Data-driven Metareasoning for Collaborative Autonomous Systems

Jeffrey W. Herrmann

Abstract— **When coordinating their actions to accomplish a mission, the agents in a multi-agent system may use a collaboration algorithm to determine which agent performs which task. This paper describes a novel data-driven metareasoning approach that generates a metareasoning policy that the agents can use whenever they must collaborate to assign tasks. This metareasoning approach collects data about the performance of the algorithms at many decision points and uses this data to train a set of surrogate models that can estimate the expected performance of different algorithms. This yields a metareasoning policy that, based on the current state of the system, estimated the algorithms' expected performance and chose the best one. For a ship protection scenario, computational results show that one version of the metareasoning policy performed as well as the best component algorithm but required less computational effort. The proposed data-driven metareasoning approach could be a promising tool for developing policies to control multi-agent autonomous systems.**

Index Terms—**Distributed task allocation, metareasoning, multivehicle.**

I. Introduction

In many multi-agent systems, the various agents use a collaboration algorithm to determine which agents should perform which tasks. Previous research has studied scenarios in which all agents use the same collaboration algorithm throughout their mission, and many different collaboration algorithms have been proposed and tested, but they may require excessive communication and computation with little additional benefit.

Metareasoning is thinking about thinking, or deciding how to decide. Examples of metareasoning include determining how to make a decision and determining when to stop deliberating and execute an action. Humans use metareasoning to select the most appropriate way to make a decision, which should lead to better decisions; the next wave of AI-enabled autonomous systems could benefit from this as well.

Cooperative search, acquisition, and tracking (CSAT) is an important application for multi-agent autonomous systems in domains such as emergency response, search and rescue, wildfire management, homeland security, and defense. Previous research has developed a variety of CSAT collaboration algorithms, so it should be possible to exploit this diversity by using metareasoning in dynamic, uncertain environments to determine which collaboration algorithm is most appropriate for the current situation. This paper describes a novel data-driven approach for generating a metareasoning policy and presents the results of computational experiments used to evaluate the metareasoning policy for a simulated ship protection scenario. This exploratory research is meant to generate insights into generating and using metareasoning policies for problems such as CSAT.

The problem of protecting naval ships from small boats in a crowded maritime environment involves finding the small boats, which includes harmless craft (such as fishing vessels) and (possibly) adversaries who wish to attack the ship; gathering information about them using various sensors (e.g., radar and electro-optical and infrared cameras); identifying and determining which (if any) are threats; and assessing, tracking, and neutralizing the threats. A variety of unmanned surface vehicles (USVs) and unmanned aerial vehicles (UAVs) may be deployed in these tasks. The objectives are to minimize the likelihood of a successful attack on the ship (by any adversary) and the likelihood of mistakenly attacking a harmless vessel.

The remainder of this paper is organized as follows: Section II discusses related work on CSAT and metareasoning. Section III presents the approach for generating the metareasoning policy. Section IV describes the experiments done to generate the metareasoning policy and the experiments done to evaluate it. Section V presents the results. Section VI discusses the results. Section VII concludes the paper.

II. Related Work

In the CSAT problem [1, 2, 3], multiple agents search for and track multiple moving, noncooperative targets. Examples of targets include injured persons in a search and rescue operation, wildfire patterns in wildfire management, and potentially hostile vessels in maritime environments. Although noncooperative targets do not share their locations with the searchers, they also do not evade the searchers (the problem with evasive targets is a different challenge). To observe the targets, the searchers use sensors that may provide a bearing (the direction from the searcher to the target), a

Manuscript received DATE. This work was supported in part by the U.S. Naval Air Warfare Center-Aircraft Division. J.W. Herrmann is with the Department of Mechanical Engineering and the Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (e-mail: jwh2@umd.edu).

range (the distance from the searcher to the target), or both. Typical sensors have a limited range and are noisy (the measurements have errors). Data fusion approaches [4] can be used to combine the measurements from multiple sensors to estimate a target's location. A searcher must track a target to obtain enough measurements to reduce the uncertainty in its location, at which point the target is "found" and can be rescued or neutralized.

Coordination among searchers may be centralized (one leader), decentralized (multiple leaders), or distributed (no leaders) [3]. The cooperative multi-robot observation of multiple moving targets (CMOMMT) problem can be viewed as a special case of CSAT with perfect sensors, and CMOMMT approaches can be adapted for use in the CSAT setting [5, 6].

Multiple types of planning algorithms have been proposed for CSAT systems. *Task-based approaches* identify specific tasks (such as search, track, and refuel) at specific locations and assign them to agents using mechanisms such as auctions. The consensus-based auction algorithm (CBAA) and consensus-based bundle algorithm (CBBA) [7] have been used and extended by other researchers [1, 8-14]. Other task allocation approaches have been studied by Jin *et al.* [15], Ganapathy and Passino [16], and Singhal and Dahiya [17].

The performance impact algorithm [18, 19] is another distributed task allocation method that systematically considers how to shift tasks from one agent to another to reduce the total cost of the task assignment. It can not only generate a new task assignment but also improve a given task assignment.

Value-based approaches define an objective function that each agent seeks to optimize. One class of objective function uses measures of information such as Fisher information measure [20] and mutual information [21, 22]. Another class uses potential fields that force agents towards targets [6, 23, 24]. Another class generates a reward function based on a partially observable Markov decision process (POMDP) [25-28]. Ahner [29], Terelius *et al.* [30], and Hale *et al.* [31] described more general optimization approaches.

No single approach is ideal. For instance, when using a task-based approach that focuses on tracking targets, a small number of targets can "capture" the searchers' attention while other targets go unobserved. Although consensus-based auction approaches have been successful at coordinating multiple agents, they may use excessive number of messages to converge on task allocations.

Metareasoning. Metareasoning is a formalization of metacognition, which an intelligent agent does when it thinks about its own thinking [32]. An agent uses metareasoning, also known as metalevel control, to improve the quality of its decisions [33]. Examples include determining which algorithm to use to make a decision and determining when to stop computing and execute an action. Cox [32], Cox and Raja [33], Russell and Wefald [34], and Anderson and Oates [35] presented fundamental concepts in metareasoning. Because finding the optimal metareasoning decision online is computationally challenging [36, 37, 38], it is appropriate to determine a policy offline or consider a small number of options.

The algorithm selection problem [39, 40] is a metareasoning problem that is closely related to the proposed metareasoning approach. Smith-Miles [40] proposed a framework that developed algorithm selection rules by correlating "features" of the problem instances and the performance of candidate algorithms on those instances. Leyton-Brown *et al.* [41] used statistical regression techniques to learn models to estimate expected runtime for combinatorial optimization problems; Hutter *et al.* [42] used forward selection to identify the instance features needed as inputs to such models. Munoz *et al.* [43] used a neural network to build a model to predict the expected number of function evaluations needed to solve a continuous optimization problem. Munoz *et al.* [44] and Kotthoff [45] reviewed algorithm selection methods for combinatorial optimization and continuous optimization problems.

Zilberstein [46] proposed the concept of "operational rationality," which determines the best way to use a set of fixed algorithms, and used metareasoning to determine the best deliberation time of anytime algorithms by considering the algorithm's performance profile (see also Zilberstein and Russell [47]). The metareasoning approach proposed herein also attempts to be operationally rational by using performance and computational cost estimates for algorithm selection, as Russell and Wefald did [34].

Distributed metareasoning can be used to coordinate a team of agents [48]. Alexander *et al.* [49] developed a framework for metareasoning in multi-agent systems, in which the agents consider and select their local problem-solving actions while coordinating with the other agents. In a multi-agent system, each agent must also consider whether to communicate with the others. Despite the complexity of the metareasoning task, however, it would be desirable if each agent could develop meta-level plans that consider a sequence of computational actions (not merely the next computation). In Alexander's framework, each agent solves a Markov decision process (MDP), which generates a computation policy that the agent can use to choose the best computation based on the current state. Because the possible actions of the other agents create pending contexts that need to be considered and affect the value of the agent's actions, the agent's metareasoning procedure must alternate between its local decision making and the coordinated decision making of the entire group. Moreover, the agents must share information about their metareasoning decisions.

Sleight and Durfee [50] considered the problem of determining an organizational design that coordinates both the agents' behavior and their reasoning by prohibiting reasoning about certain actions in certain states. For the problem of multiagent tornado tracking, Cheng *et al.* [51] presented a meta-level control approach in which the meta-level selects an abstract action that a lower-level controller implements by developing a detailed plan. Kota *et al.* [52] presented an approach in which agents decide whether to adapt the organization, which consists of relations between agents.

Among the different types of metareasoning approaches that have been proposed, optimal metareasoning, which optimizes overall agent performance given fixed object-level decision-making processes, is the most promising approach for enabling the bounded rationality of an agent due to the ease of implementation and the ability to generate guarantees about agent behavior [53]. (Bounded rationality describes the limits of a decision-maker due to limited time, finite computational resources, and other resource constraints.) It has been employed, for instance, to monitor an anytime algorithm and determine when the algorithm should stop searching and return the best solution found so far.

Previous multi-agent metareasoning approaches explored innovative, relevant ideas but did not directly address the problem of deciding which algorithm an agent should use to plan its next move while participating in a CSAT mission. The work described in this paper contributes to our knowledge of metareasoning and the CSAT problem by proposing a novel data-driven metareasoning approach and using it to generate a metareasoning policy for a specific CSAT problem. Although the experiments described in this paper focus on a specific CSAT problem, the data-driven metareasoning approach is general and can be applied to a variety of CSAT problems and other multi-agent systems.

III. Metareasoning Approach

A. Using a Metareasoning Policy

Before describing the approach used to generate the metareasoning policy, it will be useful to describe how the agents use the metareasoning policy. In the CSAT problem, a decision point is a time at which the available agents must decide which tasks to perform. A decision point may occur when a new task appears, when an existing task is completed, or when the number of available agents changes.

At each decision point, the agents use the metareasoning policy to determine which collaboration algorithm will be used to assign tasks to agents. Based on information about the current environment, the tasks, and the agents (the "state"), the metareasoning policy quickly estimates the performance of every candidate collaboration algorithm using a surrogate model. This performance represents the expected quality of the solution that that collaboration algorithm will generate. The metareasoning policy selects the collaboration algorithm that has the best estimated performance, and the agents use the selected collaboration algorithm to decide which tasks to perform. This structure, in which the agents simultaneously use the same metareasoning policy to select the collaboration algorithm, follows the design principle that the agents' metareasoning be choreographed [49].

Let X be the current state of the system (the current environment, the tasks, and the agents) at a decision point. Let $A=\{a_1,\ldots,a_I\}$ be the set of candidate algorithms. For i = 1, …, I, let $f_i(X)$ be the surrogate model that estimates the performance of algorithm a_i when the state equals X. The cost of computation is also important to consider. Let $c_i(X)$ be the penalty of using algorithm a_i when the state equals X. Then, if a smaller value of performance is preferred, the metareasoning policy is a function $MR(X)$ that selects an algorithm when the state equals X:

$$MR(X)=\underset{i=1,\ldots,I}{\arg\min}\, f_i(X)+c_i(X) \tag{1}$$

B. Generating the Metareasoning Policy

As shown in Figure 1, the proposed data-driven metareasoning approach has four steps:

1. Definition: define the state variables and performance measure and identify the candidate algorithms,

2. Data collection: conduct simulations to collect data on the performance of the candidate algorithms in different states,

3. Model building: use this data to train the surrogate models that can quickly estimate the performance of the candidate algorithms, and

4. Deployment: deploy these surrogate models in a metareasoning policy.

Step 1 (Definition) requires identifying the key variables that describe the state of the system, especially the agents and the tasks that need to be performed, and the performance measure that should be optimized. The set of candidate algorithms A should also be defined. At this point, there is no specific approach for identifying these, although our results give some insights into the desirable characteristics of the performance measure and the set of candidate algorithms (as discussed in Section VI).

Step 2 (Data collection) requires running numerous simulations of the CSAT system. At each decision point t in each replication, every algorithm in A is used to make a potential task assignment. For algorithm $a_i \in A$, the quality Y_{it} of the potential task assignment is determined; this quality is the value of the performance measure defined in Step 1. The current state X_t and quality Y_{it} are saved in a dataset $\mathbf{D}_i$ for this algorithm. One of the potential task assignments is selected at random as the solution for that decision point.

Step 3 (Model building) uses machine learning to generate, for every algorithm $a_i \in A$, a surrogate model $f_i(X)$ such that $f_i(X_t)\approx Y_{it}$. This surrogate model is trained using the inputs in the dataset $\mathbf{D}_i$.

Step 4 (Deployment) adds the surrogate models to the metareasoning policy $MR(X)$.

In this general approach, the state X could be the state of the entire system, known perfectly by every agent, or the partial information that an agent has about the state of the entire system. In the experiments described in Sections IV and V, the agents all have the same information about the state of the system and use the same collaboration algorithm determined by the metareasoning policy.

In principle, the agents could apply all of the collaboration algorithms in A to generate multiple potential task assignments, evaluate the quality of these task assignments, and use the one with the best quality. This would require the computational expense of running all of the collaboration algorithms, however. The proposed approach simplifies the metareasoning and reduces the online computational burden by using the surrogate models instead, which follows the suggestions of previous work that has shown that it is desirable to use simpler approaches for metareasoning due to the lack of exact knowledge and the need to minimize the computational effort used for metareasoning [34, 36, 37].

Although the data collection and model building steps will require computational effort, the simulation replications in the data collection step are independent from each other and can be run in parallel. In addition, the model building tasks for the candidate algorithms are independent from each other and can be run in parallel.

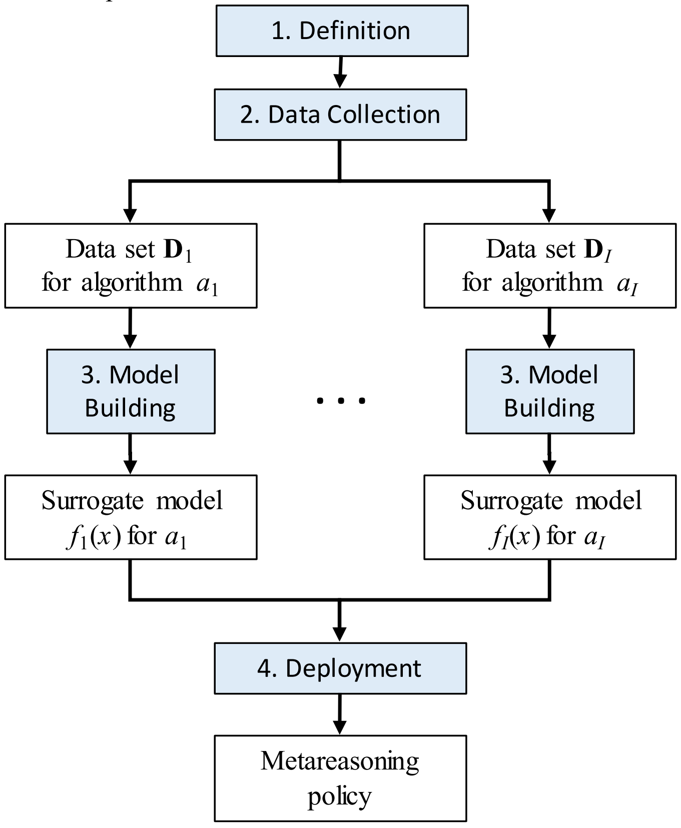
1. Definition
2. Data Collection
Data set $\mathbf{D}_1$ for algorithm a_1
Data set $\mathbf{D}_I$ for algorithm a_I
3. Model Building
3. Model Building
Surrogate model $f_1(x)$ for a_1
Surrogate model $f_I(x)$ for a_I
4. Deployment
Metareasoning policy

Fig. 1. Schematic of the data-driven metareasoning approach.

IV. Experimental Setup

As a test of this data-driven metareasoning approach for collaborative agents, we considered a simplified ship protection scenario that is related to the CSAT problem. In this scenario, a high-value naval asset (HVA) is traveling through a channel towards its destination. The HVA is accompanied by aircraft ("searchers") that fly ahead of the HVA to detect targets and aircraft ("defenders") that move to intercept and classify any targets that are found by the searchers. The targets are surface vessels of two types: (1) "attackers" approach the HVA and attack it when they are close enough to damage the HVA; (2) "non-combatants" move across the channel for their own purposes and pose no threat to the HVA.

Although a searcher can detect attackers and non-combatants, its limited sensors cannot distinguish them, so any surface vessel found is simply labeled as a "target." A defender, when it gets close enough to a target, can classify the target as an attacker or a non-combatant; if the target is an attacker, the defender can also use an anti-ship missile to stop an attacker. In the scenarios considered in this study, a defender has only one missile; when it has used its missile, it leaves the channel and participates in the mission no more.

A. Scenario Description

Some aspects of the scenario are fixed; other aspects are randomly determined when the scenario is initialized. All speeds in knots were converted to meters per second (1 knot = 0.514444 meters per second). All coordinates and distances were measured in meters. The simulation time step was 30 seconds.

There was exactly one HVA. The HVA's speed was 12 knots. The HVA's sensor range was 18,520 meters. The HVA traveled east to west; its start location was (185200, 24000); its destination was (0, 24000).

The number of searchers was in the set {1, 2, 3, 4}. The searchers' speed was 40 knots. The searchers' sensor range was 3704 meters. There were two possible search policies. In the first search policy, the search space was divided into equal-sized horizontal bands that were parallel to the HVA's path. Each searcher covered one band by moving northwest to the northern edge of the band and then southwest to the southern edge of the band (because the HVA is moving from east to west). The searchers' initial locations were 18,520 meters to the west of the HVA's initial location and on the southern edges of the search band.

In the second search policy, the search is one large band, and the searchers are in a linear formation parallel to the HVA's path. The entire formation covers the search area by moving northwest to the northern edge of the band and then southwest to the southern edge of the band. The first searcher's initial location is 18,520 meters to the west of the HVA's initial location; the remaining searchers are positioned to the west; each one is 3704 meters west of the previous one.

In both cases, the searchers' bearings are set so that they can stay in front of the HVA (that is, their westward motion is not smaller than the HVA's speed) and cover, as much as possible, the search band.

The number of defenders was in the set {5, 6, …, 10}. The defenders' speed was 40 knots. The defenders' sensor range was 1852 meters. The range for a defender's missile was 1000 meters. The defenders, when not moving towards an assigned target, circled positions that were 1852 meters (one nautical mile) ahead of the HVA. The distance between these positions was 4000 meters. The radius of these circles was 1852 meters. Because the HVA is moving, these positions are moving as well, so a defender's path became series of loops. The defenders' initial locations were 1852 meters north of these positions.

The initial locations of attackers and non-combatants were in a box bounded by the SW and NE corners at (18520, 24000) and (166680, 48000). This box was on the north side of the HVA's path.

The number of attackers was in the set {5, 6, …, 10}. The attackers' speed was 20 knots. The attackers' sensor range was 37,040 meters. The attacker must be within 600 meters to attack the HVA successfully. The attackers selected a random "trigger point" along the HVA's path within the box, and their initial locations were north of the HVA's path along an arc around that trigger point. The radius of this arc was the attacker's sensor range. Thus, the attackers simultaneously detected the HVA when it reached the trigger point. Until that time, the attackers waited at their initial position. After they detected the HVA, the attackers moved at full speed to intercept the HVA.

The number of non-combatants was in the set {20, 21, …, 40}. The non-combatants' speed was 10 knots. Each non-combatant's bearing was randomly chosen from the interval [-135, -45] degrees from positive x (east). (Thus, they moved generally to the south between SW to SE.) The non-combatant's initial *x* locations were randomly chosen to be somewhere inside the box.

There were two movement scenarios for the non-combatants; one was chosen at random. In the first scenario, the non-combatant's initial y locations were randomly chosen to be somewhere inside the box, and all non-combatants were moving at the beginning of the simulation. In the second scenario, the non-combatant's initial y locations were the northern edge of the box. For each non-combatant, given its initial location and its bearing, the simulation determined where it would cross the HVA's path, the time that the HVA would arrive at that location, and the time at which the non-combatant should leave its initial location to get to that location at the same time. It then added a randomly determined shift to that start time. The time shift was a value in the interval [-300, 600] seconds.

B. Scenario Simulation

At the beginning of the simulation, all attackers and non-combatants are unknown to the searchers and defenders. Figure 2 shows the tracks of the HVA, attackers, non-combatants, searchers, and defenders for one of the simulation replications; the symbol represents the vehicle's location at the end of the track.

During each time step, the HVA, searchers, attackers, non-combatants, and defenders moved according to their current bearings. For any defender that was assigned to intercept a target, its current bearing was towards the target's location at the beginning of the time step. Otherwise, its bearing was modified so that it circled around the position ahead of the HVA.

In the first non-combatant movement scenario, the non-combatants moved steadily on constant bearings until they left the channel (its y coordinate is less than or equal to 0 meters). In the second non-combatant movement scenario, each non-combatant remained stationary until its start time; only after its start time did it move until it left the channel. In both scenarios, non-combatants that left cannot be detected and do not need to be intercepted.

During a time step, one or more of the following events could occur:

Target detection: If the distance between an undetected attacker or a non-combatant and the HVA is less than or equal to the HVA sensor range, then it is detected. If the distance between an undetected attacker or a non-combatant and any searcher is less than or equal to the searchers' sensor range, then it is detected. Any detected attacker or non-combatant becomes a target. Once it is detected, the target's current location is always known by every defender.

Target interception: If the distance between a target and any defender is less than or equal to the defender's sensor range, then the target is intercepted. The defender then knows if it is an attacker or a non-combatant. An intercepted non-combatant doesn't need to be intercepted again. If the target is an attacker, then the defender will continue to pursue the attacker; when the distance to an attacker is less than the defender's missile range, then the attacker is no longer a threat; it cannot be detected and doesn't need to be intercepted. The defender has expended its missile, it is considered "empty," and it leaves the region of interest and is no longer available for performing tasks.

Successful attack: If distance between an attacker and the HVA is less than or equal to the attack range, the attacker successfully completes its attack. Successful attackers cannot be detected and do not need to be intercepted. A successful attack does not destroy the HVA, which continues towards its destination.

Non-combatant departure: If a non-combatant leaves the region of interest, it cannot be detected and doesn't need to be intercepted.

If no events occurred, then the simulation advanced the clock to the next time step.

If one or more events occurred, the simulation reached a decision point. If there were no targets or no active defenders, then the decision point was skipped, and the simulation advanced the clock to the next time step. Otherwise, a task assignment must be determined. Although the number of successful attacks is the key performance measure for the defenders, at most decision points, no successful attack is imminent (the targets are only moving towards the HVA). Thus, successful attacks cannot be used for making the task assignment decisions. Thus, the metareasoning policy used the performance measures discussed in Section IV.D for making these decisions.

If a metareasoning policy were being used, then the simulation used the surrogate models to estimate the expected performance of each candidate algorithm and selected the algorithm that had the best estimated performance. The selected algorithm is used to assign targets to defenders as follows:

Random allocation: this algorithm randomly perturbs the list of targets and the list of defenders. The algorithm loops over the list of defenders and gives each one the next target while there are still targets to assign. If the number of defenders exceeds the number of targets, some defenders will be assigned no target.

CBAA: this algorithm uses the CBAA algorithm [7] to assign targets to defenders. The positions of the defenders and the targets are scaled by dividing all coordinates by the distance between the HVA's start location and its end location. In the CBAA algorithm, tasks (targets) that are closer to the HVA have more value, and each defender's bid for a task exponentially decreases as its distance to the task increases. Each defender determines which task it should do, receives the other defenders' bids, and determines whether another defender has a higher bid for that task. The defenders iterate through these actions until a consensus assignment is reached. The algorithm assumes that every defender can communicate with all other defenders.

Closest target: this algorithm assigns to each defender the target that is closest to it. Thus, some targets may be assigned to more than one defender.

Performance impact: this algorithm uses the performance impact algorithm [18] to assign targets to defenders. This is initialized with the current assignments. The algorithm maintains a sequence of tasks (targets) for each defender and repeatedly modifies these sequences by removing and adding tasks to lower the total cost of performing every task. The algorithm iterates until no sequence is changed and then assigns each defender the first task in its sequence.

After updating the defenders' assignments, the simulation advanced the clock to the next time step and continued. The simulation ended when the HVA reached its destination. Note that each target is a distinct task that can be assigned to a defender. Any algorithm can change a defender's assigned target at any time; that is, a defender may be assigned a new target before it intercepts its original assignment. Any active defenders that are not assigned a target return to circling.

C. *State Variables*

The metareasoning policy used the following variables to describe the state: (1) the number of defenders still active, (2) the number of targets, (3) the targets' positions relative to HVA, and (4) the defenders' positions relative to HVA. Let n be the number of defenders; let m be the number of targets. Let $\left(x^V, y^V\right)$ be the location of the HVA. Let $\left(x_i^D, y_i^D\right)$ be the location of the i-th defender. Let $\left(x_j^T, y_j^T\right)$ be the location of the j-th target (a task that can be assigned to a defender). Let X be the vector of state variables; Table I lists the expressions for each component.

TABLE I. STATE VARIABLES.

State Variable	Expression
Number of active defenders	$X_1 = n$
Number of targets	$X_2 = m$
Min difference in x direction from target to HVA	$X_3 = \min_{j=1,\ldots,m} x_j^T - x^V$
Mean difference in x direction from target to HVA	$X_4 = \frac{1}{m}\sum_{j=1}^{m} x_j^T - x^V$
Max difference in x direction from target to HVA	$X_5 = \max_{j=1,\ldots,m} x_j^T - x^V$
Min difference in y direction from target to HVA	$X_6 = \min_{j=1,\ldots,m} y_j^T - y^V$
Mean difference in y direction from target to HVA	$X_7 = \frac{1}{m}\sum_{j=1}^{m} y_j^T - y^V$
Max difference in y direction from target to HVA	$X_8 = \max_{j=1,\ldots,m} y_j^T - y^V$
Min difference in x direction from defender to HVA	$X_9 = \min_{i=1,\ldots,n} x_i^D - x^V$
Mean difference in x direction from defender to HVA	$X_{10} = \frac{1}{n}\sum_{i=1}^{n} x_i^D - x^V$
Max difference in x direction from defender to HVA	$X_{11} = \max_{i=1,\ldots,n} x_i^D - x^V$
Min difference in y direction from defender to HVA	$X_{12} = \min_{i=1,\ldots,n} y_i^D - y^V$
Mean difference in y direction from defender to HVA	$X_{13} = \frac{1}{n}\sum_{i=1}^{n} y_i^D - y^V$
Max difference in y direction from defender to HVA	$X_{14} = \max_{i=1,\ldots,n} y_i^D - y^V$

D. *Performance Measures*

For any task (target) assignment, two performance measures were calculated. The first, called "Lmax," was the maximum "lateness" associated with each target. This was normalized by dividing the maximum lateness by the average attacker intercept time. Algorithm 1 (in the appendix) describes the procedure for calculating this. The second, called "Distance," was the average (over the defenders) of the distance to the assigned target. This was normalized by dividing the average distance by the average distance to the closest target. Because the Closest Target algorithm assigned each defender to the closest target, the distance performance measure for the task assignment created by that algorithm was always 1.

E. *Design of Experiments*

For data collection, we ran 100 replications of the simulation. At each decision point, the current state and both performance measures of all four assignments (one from each candidate algorithm) were saved. Then, to create the surrogate models, we used MATLAB's Neural Fitting tool (nftool) to generate a two-layer feed-forward neural network for each algorithm and performance measure (except the Closest Target algorithm and the distance performance measure). This tool used 70% of the data points for training, 15% for validation, and 15% for testing. Each network had 10 hidden neurons, and the tool used the Levenberg-Marquardt algorithm for training. Each resulting network was exported to create a MATLAB function for use in the policy evaluation simulations.

We evaluated forty-six policies, numbered 1 to 46 as follows: (1) random allocation, (2) CBAA, (3) closest target, (4) performance impact, (5-46) metareasoning. Each metareasoning policy included two or four algorithms, an objective function, and a cost function (the combinations are enumerated in Table II). The objective function was either Lmax or distance; this determined which set of neural networks to use for estimating algorithm performance. The cost function was one of three options (shown in Table III); in all three options $c_i(X)$ did not depend upon the state. The values of the cost function were set so that the algorithms that required more computational effort had higher cost values in cost functions 1 and 2.

For policy evaluation, we evaluated all 46 policies using 100 replications of the simulation (these were not the same replications used for data collection). For each replication, the following values were recorded: the number of successful attacks and the number of decision points at which each candidate algorithm was used.

TABLE II. METAREASONING POLICY NUMBERS FOR EACH COMBINATION OF ALGORITHMS, OBJECTIVE, AND COST FUNCTION (RA = RANDOM ALLOCATION, CT = CLOSEST TARGET, PI = PERFORMANCE IMPACT).

				Objective and cost function					
				Rel. Lmax			Rel. Distance		
Algorithm combination				0	1	2	0	1	2
RA	CBAA	CT	PI	5	6	7	8	9	10
RA	CBAA			11	12	13	14	15	16
RA		CT		17	18	19	20	21	22
RA			PI	23	24	25	26	27	28
	CBAA	CT		29	30	31	32	33	34
	CBAA		PI	35	36	37	38	39	40
		CT	PI	41	42	43	44	45	46

TABLE III. COST FUNCTIONS WITH VALUES FOR EACH ALGORITHM (RA = RANDOM ALLOCATION, CT = CLOSEST TARGET, PI = PERFORMANCE IMPACT).

Cost	Algorithm			
function	RA	CBAA	CT	PI
0	0	0	0	0
1	0	0.1	0.1	0.2
2	0	0.3	0.2	0.4

V. RESULTS

The data collection step generated, over 100 simulation runs, 3691 points for each dataset. Of these, 89 points were discarded because they have very large values for the distance performance measure (distance at least 4). This left 3602 points for building the neural networks.

After using MATLAB's Neural Fitting tool, seven neural networks were created. Table IV lists the R values reported by MATLAB for the test data. We also ran 100 simulations to evaluate the distributions of the error of each neural network (error = predicted measure – actual measure). Table IV provides the interval between the 25-th and 75-th percentiles of the distributions of error in relative Lmax and relative distance on 3471 decision points. (No neural network is used to predict relative distance for the Closest Target algorithm; this always equals 1.)

The policy evaluation experiment included 100 replications for each policy. Figure 3 shows, for each policy, the average number of successful attacks and the average computation time per decision point. (The standard deviation for the number of successful attacks ranged from 1.24 to 1.70; this variability is not shown in order to simplify the figure.) Pairwise comparisons showed that the following metareasoning policies' performance was not statistically different from the performance of the CBAA algorithm: 11, 14, 15, and 29 to 33, all of which use the CBAA algorithm and one fast algorithm (policies 11, 14, and 15 use the Random Allocation and CBAA algorithms; policies 29 to 33 use the CBAA and Closest Target algorithms). (See Table II for details of the performance measures and cost functions.)

At each decision point, the average computation time to perform metareasoning (evaluate the surrogate models and select an algorithm) was 0.36×10^{-3} seconds; the average computation time to run the Random Allocation algorithm was 0.10×10^{-3} seconds; the average computation time to run the CBAA algorithm was 0.57×10^{-3} seconds; the average computation time to run the Closest Target algorithm was 0.15×10^{-3} seconds; the average computation time to run the Performance Impact algorithm was 3.60×10^{-3} seconds.

The average computation time per decision point for the metareasoning policies includes the average time needed to perform metareasoning and the weighted average time to run the selected algorithm, which was calculated from data about the relative use of each candidate algorithm (Figure 4).

Figure 4 shows the relative use of each candidate algorithm in each metareasoning policy. It shows that metareasoning policies 11, 14, and 15 used the CBAA algorithm over 80% of the time; metareasoning policies 29, 30, 32, and 33 used the CBAA algorithm over 40% of the time. Metareasoning policy 31, however, used the CBAA algorithm at only 10% of the decision points; it used Closest Target for the remaining 90%. Unlike the other policies, this policy's computational effort was lower than that of the CBAA algorithm.

TABLE IV
R VALUES REPORTED BY MATLAB'S NEURAL FITTING TOOL
AND MIDDLE QUARTILES OF ACCURACY
FOR THE SEVEN NEURAL NETWORKS USING THE DATASETS COLLECTED.

Performance measure	Collaboration algorithm	R	Accuracy (25%, 75%)
Lmax	Random Allocation	0.77	(-0.051, 0.051)
	CBAA	0.89	(-0.028, 0.041)
	Closest Target	0.85	(-0.024, 0.043)
	Performance Impact	0.84	(-0.043, 0.057)
Distance	Random Allocation	0.66	(-0.164, 0.141)
	CBAA	0.88	(-0.067, 0.084)
	Closest Target	-	-
	Performance Impact	0.75	(-0.114, 0.129)

In general, as the cost function increased (changed from 0 to 1 to 2), the metareasoning policies used the more computationally expensive algorithms (CBAA and Performance Impact) less often (see, for example, the metareasoning policies 5-6-7, 8-9-10, 11-12-13, 14-15-16, 23-24-25, 26-27-28).

As shown in Figure 3, over all 46 policies, the performance (average number of successful attacks) did not steadily improve or degrade as the average computation time per decision point increased. Some policies with low average computation time performed poorly (e.g., policies 7 and 13), but others performed well (e.g., policies 20 and 31). The policies with the best performance had average computation time near the median. The policies with the greatest average computation time had poor performance; both characteristics were due to their relatively high use of the Performance Impact algorithm, which was computationally expensive and performed poorly (e.g., policies 23 and 26).

Only metareasoning policy 31 had performance that was equivalent to CBAA and average computation time that was better than CBAA. Metareasoning policies 11, 14, 15, 29, 30, 32, and 33 required more average computation time than CBAA; in these policies, the benefit of occasionally running the faster Random Allocation algorithm or Closest Target algorithm did not compensate for the overhead due to metareasoning.

The system performance of the metareasoning policies that used the Lmax performance measure was not consistently better or worse than that of the metareasoning policies that used the Distance performance measure.

VI. Discussion

These results show that the proposed metareasoning approach can develop a useful metareasoning policy, but the quality of the metareasoning policy (system performance and computation time) varied across the metareasoning policies that were evaluated. Eight metareasoning policies performed as well as the best algorithm (CBAA). Others performed very poorly. Although no metareasoning policy required less computational effort per decision point than the fastest algorithms (the Random Allocation and Closest Target algorithms), some were faster than CBAA, and most were faster than performance impact.

The impact of changing the cost function was mixed. In general, increasing the cost (using cost function 1 or 2) reduced the number of times that the more "expensive" algorithms (CBAA and performance impact) were used; this was true, for instance, for metareasoning policy 31, which used cost function 2 and selected the CBAA algorithm only 10% of the time. For some metareasoning policies (e.g., 11, 12, and 13, which included the random allocation and CBAA algorithms and the Lmax objective), this degraded performance because the CBAA algorithm yielded good performance in these scenarios. For metareasoning policies that included the performance impact algorithm, however, this improved performance because the performance impact algorithm yielded poor performance in these scenarios (e.g., metareasoning policies 41 to 46).

None of the metareasoning policies that used all four algorithms (policies 5 to 10) had performance better than the best metareasoning policies that used only two algorithms. This result was surprising, because a metareasoning policy with more algorithms could be expected to choose even better task assignments at each decision point. Two factors need to be considered, however. First, the inaccuracies in the performance estimates generated by the surrogate models make it possible that, at some decision points, the algorithm that would have yielded the truly best task assignment (measured by Lmax or distance) was not chosen because its estimated performance was poor while another algorithm's estimated performance was better; including another algorithm in the metareasoning policy makes this more likely. This inaccuracy could be reduced by collecting more data to cover more possible scenarios or generating the surrogate models using another approach (instead of using neural networks). Second, the dynamics of the system are complex enough that optimizing Lmax or distance at the current decision point may be a myopic choice. This results from the short time-horizon considered at each decision point; none of the algorithms attempted to optimize the performance across the entire time horizon. The metareasoning approach described here, like those considered in other work, used a myopic estimate of utility [32]. Generating better long-term performance through better short-term decision making in an uncertain environment is a well-known difficult problem [54].

In the scenarios used to evaluate the metareasoning policies, the number of agents and targets was small, so the overhead of using metareasoning required more computational effort than running the fastest algorithms (random allocation and closest target). Thus, although they reduced the number of times that the CBAA algorithm was used, metareasoning policies that used CBAA usually did not reduce the computational effort (metareasoning policy 31 was an exception). This might be different in scenarios with many more agents and targets, where the computational cost of the algorithms is much larger than that of metareasoning; in such cases, the computational savings due to using faster algorithms at the right times would more than compensate for the overhead of using metareasoning, and more metareasoning policies could yield competitive performance with less computational effort. Future work should conduct experiments to explore this possibility.

The results here are limited to the specific scenario that was considered. Although this scenario is related to the CSAT problem, it is not, however, a standard CSAT problem, and the metareasoning policy employed cannot be directly applied to other CSAT problems. Still, the metareasoning approach can be used to generate a metareasoning policy for any class of CSAT problems.

This exploratory research was meant to generate insights into using this metareasoning approach for these types of problems. The results of the experiments described here indicate that it may be possible to generate an effective metareasoning policy with only two candidate algorithms: one good, fast algorithm and one better but more expensive one.

VII. Conclusion

This paper proposed a novel data-driven metareasoning approach for CSAT and similar problems and described two sets of simulation experiments: (1) simulations used to collect data for generating surrogate models and (2) simulations used to evaluate metareasoning policies that used these surrogate models.

The proposed metareasoning approach has four steps: (1) define the state variables and performance measure and identify the candidate algorithms, (2) conduct simulations to collect data on the performance of the candidate algorithms in different states, (3) use this data to train surrogate models that can quickly estimate the performance of the candidate algorithms, and (4) deploy these surrogate models in a metareasoning policy. In our implementation, we used the data to train neural networks as the surrogate models, but other function approximation approaches could be used.

The computational results show that, for a simulated ship protection scenario that is a variant of the CSAT problem, a metareasoning policy constructed using this approach can perform as well as the best candidate algorithm while reducing the computational effort. This metareasoning policy uses one good, fast algorithm and one better but more expensive one.

These results and these insights are a first step towards understanding the utility of the proposed metareasoning approach and the behavior of such metareasoning policies. It would be interesting to apply the approach to a wide variety of CSAT problems, with different candidate algorithms, state variables, and performance measures. Performing an ablation study to determine which state variables are the most valuable would be useful, and additional analysis of the choices made at each decision point could provide insights into the metareasoning policies' performance. It would also be interesting to expand the approach to consider ways to incorporate agent-specific information in the metareasoning policy; with that type of policy, some agents could choose (via the metareasoning policy) one algorithm while the others choose another algorithm. There may be simpler metareasoning policies (based on simple rules, for instance) that perform as well as the data-driven approach that this paper has proposed; such rules could be based on the characteristics of the states in which each collaboration algorithm is used. Other surrogate modeling approaches could be used as well. Finally, approaches in which the surrogate models are learned or updated online could be considered. Although these ideas are beyond the scope of this paper, there could be considerable merit in them.

Appendix

This appendix describes the procedure for calculating Lmax for a given task assignment. It calls the subroutines *getintercept* and *calcposition*, which are also described in this appendix.

Given:

$\{1,\ldots,n\}$: the set of defenders that are assigned a target.

$\{1,\ldots,m\}$: the set of targets (some may be assigned to no defender).

T_i : the target to which defender *i* is assigned, for $i = 1,\ldots,n$.

$\left(x^V, y^V\right)$: the location of the HVA.

$\left(x^W, y^W\right)$: the destination of the HVA.

$\left(x_i^D, y_i^D\right)$ be the location of defender *i*, for $i = 1,\ldots,n$.

$\left(x_j^T, y_j^T\right)$: the location of the *j*-th target, for $j = 1,\ldots,m$.

v_V : the speed of the HVA (meters / second)

v_D : the speed of a defender (meters / second)

v_T : the presumed speed of a target (meters / second)

M: a very large time (e.g., the time needed for the HVA to reach its destination).

Step 1. For $j = 1,\ldots,m$, determine $\left(x_j^A, y_j^A\right)$ and t_j^A , the location and time at which target *j* could attack the HVA (its "due date"), by calling *getintercept*$\left(x^V, y^V, v_V, x^W, y^W, x_j^T, y_j^T, v_T\right)$. If $t_j^A = \infty$, then set $t_j^A = M$. Calculate $\Delta t_j = (x_j^A - x_j^T, y_j^A - y_j^T)$.

Step 2. For $i = 1,\ldots,n$, determine $\left(x_i^I, y_i^I\right)$ and t_i^I , the location and time at which defender *i* could intercept target T_i , by calling *getintercept*$\left(x_{T_i}^T, y_{T_i}^T, v_T, x_{T_i}^A, y_{T_i}^A, x_i^D, y_i^D, v_D\right)$. If $t_i^I = \infty$, then set $t_i^I = M$.

Step 3. For $j = 1,\ldots,m$, determine Φ_j , the set of defenders assigned to target *j*: $\Phi_j = \{i : T_i = j\}$.

Step 4. For all *j* such that Φ_j is not empty, calculate the target's "completion time" $C_j = \min\left\{t_i^I : i \in \Phi_j\right\}$.

Step 4. If there exist any defenders *i* such that $t_i^I < M$, go to Step 5. Else, go to Step 6.

Step 5. For all *j* such that Φ_j is empty, perform Steps 5a and 5b. Then go to Step 7.

Step 5a. For all *i* such that $t_i^I < M$, perform Steps 5a1, 5a2, and 5a3.

Step 5a1. Calculate $\left(x_j^P, y_j^P\right)$, the location of target *j* at time t_i^I, by calling *calcposition*$\left(x_j^T, y_j^T, \Delta t_j, v_T, t_i^I\right)$.

Step 5a2. Calculate $\left(x_j^Q, y_j^Q\right)$, the location of target *j* at time *M*, by calling *calcposition*$\left(x_j^T, y_j^T, \Delta t_j, v_T, M\right)$.

Step 5a3. Determine $\left(x_i^K, y_i^K\right)$ and t_i^K, the location and elapsed time at which defender *i* could intercept target *j* after intercepting its assigned target, by calling *getintercept*$\left(x_j^P, y_j^P, v_T, x_j^Q, y_j^Q, x_i^I, y_i^I, v_D\right)$. If $t_i^K = \infty$, then set $t_i^K = M - t_i^I$.

Step 5b. Calculate the target's "completion time" $C_j = \min\left\{t_i^I + t_i^K : t_i^I < M\right\}$.

Step 6. For all *j* such that Φ_j is empty, set $C_j = M$.

Step 7. Calculate $L_{\max} = \max\limits_{j=1,\ldots,m}\left\{C_j - t_j^A\right\}\cdot m / \sum\limits_{j=1}^{m} t_j^A$

getintercept$\left(x_0, y_0, v_1, x_1, y_1, x_2, y_2, v_2\right)$ returns an intercept position (x_3, y_3) and an intercept time t^*.

Given:

$\left(x_0, y_0\right)$: the target's location

$\left(x_1, y_1\right)$: the target's destination

v_1: speed of the target (meters per second)

$\left(x_2, y_2\right)$: location of the pursuer

v_2: speed of the pursuer (meters per second)

Step 1. Calculate $D = \sqrt{\left(x_1 - x_0\right)^2 + \left(y_1 - y_0\right)^2}$.

Step 2. If $y_1 \le y_0$, calculate $\alpha = \cos^{-1}((x_1 - x_0)/D)$; else, $\alpha = -\cos^{-1}((x_1 - x_0)/D)$.

Step 3. Calculate $\left(\tilde{x}_p, \tilde{y}_p\right) = \begin{bmatrix}\cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha\end{bmatrix}\begin{bmatrix}x_2 - x_0 \\ y_2 - y_0\end{bmatrix}$, the transformed location of the pursuer.

Step 4. Find the roots of the polynomial $\left(v_2^2 - v_1^2\right)t^2 + 2\tilde{x}_p v_1 t - \tilde{x}_p^2 - \tilde{y}_p^2 = 0$.

Step 5. If there are no real roots, then return $t^* = \infty$.

Step 6. If there is only one negative root, then return $t^* = \infty$.

Step 7. Set t^* to equal the smallest positive root and calculate $r = v_1 t^* / D$

Step 8. If $r > 1$, then return $t^* = \infty$.

Step 9. Calculate $\left(x_3, y_3\right) = \left(x_0 + r\left(x_1 - x_0\right), y_0 + r\left(y_1 - y_0\right)\right)$ and return (x_3, y_3) and t^*.

calcposition$\left(x_0, y_0, \Delta, v, t\right)$ returns the location $\left(x_1, y_1\right)$

Given:

$\left(x_0, y_0\right)$: the current location

$\Delta = (\Delta x, \Delta y)$: a vector parallel to the bearing.

v: the speed (meters per second)

t: the time (seconds)

Step 1. Calculate $D = \sqrt{\Delta x^2 + \Delta y^2}$.

Step 2. Return $\left(x_1, y_1\right) = \left(x_0 + vt\Delta x / D, y_0 + vt\Delta y / D\right)$.

Acknowledgment

This research was conducted while the author spent a sabbatical at the Naval Air Warfare Center-Aircraft Division (NAWC-AD) at NAS Patuxent River. The author would like to thank Stephen Kracinovich, Johann Soto, and their colleagues at NAWC-AD for the financial support and the resources used in this research. The anonymous referees also contributed greatly to improving the paper.

References

[1] J. P. How, C. Fraser, K. C. Kulling, and L. F. Bertuccelli, "Increasing autonomy of UAVs," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, 2009.

[2] M. Senanayake, I. Senthooran, J. C. Barca, H. Chung, J. Kamruzzaman, and M. Murshed, "Search and tracking algorithms for swarms of robots: a survey," *Robotics and Autonomous Systems*, vol. 75, pp. 422-434, 2016.

[3] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative robots to observe moving targets: review," *IEEE Transactions on Cybernetics*, vol. 48, no. 1, pp. 187-198, 2018.

[4] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, Artech House, 1999.

[5] L. E. Parker, "ALLIANCE: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots," in *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 94*, vol. 2, 1994, pp. 776-783.

[6] Y. Ding, M. Zhu, Y. He, and J. Jiang, "P-CMOMMT algorithm for the cooperative multi-robot observation of multiple moving targets," in *The Sixth World Congress on Intelligent Control and Automation*, vol. 2, 2006, pp. 9267-9271.

[7] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912-926, 2009.

[8] S. Koenig, P. Keskinocak, and C. A. Tovey, "Progress on agent coordination with cooperative auctions," *AAAI*, vol. 10, pp. 1713–1717, 2010.

[9] M. TJ Spaan, N. Gonçalves, and J. Sequeira, "Multirobot coordination by auctioning POMDPs," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 1446-1451.

[10] M. L. Cummings, J. P. How, A. Whitten, and O. Toupet, "The impact of human–automation collaboration in decentralized multiple unmanned vehicle control," *Proceedings of the IEEE*, vol. 100, no. 3, pp. 660-671, 2012.

[11] S. Hunt, Q. Meng, C. Hinde, and T. Huang, "A consensus-based grouping algorithm for multi-agent cooperative task allocation with complex requirements," *Cogn. Comput*, vol. 6, pp. 338–350, 2014.

[12] E. Raboin, P. Švec, D. S. Nau, and S. K. Gupta, "Model-predictive asset guarding by team of autonomous surface vehicles in environment with civilian boats," *Autonomous Robots*, vol. 38, no. 3, pp. 261–282, 2015.

[13] M. Otte, M. Kuhlman, and D. Sofge, "Competitive target search with multi-agent teams: symmetric and asymmetric communication constraints," *Autonomous Robots*, 2017.

[14] M. Otte, M. Kuhlman, and D. Sofge, "Multi-robot task allocation with auctions in harsh communication environments," in *IEEE International Symposium on Multi-Robot and Multi-Agent Systems*, 2017.

[15] Y. Jin, M. M. Polycarpou, and A. A. Minai, "Cooperative real-time task allocation among groups of UAVs," in *Recent Developments in Cooperative Control and Optimization*. Boston, MA, USA: Kluwer Academic Publishers, 2004.

[16] S. Ganapathy and K. M. Passino, "Distributed agreement strategies for cooperative control: modeling and scalability analysis." in *Recent Developments in Cooperative Control and Optimization*. Boston, MA, USA: Kluwer Academic Publishers, 2004.

[17] V. Singhal and D. Dahiya, "Distributed task allocation in dynamic multi-agent system," in *International Conference on Computing, Communication and Automation*, 2015, pp. 643-648.

[18] W. Zhao, Q. Meng, and P. WH Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Transactions on Cybernetics*, vol. 46, no. 4, April 2016.

[19] A. Whitbrook, Q. Meng, and P. WH Chung, "A robust, distributed task allocation algorithm for time-critical, multi-agent systems operating in uncertain environments," in *Proceedings, 30th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Arras, France, 2017.

[20] A. Sinha, T. Kirubarajan, and Y. Bar-Shalom, "Autonomous ground target tracking by multiple cooperative UAVs," in *Aerospace Conference*, 2005, pp. 1-9.

[21] G. M. Hoffmann and C. J. Tomlin, "Mobile sensor network control using mutual information methods and particle filters," *IEEE Transactions on Automatic Control,* vol. 55, no. 1, pp. 32-47, 2010.

[22] M. Stachura and E. Frew, "Communication-aware information-gathering experiments with an unmanned aircraft system," *Journal of Field Robotics*, vol. 34, no. 4, pp. 736-756, 2017

[23] L. E. Parker and B. A. Emmons, "Cooperative multi-robot observation of multiple moving targets," in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, vol. 3, 1997, pp. 2082-2089.

[24] N. Sydney, D. A. Paley, and D. Sofge, "Physics-inspired motion planning for information-theoretic target detection using multiple aerial robots," *Autonomous Robots*, vol. 41, no. 1, pp. 231-241, 2017.

[25] F. Wu, S. Zilberstein, and X. Chen, "Multi-agent online planning with communication," in *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 2009, pp. 321-328.

[26] F. Wu, S. Zilberstein, and X. Chen, "Online planning for multi-agent systems with bounded communication," *Artificial Intelligence*, vol. 175, pp. 487-511, 2011.

[27] S. Seuken and S. Zilberstein, "Formal models and algorithms for decentralized decision making under uncertainty," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190-250, 2008.

[28] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, "Coordinated multi-robot exploration under communication constraints using decentralized Markov decision processes," *AAAI*, 2012.

[29] D. K. Ahner, "Real-time planning and control of army UAVs under uncertainty," *Journal of Aerospace Computing, Information, and Communication*, vol. 4, no. 5, pp. 798-815, 2007.

[30] H. Terelius, U. Topcu, and R. M. Murray, "Decentralized multi-agent optimization via dual decomposition," in *Proceedings of the 18th World Congress, The International Federation of Automatic Control*, 2011, pp. 11245-11251.

[31] M. T. Hale, A. Nedic, and M. Egerstedt, "Cloud-based centralized/decentralized multi-agent optimization with communication delays," in *IEEE 54th Annual Conference on Decision and Control*, 2015, pp. 700-705.

[32] M.T. Cox, "Metacognition in computation: A selected research review," *Artificial Intelligence*, vol. 169, no. 2, pp. 104-141, 2005.

[33] M. Cox and A. Raja, "Metareasoning: an introduction," in *Metareasoning: Thinking about Thinking*. Cambridge, MA, USA: MIT Press, 2011.

[34] S. Russell and E. Wefald, *Do the Right Thing.* Cambridge, MA, USA: The MIT Press, 1991.

[35] M. L. Anderson and T. Oates, "A review of recent research in metareasoning and metalearning," *AI Magazine*, vol. 28, no. 1, p. 12, 2007.

[36] S. Milli, F. Lieder, and T. L. Griffiths, "When does bounded-optimal metareasoning favor few cognitive systems?" *AAAI*, pp. 4422-4428, 2017.

[37] P. M. Krueger, F. Lieder, and T. L. Griffiths, "Enhancing metacognitive reinforcement learning using reward structures and feedback." in *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*, Austin, TX, USA, 2017.

[38] V. Conitzer, "Metareasoning as a formal computational problem," in *Metareasoning: Thinking about Thinking*, M.T. Cox and A. Raja, eds., pp. 119-128, MIT Press, Cambridge, Massachusetts, 2011.

[39] J. R. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65-118, 1976.

[40] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, vol. 41, no. 1, p. 6, 2009.

[41] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Empirical hardness models: methodology and a case study on combinatorial auctions," *Journal of the ACM*, vol. 56, no. 4, p. 22, 2009.

[42] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Identifying key algorithm parameters and instance features using forward selection," in *International Conference on Learning and Intelligent Optimization*, 2013, pp. 364-381.

[43] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "A meta-learning prediction model of algorithm performance for continuous optimization problems," in *International Conference on Parallel Problem Solving from Nature*, pp. 226-235.

[44] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: a survey on methods and challenges," *Information Sciences*, vol. 317, pp. 224-245, 2015.

[45] L. Kotthoff, "Algorithm selection for combinatorial search problems: a survey," in *Data Mining and Constraint Programming*. Springer, 2016, pp. 149-190.

[46] S. Zilberstein, Operational rationality through compilation of anytime algorithms. Ph.D. Dissertation, University of California at Berkeley, 1993.

[47] S. Zilberstein and S.J. Russell, "Optimal composition of real-time systems," *Artificial Intelligence*, vol. 82, no. 1–2, pp. 181–213, 1996.

[48] A. Raja and V. Lesser, "A framework for meta-level control in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 15, no. 2, pp. 147-196, 2007.

[49] G. Alexander, A. Raja, E. H. Durfee, and D. J. Musliner, "Design paradigms for meta-control in multi-agent systems," in *Proceedings of AAMAS 2007 Workshop on Metareasoning in Agent-based Systems*, 2007, pp. 92-103.

[50] J. Sleight and E. H. Durfee, "Multiagent metareasoning through organizational design," in *Proceedings of the 2014*

International Conference on Autonomous Agents and Multi-agent Systems, 2014, pp. 1579-1580.

[51] S. Cheng, A. Raja, and V. Lesser, "Multiagent meta-level control for radar coordination," *Web Intelligence and Agent Systems: An International Journal*, vol. 11, no. 1, pp. 81-105, 2013.

[52] R. Kota, N. Gibbins, and N. R. Jennings, "Decentralized approaches for self-adaptation in agent organizations," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 1, p. 1, 2012.

[53] S. Zilberstein, "Metareasoning and bounded rationality," in *Metareasoning: Thinking about Thinking.* Cambridge, MA, USA: MIT Press, 2011.

[54] W.B. Powell. A Unified Framework for Optimization Under Uncertainty. In INFORMS Tutorials in Operations Research. Published online: 04 Nov 2016; 45-83. https://doi.org/10.1287/educ.2016.0149

Jeffrey W. Herrmann received the B.S. degree in applied mathematics from Georgia Institute of Technology, Atlanta, Georgia, USA, in 1990, and the Ph.D. degree in industrial and systems engineering from the University of Florida, Gainesville, Florida, USA, in 1993.

He is currently a Professor at the University of Maryland in College Park, Maryland, USA, where he holds a joint appointment in the Department of Mechanical Engineering and Institute for Systems Research. He is the author of the textbook *Engineering Decision Making and Risk Management* (Wiley, 2015). His current research interests include collaborative search and tracking and engineering design decision making.

Dr. Herrmann is a member of ASEE, IISE, ASME, the Design Society, and INFORMS.

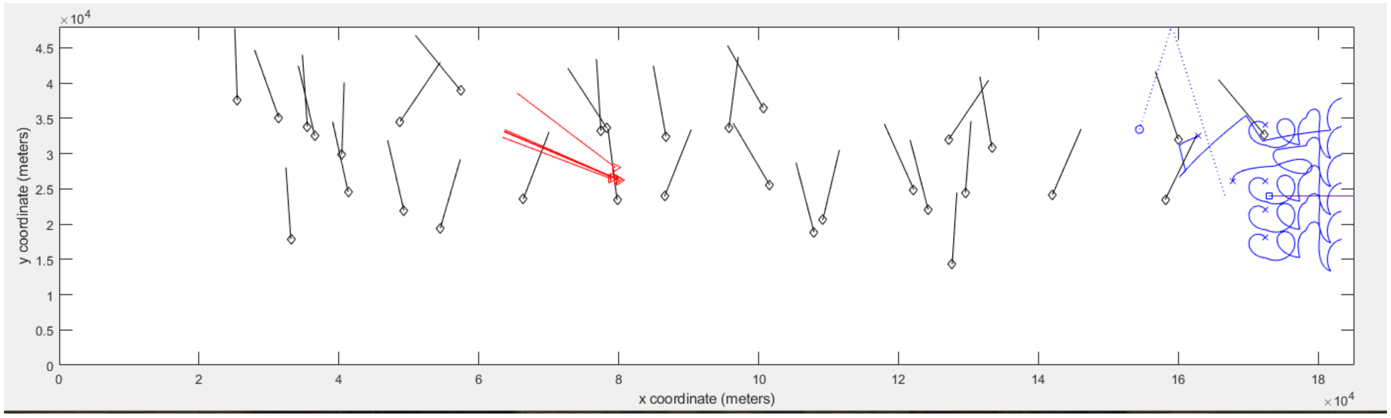

Fig. 2. Tracks showing the movement of the HVA, searchers, defenders, attackers, and non-combatants in one of the simulation replications. Each track covers the same amount of simulated time.
Blue square: HVA. Blue circle: searcher. Blue x: defender. Red triangle: attacker. Black diamond: non-combatant.

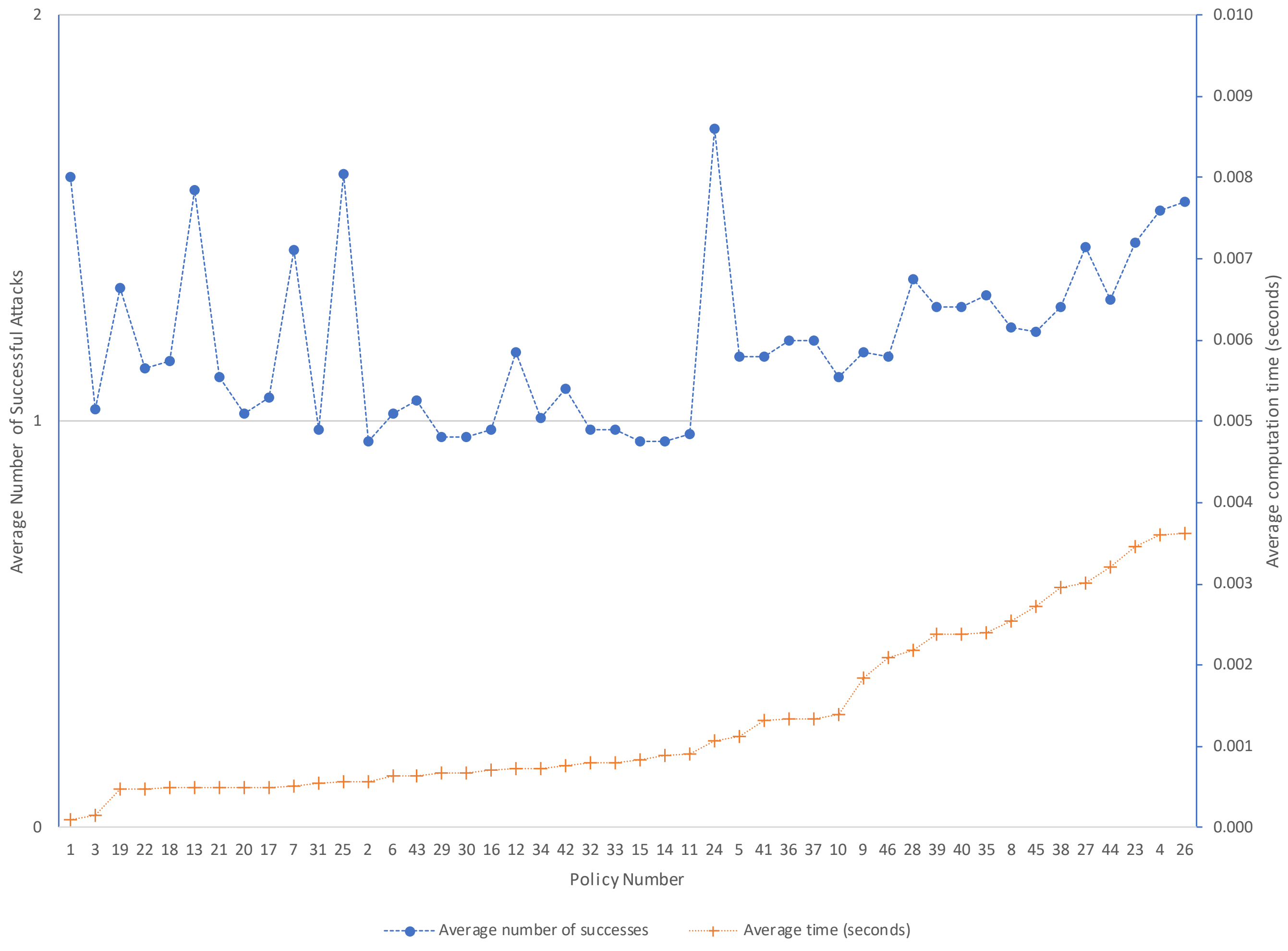

Fig. 3. The average number of successful attacks and the average computation time per decision for each policy.
1 = Random Allocation, 2 = CBAA, 3 = Closest Target, 4 = Performance Impact.
See Table II for definition of each metareasoning policy (5 to 46).

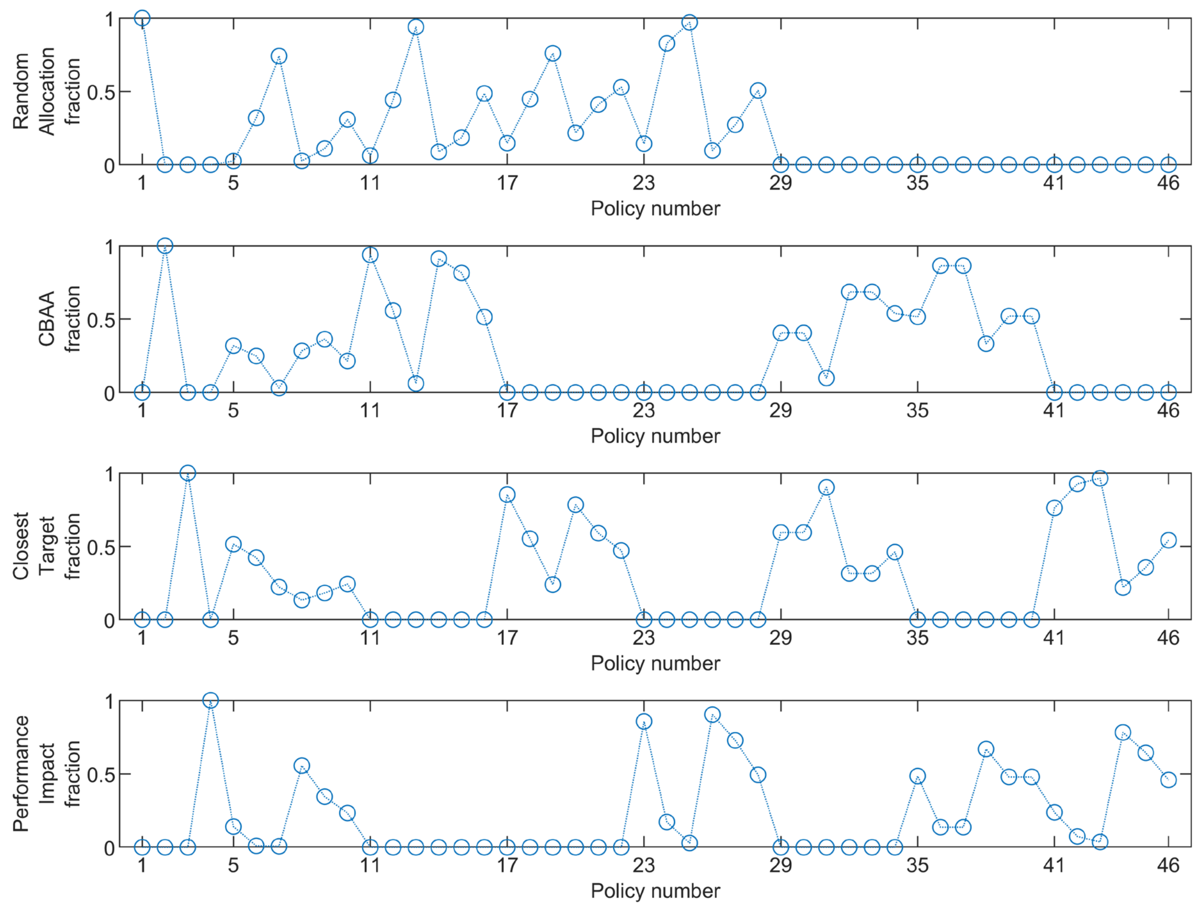

Fig. 4. The relative use of each candidate algorithm in each policy.
1 = Random Allocation, 2 = CBAA, 3 = Closest Target, 4 = Performance Impact.
See Table II for definition of each metareasoning policy (5 to 46).