{p}_i}$, but iteratively improves this estimate to reduce the approximation error $\hat{J}_{\mathbf{p}_i} - J_{\mathbf{p}_i}$.

Algorithm 4 Compute-Expected-Score: Passive Learning

```
1: Input: set $\mathcal{U}$, set $\mathcal{S}$
2: Train a regression model $\mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^T))$ using training set $\mathcal{S}$
3: Compute estimated scores $\hat{J}^k_{\mathbf{p}_i} = \mu(\boldsymbol{\theta}_k)$ for all $\boldsymbol{\theta}_k \in \mathcal{S} \cup \mathcal{U}$
4: Estimate expected score $\hat{J}_{\mathbf{p}_i} = \sum_{k=1}^N w_k \hat{J}^k_{\mathbf{p}_i}$
5: return $\hat{J}_{\mathbf{p}_i}$
```

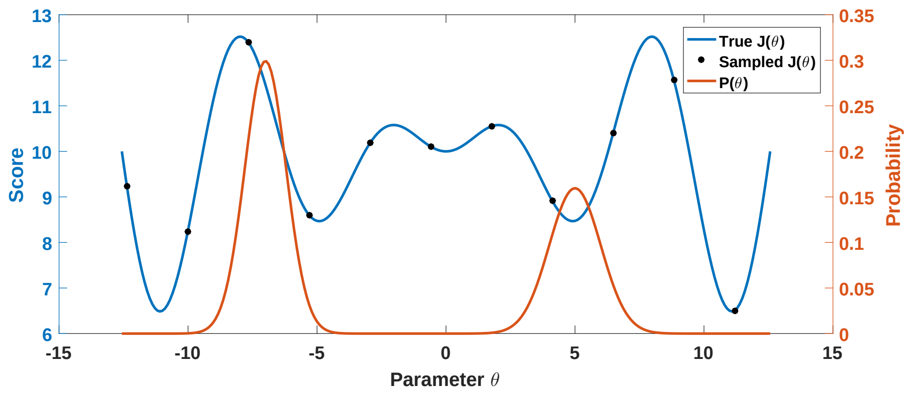


Figure 2. Probability distribution of the parameter $\mathbb{P}(\boldsymbol{\theta})$ compared to samples of the true score function $J(\boldsymbol{\theta}) = J_{\mathbf{p}_i}(\boldsymbol{\theta})$. Note that samples in regions around $\boldsymbol{\theta} = 0$ or $\boldsymbol{\theta} = 10$ have zero probability and will thus have no effect upon $\hat{J}_{\mathbf{p}_i}$.

B. Active Learning for Sample Selection

The last subsection illustrated that a trained GP regression model can be used to efficiently predict the score over the entire set of Θ; however, the GP model can also be used to intelligently select future samples in order to improve the accuracy of these predictions. In a realistic scenario, only a limited number of additional samples can be taken from the true score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ and the problem is how to select these samples to improve $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ and $\hat{J}_{\mathbf{p}_i}$ the most. In order to address this issue, active learning can be used to select the "best" samples to decrease the approximation error $J_{\mathbf{p}_i} - \hat{J}_{\mathbf{p}_i}$.

Active learning methods are closed-loop statistical learning processes used to select future training points to improve a data-driven model according to an appropriate error metric. While the majority of active learning work has focused on binary classification problems,[10,11,24–26] active learning has been applied to regression problems as well.[12–14,27–30] In particular, the work in this paper modifies earlier active learning regression methods for Bayesian quadrature[12–14] to create an iterative process to select samples and retrain the Gaussian process regression model.

The crux of any active learning procedure is the metric used to evaluate possible samples and subsequently select the next samples to query. As the goal of this work is to decrease the error in the estimated expected score $J_{\mathbf{p}_i} - \hat{J}_{\mathbf{p}_i}$, the "best" samples for this are not necessarily the same as the best samples to improve the estimated scored function $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ directly. Equation 23 suggests that the most appropriate selection criteria is a function of not just $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$, but also the probability distribution $\mathbb{P}(\boldsymbol{\theta})$. This is illustrated in Figure 2, which shows the probability distribution $\mathbb{P}(\boldsymbol{\theta})$ compared to the same true score function $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ from Figure 1. Further samples should ignore regions such as the one around $\boldsymbol{\theta} = 0$ where there is little-to-no probability and thus will have no effect upon $\hat{J}_{\mathbf{p}_i}$ and $J_{\mathbf{p}_i} - \hat{J}_{\mathbf{p}_i}$ even if the accuracy was improved in that region.

The selection metric used to identify the "best" samples to improve $\hat{J}_{\mathbf{p}_i}$ is the minimization of the entropy of the total integrand $J_{\mathbf{p}_i}(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})$ from Eq. 11. This metric has been used before for active learning,[12] known as the uncertainty sampling metric. According to this metric, the parameter vector $\overline{\boldsymbol{\theta}}$ with the largest variance of the integrand $\mathbb{V}[J_{\mathbf{p}_i}(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})]$ should be sampled as this reduces the entropy by the largest amount. The variance of the integrand actually reduces to the weighting of the posterior covariance of the estimated score function with the square of the probability distribution.

$$\mathbb{V}[J_{\mathbf{p}_i}(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})] = \Sigma(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})^2 \tag{24}$$

The best parameter vector to sample, labeled as $\overline{\boldsymbol{\theta}}$, is therefore the point with highest value of this weighted term.

$$\overline{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ \mathbb{V}[J_{\mathbf{p}_i}(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})] \right\} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ \Sigma(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})^2 \right\} \tag{25}$$

Interestingly enough, this is the same result achieved when the samples are chosen to minimize the expected squared loss between $J_{\mathbf{p}_i}$ and $\hat{J}_{\mathbf{p}_i}$.[31]

1. *Active Learning Framework*

The sample selection metric is used as the basis for a closed-loop active sampling procedure to replace the sampling-based procedure detailed in Algorithm 3 and the passive-learning approach in Algorithm 4. Just as was seen in Algorithm 4, the output $\hat{J}_{\mathbf{p}_i}$ is a direct replacement for the numerical approximation $J_{\mathbf{p}_i}$ output from line 9 of Algorithm 3.

The active learning procedure assumes the existence of a large, but finite set of feasible parameter vectors, labeled $\mathcal{F}$, with cardinality N that approximates the uncountable set Θ. Set $\mathcal{F}$ is initially broken up into two subsets, $\mathcal{S}$ and $\mathcal{U}$. Set $\mathcal{S}$ contains a small collection of sampled vectors $\boldsymbol{\theta} \in \Theta_S$ and their corresponding true scores $J_{\mathbf{p}_i}(\Theta_S)$. Set $\mathcal{U}$, known as the unsampled set, is significantly larger and contains the remaining points in $\mathcal{F}$ that have not been sampled. Parameter vectors $\boldsymbol{\theta}_k$ in both $\mathcal{S}$ and $\mathcal{U}$ have their weights w_k precomputed based upon their probability $\mathbb{P}(\boldsymbol{\theta}_k)$. Note that this set $\mathcal{F}$ can be the same exact set of points originally intended for use in Algorithm 3. However, only the training set $\mathcal{S}$ is sampled, not all of $\mathcal{F}$, and these samples are used to construct a Gaussian process regression model $\mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^T))$ (Alg. 5, line 2). The resulting GP predicts the score over the entire set $\mathcal{F}$. As they are explicitly known, the training points in $\mathcal{S}$ will have zero prediction error with a noise-free GP, that is $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta}_k) = J_{\mathbf{p}_i}(\boldsymbol{\theta}_k)$ for all $\boldsymbol{\theta}_k \in \Theta_S$. This was seen at the sample points in Figure 1(b).

With the trained GP regression model, the iterative portion of the active learning procedure can begin. Assuming there is a limit T on the number of additional samples to take beyond the initial training set $\mathcal{S}$, where $|S| + T << N$, the iterative process is repeated T times (line 3). This limit T is set ahead of time by the controls engineer as a suitable upper bound. For instance, this limit can be determined by the number of iterations than can be feasibly achieved within a time restriction. Most active learning work has either assumed this limit T or iterates until the unsampled set $\mathcal{U}$ is empty; however, the latter case is precisely what Algorithm 5 is aiming to avoid. During each of the T iterations, the entropy metric from Eq. 25 identifies the next sample $\overline{\boldsymbol{\theta}}$ to query from the true score (lines 4-5). This data is then added to the training set $\mathcal{S}$ and $\overline{\boldsymbol{\theta}}$ is removed from the unsampled set $\mathcal{U}$ so that the same point is not sampled twice (lines 6-7). The GP model is retrained with this new training set $\mathcal{S}$ and the process is repeated. After the loop terminates, the GP's posterior mean function $\mu(\boldsymbol{\theta})$ is used to compute the estimated scores over the entire set $\mathcal{F}$ (line 10). These estimated scores are then combined with their corresponding weights w_k to obtain the estimate of the expected score $\hat{J}_{\mathbf{p}_i}$ as in Eq. 23.

One step of the loop from Algorithm 5 is depicted in Figure 3. The score function is the same function from Figures 1 and 2. The trained GP from Figure 1 and the selection metric from Eq. 25 is used to select the next sample point, shown in Figures 3(a) and 3(b). After obtaining a measurement of the true score function at this sample point, a new GP regression model is trained. The results are shown in Figure 3(c).

V. Results

This section demonstrates the active learning procedure on multiple task allocation problems. The first two examples better illustrate the active learning process on simpler problems and compare it to passive learning-based procedures as well as the original Monte Carlo approach. The third example is an aerial forest firefighting problem that employs a fire simulation engine to assign surveillance tasks to multiple UAVs. This last example highlights the need for limiting the number of samples as the inefficiency of the fire simulator severely restricts the practicality of Monte Carlo methods.

A. Single Agent Example

The first example demonstrates the active learning-based expected score algorithm (Algorithm 5) on the evaluation of a single path $\mathbf{p}_i$ from Algorithm 2, line 5. In particular, there are two tasks that the agent must complete. Both tasks have a time-varying reward associated with completion of the particular task;

Algorithm 5 Compute-Expected-Score: Active Learning

```
1: Input: set $\mathcal{U}$, set $\mathcal{S}$, limit $T$
2: Train a regression model $\mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^T))$ using training set $\mathcal{S}$
3: for each iteration $t = 1 : T$ do
4:     Select best sample $\overline{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \mathcal{U}}{\operatorname{argmax}} \ (\Sigma(\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})^2)$
5:     Obtain true score $J_{\mathbf{p}_i}(\overline{\boldsymbol{\theta}})$
6:     $\mathcal{S} \longrightarrow$ add $\{\overline{\boldsymbol{\theta}}, J_{\mathbf{p}_i}(\overline{\boldsymbol{\theta}})\}$ to training set $\mathcal{S}$
7:     $\mathcal{U} \longrightarrow$ remove $\{\overline{\boldsymbol{\theta}}, J_{\mathbf{p}_i}(\overline{\boldsymbol{\theta}})\}$ from unsampled set $\mathcal{U}$
8:     Retrain regression model $\mathcal{GP}(m(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^T))$ with new set $\mathcal{S}$
9: end for
10: Compute estimated scores $\hat{J}^k_{\mathbf{p}_i} = \mu(\boldsymbol{\theta}_k)$ for all $\boldsymbol{\theta}_k \in \mathcal{S} \cup \mathcal{U}$
11: Estimate expected score $\hat{J}_{\mathbf{p}_i} = \sum_{k=1}^{N} w_k \hat{J}^k_{\mathbf{p}_i}$
12: return $\hat{J}_{\mathbf{p}_i}$
```

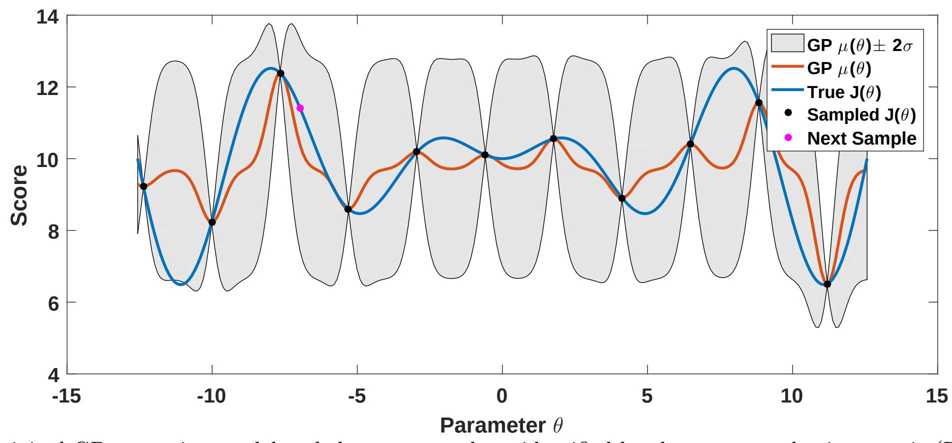
however, the path $\mathbf{p}_i$ is already fixed (Alg 2, line 4) and thus the first assigned task will always be initiated at the same time, resulting in the same reward every time. Essentially, the uncertainty only affects the score of the second task. Both tasks are subject to the uncertain parameters $\boldsymbol{\theta} = [\theta_1, \theta_2]$ which directly affect the time it takes to complete the task. The mapping of the uncertain parameters to the time required to complete a task is shown in Figure 4. The likelihood of these uncertain parameters is given by the multivariate probability distribution depicted in Figure 5. Ultimately, the uncertainty will result in a parameter-varying score function for the given path, $J_{\mathbf{p}_i}(\boldsymbol{\theta})$, illustrated in Figure 6. The goal of the sampling-based procedure is to compute the expected score from samples of this true score function and the probability distribution $\mathbb{P}(\boldsymbol{\theta})$.

The active learning procedure is illustrated in Figures 7 and 8. The initial training set $\mathcal{S}$ consists of 25 randomly selected measurements of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$ which are used to train the initial GP regression model. This trained model is then used to predict the score over the entirety of Θ, shown in Figure 7(a) and (b). The active learning procedure from Algorithm 5 then selects the next sample point from the entropy metric. This process is repeated for $T = 100$ iterations. Figure 7(c)-(f) plot the predicted scores early in this process after 5 and 10 iterations while Figure 8 depicts the scores after 50 and 100 iterations. Even after 100 iterations, with 125 training points in total, the GP model is not able to perfectly capture the true score function from Figure 6. Ultimately, that is unnecessary as the procedure is weighted towards areas of high probability. This is seen in Figure 8, where the majority of the samples are located near regions corresponding to the high probability peaks in Figure 5. The end result is the gradual improvement in the estimated expected score $\hat{J}_{\mathbf{p}_i}$ when using Algorithm 5 depicted in Figure 9. After 100 iterations of the procedure, the estimated expected score $\hat{J}_{\mathbf{p}_i}$ closely matches the actual expected score $J_{\mathbf{p}_i}$. A comparable approximation using Monte Carlo estimation (Algorithm 3) requires 65,000 samples of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$, in comparison to the 125 samples used by active learning. The reduction in the number of samples drastically reduces the computational overhead for problems when it is computationally intensive to obtain samples from $J_{\mathbf{p}_i}(\boldsymbol{\theta})$.

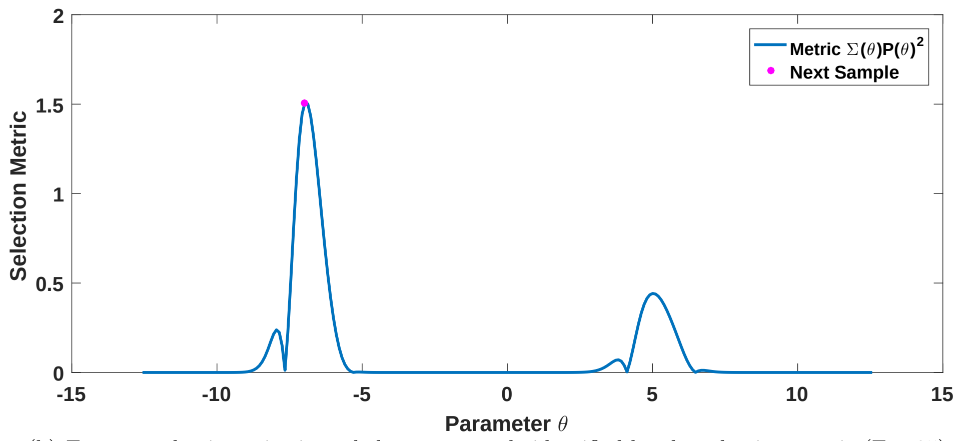
B. Multi-Agent Planning under Uncertainty

The second example demonstrates the entire CBBA assignment procedure on a multi-agent, multi-task planning under uncertainty problem when Algorithm 3 has been replaced by active and passive learning. The planning problem consists of 2 UAV agents and 10 tasks to complete. All of the tasks have the same time-varying reward. Just as in the previous example, the task duration is effected by two uncertain parameters $\boldsymbol{\theta} = [\theta_1, \theta_2]$. These two parameters are defined by the same probability distribution from Figure 5. There are two different types of tasks in this problem (5 each) which are both affected by the uncertainty. These tasks are the function from Figure 4 scaled by factors of 2 and 4, respectively.

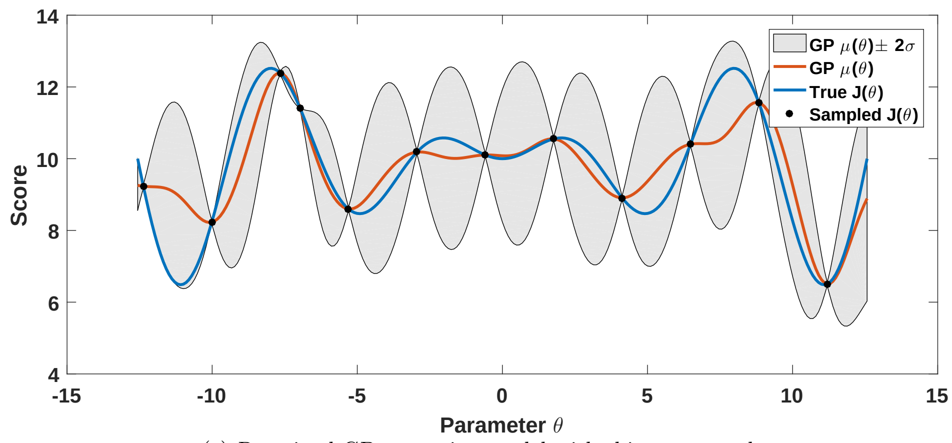
Even though there are only 2 UAVs and 10 tasks in total, the robust CBBA planning procedure requires roughly 400 separate function calls of Algorithm 5 on average before obtaining a suitable task allocation. Each function call utilized 75 training samples to approximate the expected score. After completion of the planning, this translates into roughly 30,000 different samples of score functions. This active learning procedure is compared against the passive learning approach from Algorithm 4 and the Monte Carlo estimation



(a) Original GP regression model and the next sample as identified by the entropy selection metric (Eq. 25)



(b) Entropy selection criteria and the next sample identified by the selection metric (Eq. 25)

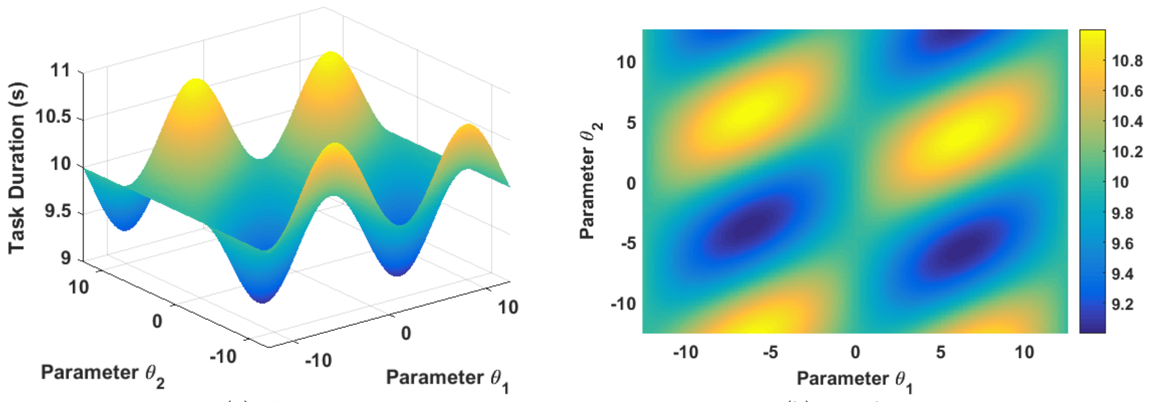


(c) Retrained GP regression model with this new sample

Figure 3. One step of the active learning procedure from Algorithm 5. The original GP regression model and the selection metric are used to select the next sample. This point has the largest entropy in the integrand and will induce the greatest change on the model when sampled. The GP model retrained with this new sample is shown at the bottom.

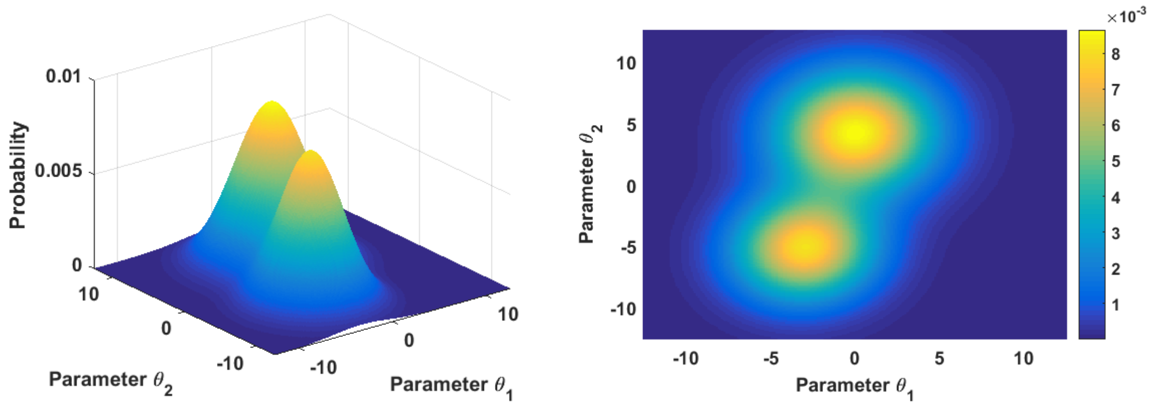
approach from Algorithm 3. In this latter approach, no GP model is constructed, instead a large number of randomly selected samples are drawn according to $\mathbb{P}(\boldsymbol{\theta})$ and the expected score is the statistical average. The number of samples for each function call of Algorithm 3 is gradually increased up to 12,500 for suitable comparison to the learning-based approximations.

Figure 10 shows the normalized approximation error for each of the three approaches. All three strategies



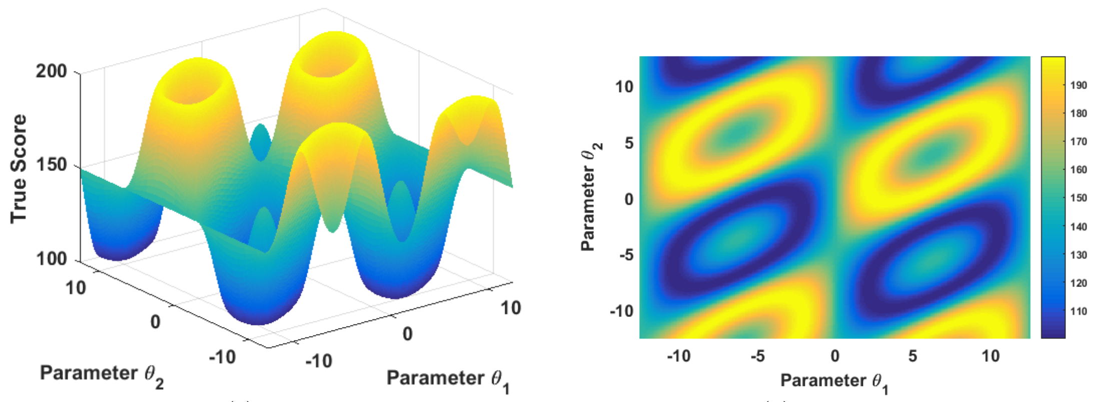
(a) 3D view (b) Top-down view

Figure 4. Task duration (in seconds) as a function of the uncertain parameters θ_1 and θ_2.



(a) 3D view (b) Top-down view

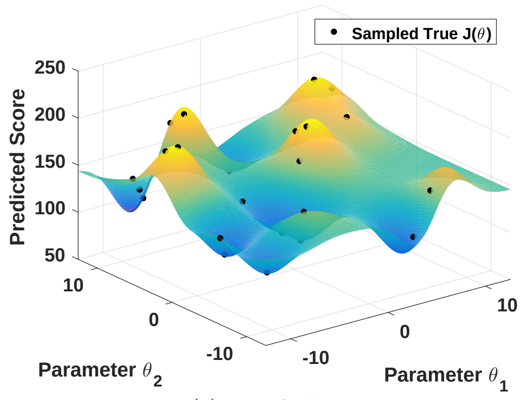
Figure 5. Likelihood of the uncertain parameters θ_1 and θ_2.



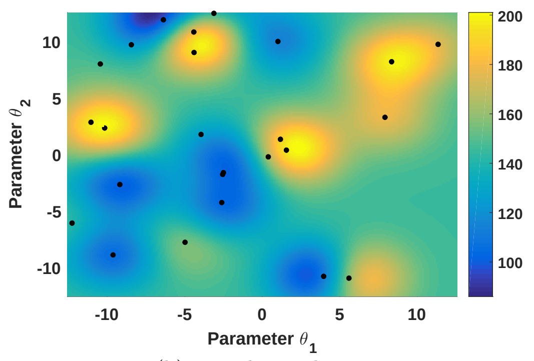
(a) 3D view (b) Top-down view

Figure 6. True score $J_{\mathbf{p}_i}$ as a function of the uncertain parameters θ_1 and θ_2.

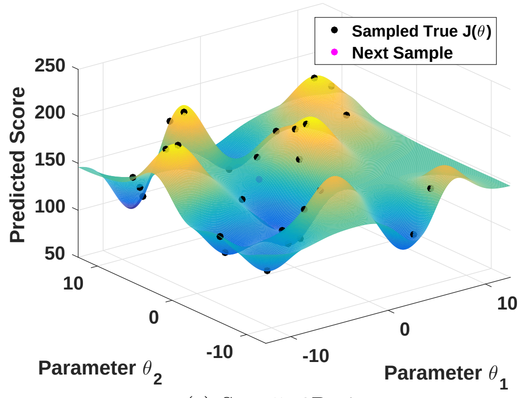
have an average approximation error of around 0%. Note that the mean error from Figure 10(b) is also roughly 0% but is omitted for easier viewing. The main difference between the approaches is in the error bounds. The active learning procedure has the lowest upper bound on the normalized error while the passive learning approach has a slightly higher error. Likewise, the randomized Monte Carlo approach has noticeably worse error bounds. The Monte Carlo estimation procedure required roughly 6000 samples to match the approximation error performance bounds that were achieved by active learning procedure in 75 samples



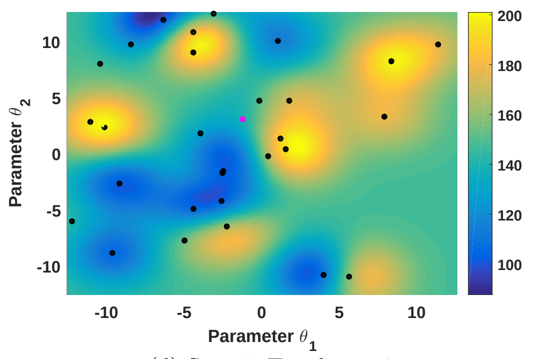
(a) Initial 3D view



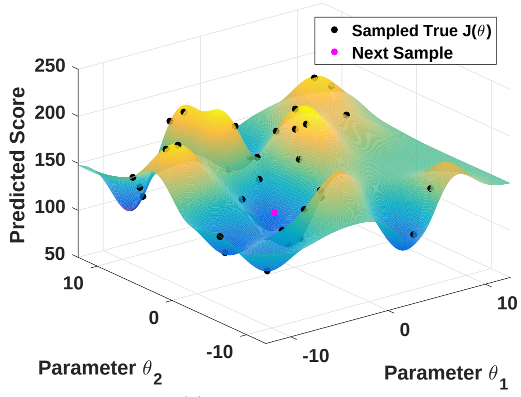
(b) Initial Top-down view



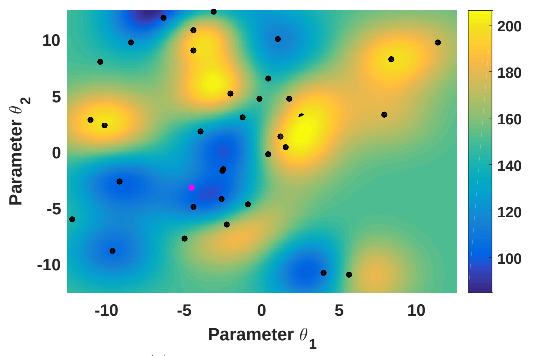
(c) Step 5: 3D view



(d) Step 5: Top-down view



(e) Step 10: 3D view



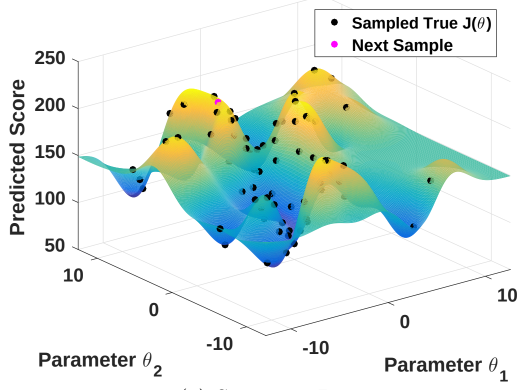
(f) Step 10: Top-down view

Figure 7. Adaptation of the predicted score $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ during the active learning procedure. The first two figures (a) and (b) depict the starting GP model trained using the initial training set $\mathcal{S}$. The latter figures show the predicted score after 5 and 10 iterations of the procedure.

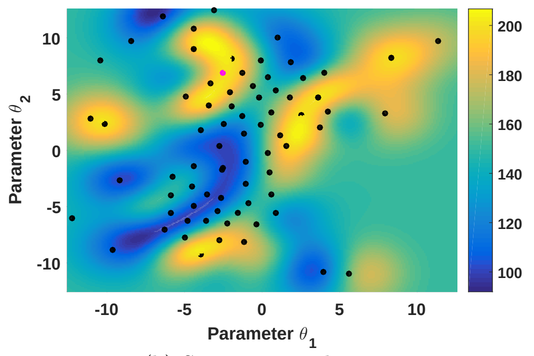
(50 iterations). Even though they all have roughly the same statistical mean error, the active learning procedure will have a lower variance, which is not surprising considering the active learning procedure used the minimization of the variance of the integrand as the selection criteria. These approximation error results highlight the benefits in sample efficiency with an active learning sampling approach.

C. Aerial Forest Firefighting

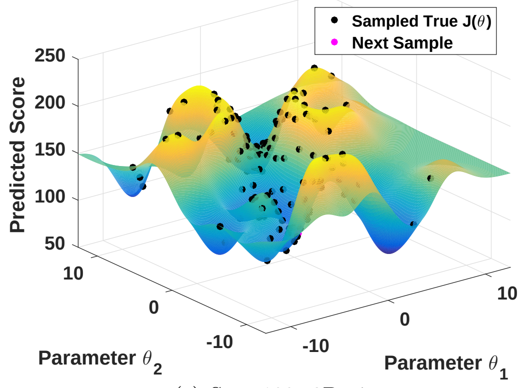
The last example is an aerial forest firefighting task allocation problem. In this example, a team of 2 UAVs on a simplified surveillance mission is used to identify and monitor the expansion of a forest fire. These agents



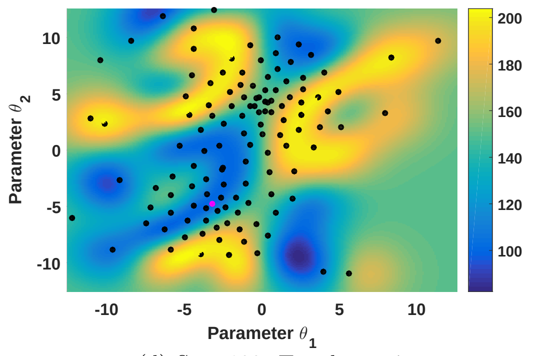
(a) Step 50: 3D view



(b) Step 50: Top-down view



(c) Step 100: 3D view



(d) Step 100: Top-down view

Figure 8. Adaptation of the predicted score $\hat{J}_{\mathbf{p}_i}(\boldsymbol{\theta})$ during the active learning procedure (continuation of Figure 7). The first two figures (a) and (b) show the predicted score after 50 iterations. The latter two show the predict score at the conclusion of the procedure.

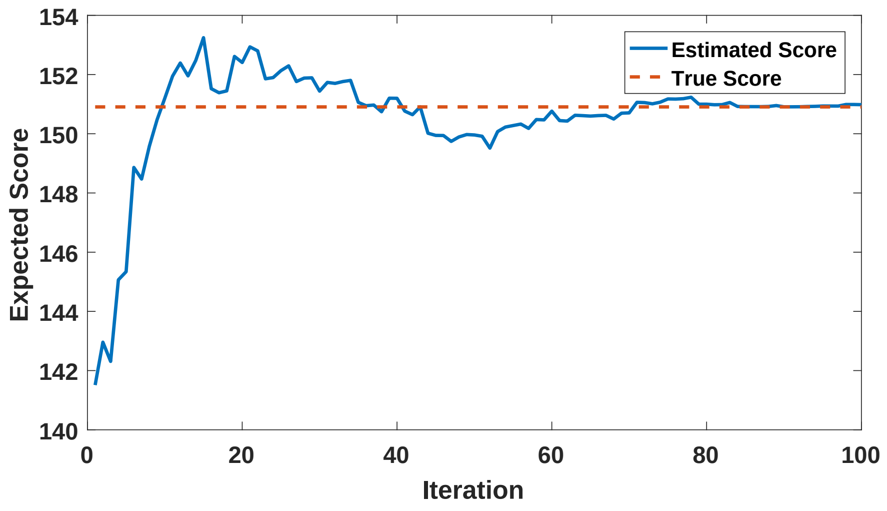
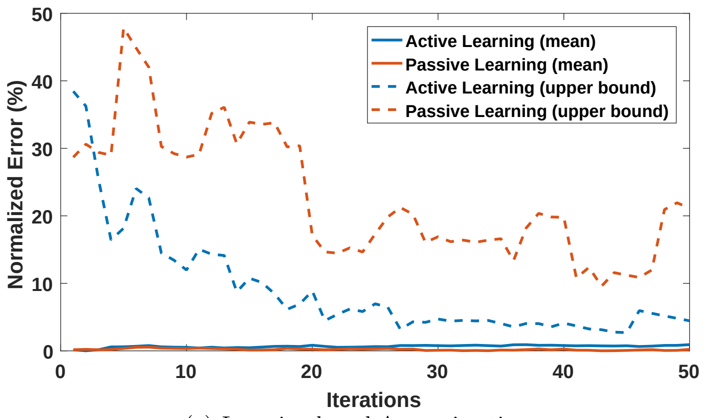
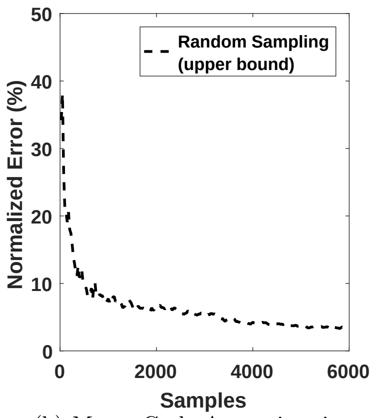


Figure 9. Expected score $\hat{J}_{\mathbf{p}_i}$ after T iterations of the active learning procedure. The active learning procedure only utilizes the best sample from each of the T iterations in addition to 25 initial training points. After $T = 100$ iterations, the updated training set only consists of 125 samples from $J_{\mathbf{p}_i}(\boldsymbol{\theta})$.

must periodically visit the four quadrants of the assigned search zone and map the fire front expansion if any flames are detected within the quadrant. Mapping the fire expansion is a slower process than simply



(a) Learning-based Approximations



(b) Monte Carlo Approximation

Figure 10. Normalized approximation error with the number of samples. Figure 10(a), compares the active learning procedure in Algorithm 5 with a passive learning variation. Both approaches construct a GP prediction model and obtains 1 additional sample during each iteration, but the active learning procedure selects its sample while the passive learning procedure randomly draws one according to $\mathbb{P}(\boldsymbol{\theta})$. Before the first iteration, the GP regression model is constructed using 25 randomly selected training samples. Figure 10(b) shows the normalized approximation error from a Monte Carlo implementation of Algorithm 3. In this plot, the mean is not shown as it was essentially a constant 0.

searching for fires, thus a quadrant's surveillance task will require additional time to complete if flames are present and this time will increase with the growth of the fire within that quadrant. The goal of the task allocation algorithm is to assign surveillance tasks to the UAVs in order to balance quadrant visits with the need to map fire expansion.

One of the major issues that arises in the firefighting problem is that the fire expansion is initially unknown and is greatly affected by uncertain parameters such as wind speed, wind direction, and vegetation types (i.e. grassland, forest, rocky terrain). In this work, it is assumed the initial fire location is known, such as after a lightning strike, along with a map of the vegetation, but the wind speed and direction are poorly modeled with noisy measurements. Despite these uncertainties, the robust CBBA planning algorithm must still provide a single task assignment for the UAVs. Replanning can be used to reassign tasks during the middle of the actual execution, but this work focuses entirely on the initial task assignments. In order to preemptively evaluate the robustness of prospective paths over various parameter combinations, the aforementioned fire simulator must be employed during the construction of the task assignments. Depending on the size of the map and time scales, the simulator may require close to a minute to compute a single fire trajectory. This computational inefficiency served as the primary motivation for the development of sample-efficient methods for robust task allocation.

In the firefighting scenario, the 2 agents can potentially visit the four quadrants four times each, for a total of 16 surveillance tasks of uncertain duration. The estimation of the wind speed and direction resulted in the multivariate distribution shown in Figure 11. Robust CBBA with active learning was then applied to compute a task assignment with desirable robustness to this uncertainty distribution. The resulting task assignment only schedules two visits to each of the quadrants in order to ensure quadrants aren't neglected as the result of unplanned delays. During the computation of the task allocation, there were 760 different function calls to Algorithm 5 and each function call used up to 50 iterations of the active learning step to compute the approximate expected score. The resulting normalized mean and standard deviation over these 760 function calls is shown in Figure 12. These results suggest that fewer than 50 iterations of the active learning procedure can be employed to obtain a reasonably accurate approximation. This reduction in the number of simulated traces greatly enhances the practicality of robust CBBA for the derivation of task assignment strategies for this and similar applications.

While active learning does offer a viable alternative to other sampling-based procedures, each procedure has their own strengths and weaknesses. One potential issue with active learning is that it can induce a small bias in the evaluation whereas randomized Monte Carlo sampling will not. In the results from Figures 10(a) and 12, active learning demonstrates a slight error bias. This is due to the fact samples are chosen based upon an imperfect model of $J_{\mathbf{p}_i}(\boldsymbol{\theta})$, but this bias will decrease as more samples are obtained. Randomized

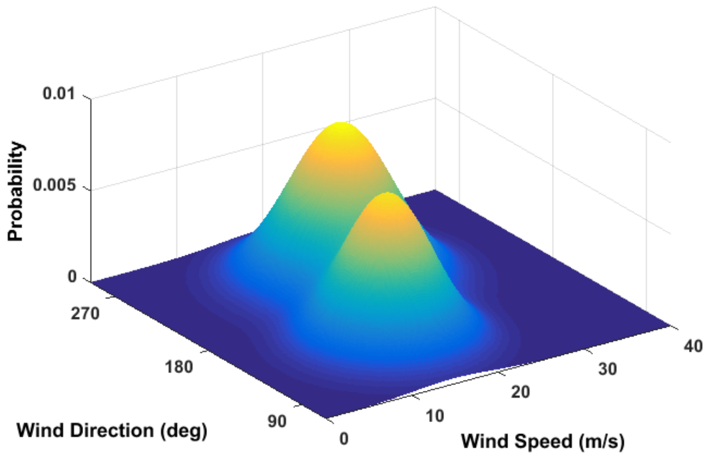


Figure 11. Likelihood of the uncertain wind direction and wind speed.

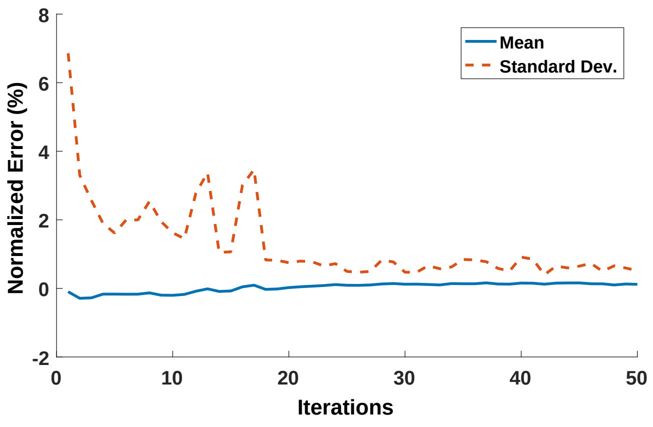


Figure 12. Mean and standard deviation of the normalized error from 760 robust score approximations used to compute the task assignment for the firefighting problem.

Monte Carlo approaches do not suffer from this bias in theory, as they rely upon the law of large numbers; however, this returns to the previous concern with sample efficiency. Ultimately, the choice will depend on the application. These results in this paper demonstrate that the active learning procedure is a suitable replacement for the existing sampling-based procedures when necessary.

VI. Conclusion

This paper presents a machine learning framework for sampling-based robust multi-agent planning under uncertainty. In particular, this work focuses on distributed task allocation using the robust consensus-based bundle algorithm (CBBA), although it can easily be modified for other planning algorithms. Sampling-based methods typically require a high number of simulations in order to compute the robust score. However, in many applications, such as an aerial forest firefighting problem, it is difficult and computationally intensive to determine the robust score. The goal of this work is to reduce the number of simulations required to compute the expected score without sacrificing accuracy.

A machine learning-based algorithm is developed to compute the expected score with few evaluations of the actual score function. Given a likelihood of the uncertain parameters, the algorithm can evaluate a particular task assignment. Gaussian process (GP) regression models are trained from a small subset of samples of the true, computationally-intensive score function. This regression model is used to inexpensively predict the score over the unobserved regions of the parameter space. These predictions are combined with the probability distribution of the parameters to estimate the expected score. Furthermore, a small number of additional samples are actively selected according to an entropy reduction metric. These new samples induce the largest possible improvement in the estimated score. Three example problems demonstrate the new algorithm and the reduction in the number of samples with minimal impact on the accuracy of the estimated scores.

Future work will consider extensions to this algorithm. The most immediate step is to derive batch versions of Algorithm 5 to select multiple samples before retraining the regression model. Batch methods can offer further computational savings with minimal impact on the accuracy by lowering the number of training steps for the same number of samples. Additional work will focus on removing the slight error bias encountered in the last two examples. Ultimately, the active learning process presented here is intended as a tool to extend existing multi-agent planning architectures to more complex real-world applications.

References

[1]Ponda, S. S., Johnson, L. B., Geramifard, A., and How, J. P., *Handbook of Unmanned Aerial Vehicles*, chap. Cooperative Mission Planning for Multi-UAV Teams, Springer, 2012.

[2]Bertsimas, D. and Weismantel, R., *Optimization over integers*, Dynamic Ideas Belmont, MA, 2005.

[3]Boutilier, C., "Planning, learning and coordination in multiagent decision processes," *TARK '96: Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996, pp. 195–210.

[4]Alighanbari, M. and How, J. P., "A Robust Approach to the UAV Task Assignment Problem," *International Journal of Robust and Nonlinear Control*, Vol. 18, No. 2, January 2008, pp. 118–134.

[5]Choi, H.-L., Brunet, L., and How, J. P., "Consensus-Based Decentralized Auctions for Robust Task Allocation," *IEEE Transactions on Robotics*, Vol. 25, No. 4, August 2009, pp. 912–926.

[6]Ponda, S. S., *Robust Distributed Planning Strategies for Autonomous Multi-Agent Teams*, Ph.D. thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, September 2012.

[7]Boychuk, D., Braun, W. J., Kulperger, R. J., Krougly, Z. L., and Stanford, D. A., "A stochastic forest fire growth model," *Environmental and Ecological Statistics*, Vol. 16, No. 2, 2009, pp. 133–151.

[8]Tymstra, C., Bryce, R. W., Wotton, B. M., Taylor, S. W., and Armitage, O., *Development and Structure of Prometheus: The Canadian Wildland Fire Growth Simulation Model*, Candian Forest Service, 2010.

[9]Rasmussen, C. and Williams, C., *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, MA, 2006.

[10]Settles, B., "Active Learning Literature Survey," Tech. rep., University of Wisconsin - Madison, 2010.

[11]Settles, B., *Active Learning*, Morgan and Claypool, 2012.

[12]Gunter, T., Osborne, M. A., Garnett, R., Hennig, P., and Roberts, S. J., "Sampling for Inference in Probabilistic Models with Fast Bayesian Quadrature," *Advances in Neural Information Processing (NIPS)*, 2014.

[13]Briol, F.-X., Oates, C. J., Girolami, M., Osborne, M. A., and Sejdinovic, D., "Probabilistic Integration: A Role for Statisticians in Numerical Analysis?" Tech. rep., arXiv, 2016.

[14]Osborne, M. A., Duvenaud, D., Garnett, R., Rasmussen, C. E., Roberts, S. J., and Ghahramani, Z., "Active Learning of Model Evidence Using Bayesian Quadrature," *Advances in Neural Information Processing Systems*, 2012.

[15]Ponda, S. S., Johnson, L. B., and How, J. P., "Distributed Chance-Constrained Task Allocation for Autonomous Multi-Agent Teams," *American Control Conference (ACC)*, June 2012.

[16]Ponda, S. S., Redding, J., Choi, H.-L., How, J. P., Vavrina, M. A., and Vian, J., "Decentralized Planning for Complex Missions with Dynamic Communication Constraints," *American Control Conference (ACC)*, Baltimore, MD, July 2010.

[17]Giunta, A. A. and Watson, L. T., "A Comparison of Approximation Modeling Techniques: Polynomial Versus Interpolating Models," *AIAA Multidisciplinary Analysis and Optimization Symposium*, 1998.

[18]Jeong, S., Murayama, M., and Yamamoto, K., "Efficient Optimization Design Method Using Kriging Model," *AIAA Journal of Aircraft*, Vol. 42, No. 2, March-April 2005, pp. 413–420.

[19]Schwartz, H. D., Lee, C. H., and Mavris, D. N., "Multifidelity Aerodynamics with Second Order Augmented Kriging Adaptive Sampling," *AIAA Applied Aerodynamics Conference*, 2013.

[20]Othman, N. and Kanazaki, M., "Efficient Flight Simulation Using Kriging Surrogate Model Based Aerodynamic Database," *AIAA Aerospace Sciences Meeting*, 2015.

[21]Chowdhary, G., Kingravi, H. A., How, J. P., and Vela, P. A., "Bayesian Nonparametric Adaptive Control Using Gaussian Processes," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 26, No. 3, March 2015, pp. 537–550.

[22]Cutler, M. J., *Reinforcement learning for robots through efficient simulator sampling*, Ph.D. thesis, Massachusetts Institute of Technology, 2015.

[23]Huszar, F. and Duvenaud, D., "Optimally-Weighted Herding is Bayesian Quadrature," *Conference on Uncertainty in Artificial Intelligence*, 2012.

[24]Scholkopf, B. and Smola, A. J., *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, 2002.

[25]Brinker, K., "Incorporating Diversity in Active Learning with Support Vector Machines," *International Conference on Machine Learning*, 2003.

[26]Yang, X., Song, Q., and Wang, Y., "A Weighted Support Vector Machine for Data Classification," *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 21, No. 5, 2007, pp. 961–976.

[27]Cai, W., Zhang, Y., and Zhou, J., "Maximizing Expected Model Change for Active Learning in Regression," *IEEE International Conference on Data Mining*, 2013.

[28]Castro, R., Willett, R., and Nowak, R., "Faster Rates in Regression via Active Learning," *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[29]Sugiyama, M., "Active Learning in Approximately Linear Regression Based on Conditional Expectation of Generalization Error," *Journal of Machine Learning Research*, Vol. 7, 2006, pp. 141–166.

[30]Sugiyama, M. and Nakajima, S., "Pool-based Active Learning in Approximate Linear Regression," *Machine Learning*, Vol. 75, No. 3, 2009, pp. 249–274.

[31]Berger, J. O., *Statistical Decision Theory and Bayesian Analysis*, Springer, 1985.