The Environment Value of an Opponent Model

Brett J. Borghetti

***Abstract*—We develop an upper bound for the potential performance improvement of an agent using a best response to a model of an opponent instead of an uninformed game-theoretic equilibrium strategy. We show that the bound is a function of only the domain structure of an adversarial environment and does not depend on the actual actors in the environment. This bounds-finding technique will enable system designers to determine if and what type of opponent models would be profitable in a given adversarial environment. It also gives them a baseline value with which to compare performance of instantiated opponent models. We study this method in two domains: selecting intelligence collection priorities for convoy defense and determining the value of predicting enemy decisions in a simplified war game.**

***Index Terms*—Equilibrium, game theory, multiagent systems, opponent model.**

I. Introduction

SUPPOSE that we are planning to send a convoy through dangerous territory in a hostile area. There is a chance that we will encounter an ambush. We would like to minimize our risk while ensuring that the cargo is delivered. We have two roads to choose from—road A, which is the straightest shortest distance to the destination, and road B, which is significantly longer but still leads to the destination. Because of the time sensitivity of the cargo, the travel time to the destination is important. Based on our estimates, the enemy may have planned up to two ambushes, but we are not sure whether they have planned any on road A, road B, or both. We also know based on a probabilistic analysis of past events that each ambush succeeds with probability p.

If we had access to intelligence sources, which could reveal more information about the enemy's activities, we would be able to plan better our convoy. Fortunately, we do have access to two potential intelligence sources: capability X, which we could use to determine the number of teams the enemy has that it could use for ambushes (0, 1, or 2), and capability Y, which could persistently observe road A and report if and when the enemy sets up an ambush on that road (road B is far too long to get complete information on). Each of these capabilities is fairly accurate (although not perfect). Unfortunately, due to high demand, we have only been authorized to use one of these intelligence capabilities. Our goal is to choose which intelligence capability (X or Y) we want to use. This paper describes a computational method for determining answers to such questions.

Borrowing language from intelligent agents, we can define the enemy as an *agent*, the number of ambush teams that they have available as their *state*, and the locations (road A or B) that they plan to ambush as their *action*. We define an agent's *policy* as a function that maps its states to actions

$$P : S \rightarrow A$$

where S is the set of all possible states, and A is the set of all possible actions.

An agent *model* is a function that predicts something about the agent. One class of models predicts the likelihood of each action that the agent might take. The model takes a subset of the features of a state as input and provides a probability distribution over possible actions, i.e.,

$$M : \mathbf{wa}$$

where $\mathbf{a}$ is a set of possible actions in the state, and $\mathbf{w}$ is a probability distribution over n actions such that $\sum_{i=1}^{n} w_i = 1$.

Another type of a model predicts the likelihood of the agent being in each state that it could be in, i.e.,

$$M : \mathbf{ws}$$

where $\mathbf{w}$ is a probability distribution over m states $\mathbf{s}$, and $\sum_{i=1}^{m} w_i = 1$.

Since intelligence capabilities X and Y both have the potential to reveal information about the enemy's activities, the information they provide was as if it was given by opponent models. Since intelligence capability X provides (a probability distribution over) the number of ambush teams that the enemy has, it predicts the enemy's state, i.e.,

$$X : \mathbf{ws}.$$

Y is equivalent to a model that predicts the (probability distribution over) likelihood of an ambush team residing on road A; it (partially) predicts the enemy's action, i.e.,

$$Y : \mathbf{wa}.$$

In agent-based literature, the active entities are the agents, and the world in which they perceive and take action is sometimes called the environment. Mathematically, the environment is a function that maps states (of the world) and actions (of the agents) to successor states, i.e.,

$$Env : S_t \times \mathbf{A_t} \rightarrow S_{t+1}.$$

Note that environments are often nondeterministic: The mapping from states and actions to successor states could be

Manuscript received December 21, 2008; revised July 29, 2009. First published November 3, 2009; current version published June 16, 2010. The views expressed in this paper are those of the author and do not reflect the official policy or position of the U.S. Air Force, the Department of Defense, or the U.S. Government. This paper was recommended by Associate Editor T. Vasilakos.

The author is with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Dayton, OH 45433 USA (e-mail: brett.borghetti@afit.edu).

Digital Object Identifier 10.1109/TSMCB.2009.2033703

described by a probability transition table instead of a set of simple if–then rules.

In this example, there are two models of our opponents: X and Y. When considering which of several candidate opponent models or classes of models to install in an agent that is trying to decide which road to take in the convoy, we would like to be able to compare models, which requires some comparison of the value of the information that they could provide.

The best model is one that makes the best predictions (where "best" is determined by some measurement of prediction accuracy). We define a *class* of opponent models as a set of models that have the same type of output. While we can relatively easily compare the performance of the members of the class in a given environment against a specific opponent, the comparison does not provide how relevant the class of models is to the problem that the agent is trying to solve. If the agent developer wishes to know which class of models to select, we need a different measurement.

Ideally, the agent developer would like to know the value of a candidate model class in an environment. The value is the differential in performance between an agent with a perfect model in the class and an agent without the class model. In other words, we would like to know the performance that could be obtained for the agent if the opponent model was perfect at predicting the opponent and the agent was able to use the information that the model provided perfectly, in comparison to the performance of the agent if the information that the model provided was unavailable. We define the *environment value* of a class of opponent models as the value of the information that the class of models could provide in the particular environment.

We will show that the value of an opponent model class is actually invariant of the agent or his opponent and depends only on the structure of the environment. Thus, given an environment and a class of opponent models, we show techniques for determining the value of the information that is provided by the class—a value that is independent of agent, opponent, or the inner workings of any particular model in the class.

The remainder of this paper provides techniques for calculating the potential value of an opponent model class within a certain environment using tools from game theory. The environment value of a model is expressed in terms of the minimum guaranteed expected payoff that is obtainable if the opponent model was perfect. We introduce the concept of an *opponent-model oracle* to represent a perfect (but most likely unattainable) computational model of some aspect of the opponent. In a two-agent environment, we show how the minimum guaranteed expected value can be computed using an environment transformation and computational tools from game theory.

II. Related Work

Our goal is to determine the value of a model in an environment. Since a model is an uncertainty-reduction tool, we are attempting to determine how much additional value can be obtained from the environment if the uncertainty is reduced.

In game-theory parlance, agents are equivalent to players, and the environment previously described is equivalent to the structure of the game. If the environment allows only one simultaneous interaction between the agents, then the environment can be depicted as a normal-form (strategic) game. Environments that allow more than one action per agent or those that allow the agents to take turns are often represented as extensive-form games.

In a normal-form game, there are no states. Each player simply takes an action; the outcome is determined, and the payoffs are awarded according to the structure of the game.

In an extensive-form game (often depicted in a tree form), there are nodes and edges. Nodes represent states in which a player or an agent could reside, and edges represent decisions or actions that the player could make. A pure strategy is a decision of what action to take for each state that the player could be in. A mixed strategy can be thought of as a probability distribution over actions that the agent could take in each state. Thus, a pure strategy represents the set of chosen edges for the agent, and a mixed strategy is a probability distribution over the edges. Chance nodes are states where an action is stochastically determined (via a dice roll or drawing a card, for example). Chance nodes are often thought of as being decided by another player, which is often referred to as Nature.

In some cases, an agent may not be able to discern which of several states he is actually in. This may be due to the other players having the option to simultaneously make their choices of actions (in game-theory parlance) or could be the result of having only partial observability of the current state of the world (in agent parlance). When an agent cannot discern which of several states he is in, we say that the game has imperfect information, and we label the set of states that appear equivalent as an information set.

Both game-theory and agent-based systems share the concept of a best response [1]. Given that the agent finds itself in a particular state or at a particular node in an extensive-form game, there is a (probability distribution over) action(s) that will yield the highest payoff or utility for the agent.

Let us consider the nature of the opponent for a moment. If our opponent was rational (he makes decisions to maximize his own utility given his beliefs [2]), then he would make a choice (or probability distribution over choices) that was in his own best interest at every point where he had to make a decision, given his beliefs about the way the decisions would affect his utility. Furthermore, if an agent and his opponent in a two-player environment are both rational and know the rules of the game and the available payoffs for all outcomes, then game theory describes the intersection of their joint choices (their strategies) as Nash equilibrium [1]–[3]. In equilibrium, neither player can improve his utility by unilaterally altering his strategy (probability distribution over actions); thus, the payoff for a player at Nash equilibrium constitutes a lower bound on the score that a player could achieve when both players select their strategic element of the equilibrium. They could earn more, but they could not earn less as long as they do not deviate from their equilibrium strategy.

There are several forms of tree search (for agent-based approaches) or equilibrium finding (for game-theoretic approaches) that yield the sequence of actions where each player is trying to do their best. If there are only two players in the

environment, and we assume that one player's gains are another player's losses, then we focus on a specific class of games known as zero-sum games. One of the key techniques for finding equilibrium in two-player zero-sum games is the minimax algorithm [4]. Several versions of minimax exist for multiplayer games, such as Luckhardt and Irani's $\max^n$ algorithm [5], and there are probabilistic versions that even work when the game has stochastic elements. When computationally feasible, these algorithms yield a solution for the game, which consists of a strategy for each player (a probability distribution over actions for every node where they make a decision) and the value of the game for each player at equilibrium.

In the methods discussed above, equilibrium is generally calculated assuming that the players are uninformed (they know nothing about which node they are in in any given information set) and unboundedly rational (they have unlimited ability to compute the best response). If there is imperfect information that is present in the system (as depicted by nonsingleton information sets), then there is an opportunity for a player with a nonsingleton information set to become better informed about his location within the information set and make better decisions than he would if he would remain uninformed. This change in knowledge can alter the equilibrium point for all of the players, implying a change in a value for each player.

Models are sometimes used in agent-based systems to discern some aspect of the environment or predict the behavior of another agent. In game-theoretic terms, a model can be used to generate a probability distribution over being in each node of an information set. If the model is accurate, an agent using it does not need to consider computations on information set nodes with zero probability and can instead focus on the nodes with positive probability of occurrence. In many algorithms, this pruning of possibilities can lead to a much more accurate characterization of what state an agent is in or how it will behave. Agent models have been successfully used in augmenting standard game-theoretic equilibrium-finding algorithms. One example is Carmel and Markovich's M^* algorithm—an extension of minimax that incorporates opponent models [6].

The majority of the literature on opponent models discusses structure and architecture, model learning methods, and model use. Although they present methods for evaluating the performance of a specific model (and often compare performance of two or more models), there is no work that discusses the bounds on how well any particular model could perform if it were perfect. This paper focuses on filling that research gap.

III. Approach

In the remainder of this paper, we show how we can compute a static minimum bound on the expected value of an oracle (a perfect opponent model) in a specific environment. The oracle only makes predictions about one player, which we refer to as the *exposed* player. Both players know of the oracle's existence and purpose and which player is being exposed. Both players can see what information the oracle is revealing. Thus, the exposed player knows what information the oracle is revealing about him. We also show that the environment value (the amount of score improvement that is earned in the game, for example) that could be obtained by using an opponent model is a function of the environment itself. We do this using a three-step process.

1) Form an oracle that provides the information that a perfect model would yield.
2) Transform the environment into a new environment where the oracle's information is part of the environment. In game-theoretic terms, this step is equivalent to repartitioning the nodes in an information set of an extensive-form game.
3) Solve for the environment value of the model using game-theoretic techniques such as computing the Nash equilibrium [1], [3].

Each of the steps will be explained in detail in the following sections. If we perform this task for each type of an opponent model of interest, we can discover the bounds and use the information to select promising models to develop for any particular environment.

A. Oracles

An *oracle* is a function that makes *correct* predictions about the modeled agent. The oracle is a hypothetical perfect model that is solely used for the purpose of analyzing agent interaction in a specific environment. Thus, it has certain inputs and certain outputs; however, how it works is irrelevant. It is perfect in the sense that it will always correctly predict some aspect of the modeled agent given the required inputs.

To find performance bounds that are attainable by an oracle in an environment, we transform the environment using a technique that simulates having an oracle of that type, and then we determine a minimum bound on how well we could have performed in the transformed environment.

B. Oracle Simulation Through Environment Transformation

For the analysis in this paper, we focus on oracles that take otherwise hidden information and make it public information. To determine the value of an oracle that reveals some of the imperfect information in an environment Γ, we derive a new environment Γ' where we have access to that information as part of the environment. In other words, the predictions that are made by the oracle are revealed to our agent in the transformed environment, and the actions our agent takes will be able to use the information. We can then calculate the value of a strategy in the transformed environment in relation to the original game.

Since we are planning to do actual calculations in the game Γ', we will need to access its game tree τ', and we will need to instantiate a device that reveals information about the game state. This process is equivalent to repartitioning the information sets in the game such that the oracles' information is revealed to the player prior to the player having to make the decision.

C. Finding the Value Bounds With Game Theory

Here, we describe how to find the environment value of the opponent model in the original environment by finding the

agent's expected value in the transformed environment and subtracting the agent's expected value in the original environment. At the core of this step is the method to be used for finding the expected value—in essence, we need to evaluate the solution from the agent's perspective in both environments. In the remainder of this section, we motivate the use of game-theoretic tools for finding this solution and discuss the implications of using those tools.

There is a strong motivation for using solution techniques that yield Nash equilibrium to find strategies that could be evaluated to determine the lower bound of utility in an environment. To use game-theoretic tools to solve an environment, we must make an assumption that the Nash equilibrium of the game can be found. This assumption is required to ensure the tractability of the technique for finding the solution on the computational hardware available. Thus, some games such as chess and heads-up limit Texas Hold'em Poker would not be suitable targets for this form of analysis.

One of the simplest and most straightforward implementations of equilibrium calculation is the minimax algorithm [4]. If we assume that our opponent is a game-theoretic equilibrium player, then we could assign some form of the minimax (or a probabilistic-capable extension of the minimax) algorithm to represent the policy of our hypothetical opponent. Thus, we could find an optimal set of actions in the new environment by providing a function that maximizes the utility of our decisions in the new environment. By assessing the expected values of the new environment when playing our optimal strategy against the minimax player opponent, we can obtain the minimum bound for the expected value in the new environment. Performing the same calculation in the old environment, we could obtain the minimum expected value in the old environment. Subtracting the original value from the new value would yield the environment value of the model, i.e.,

$$V_\Gamma(M) = U(M \mid \Gamma') - U(M \mid \Gamma).$$

IV. Results

Next, we examine two case studies to demonstrate how to find the value of a model in an environment: the ambushed convoy scenario and the analysis of opponent models in a simultaneous-move strategy game.

A. Case Study: The Ambushed Convoy Scenario

In the convoy scenario, as described in Section I, our goal is to determine which of two intelligence sources is preferable for a given instance of the problem. First, we formally define the game. It is a two-player zero-sum game (we assume that the ambusher's gain is equivalent to the convoy's loss in this example). Player 1 is the convoy driver. The driver must choose whether to travel road A or road B. If he succeeds without being ambushed on road A, he obtains a payoff of 1. If he travels road B and succeeds without being ambushed, he obtains a payoff of 1/2 (due to the extra time required to travel and the time sensitivity of the cargo). If he gets ambushed on any road, then his payoff is -1. Being a zero-sum game, the ambusher receives the negative of each of these payoffs at the outcome of the game. The ambusher is given 0, 1, or 2 ambushing teams (selected by chance), and his action is in determining where to place the teams. If he has 0 teams, he essentially has no choice. If he has received one team, he may choose to place the team on road A or road B. If he has received two teams, he may choose to split them and cover both roads, or he may double up his forces on either road A or B. The base probability that an ambush team will succeed when the convoy travels the road where the ambush team is hiding is p. If there are two ambush teams on a single road when the convoy traverses the road, the probability of both ambush teams failing is $(1-p)^2$; thus, the probability of at least one successful ambush is $1-(1-p)^2$. The probability chosen for p defines a problem instance of this game and has an influence over which type of intelligence collection is preferred.

In the uninformed version of the game, the convoy player has no information about the number of teams the ambusher has, and he has no information regarding which road(s) the ambusher has selected. All of the convoy player's nodes are in a single information set. This situation is depicted using an extensive-form game tree in Fig. 1. By setting the probability distribution over the number of teams that the ambusher receives and the likelihood that a team will succeed if it attacks a convoy, we can compute the value of the game for the convoy driver. When we make an assumption of equal probability that the number of teams the ambusher has is 0, 1, or 2, and we set the likelihood of each ambush team succeeding in the ambush to $p = 50\%$, we obtain a value of 0.300 for the convoy driver when both players play the Nash equilibrium strategy (as computed by Gambit [7]).

If the convoy player had an opponent model that could predict the opponent's state (how many ambush teams he had available?) or part of his action (did the ambusher place a team on road A?), the convoy player could make a better decision about which road to travel on. If the opponent model in either of these two cases was an oracle (i.e., a perfect opponent model), the effect would be a transformation of the game. The transformed game can be created by redistributing the convoy player's decision nodes into new information sets. Since the convoy player would have more information, his value of the game when both players play a Nash equilibrium strategy will be at least as high (and probably higher) than when he was uninformed. To calculate the value of the information that is provided by the model in this environment (the environment value), we use $V_\Gamma(M) = U(M \mid \Gamma') - U(M \mid \Gamma)$, where Γ is the original game, and Γ' is the transformed game.

Fig. 2 depicts a transformation of the game in which the convoy player perfectly knows whether there is an ambush on road A. In this case, the value of the game for the convoy driver is 0.416; thus, the environment value of knowing whether the ambusher has placed a team on road A is $0.416 - 0.300 = 0.116$.

Another transformation of the original game in which the convoy player perfectly knows the number of ambush teams that the enemy has available is shown in Fig. 3. The value of the game for the convoy driver when he knows how many teams

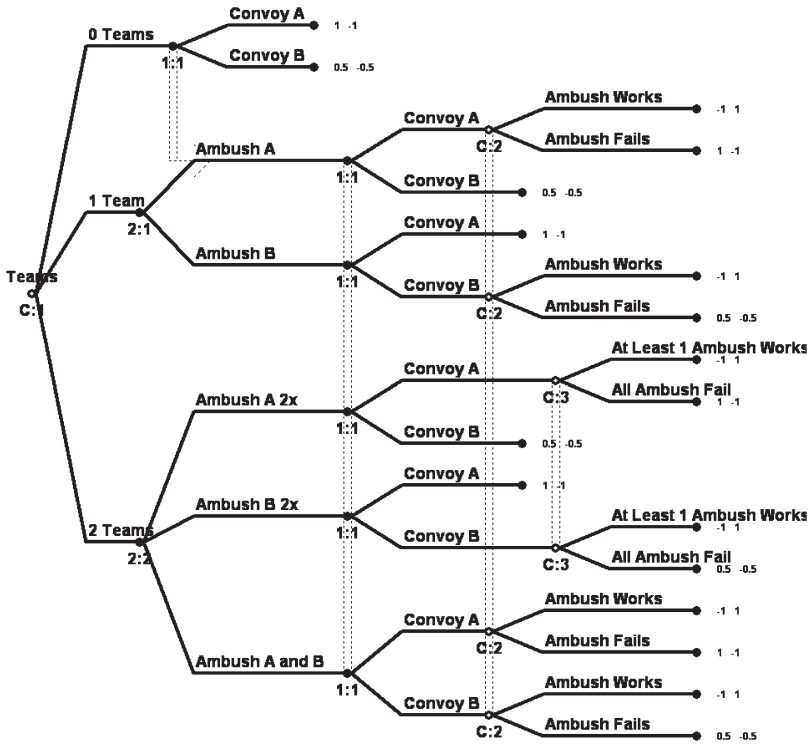


Fig. 1. Ambushed–convoy scenario with no intelligence information. (Figure produced with Gambit software [7].)

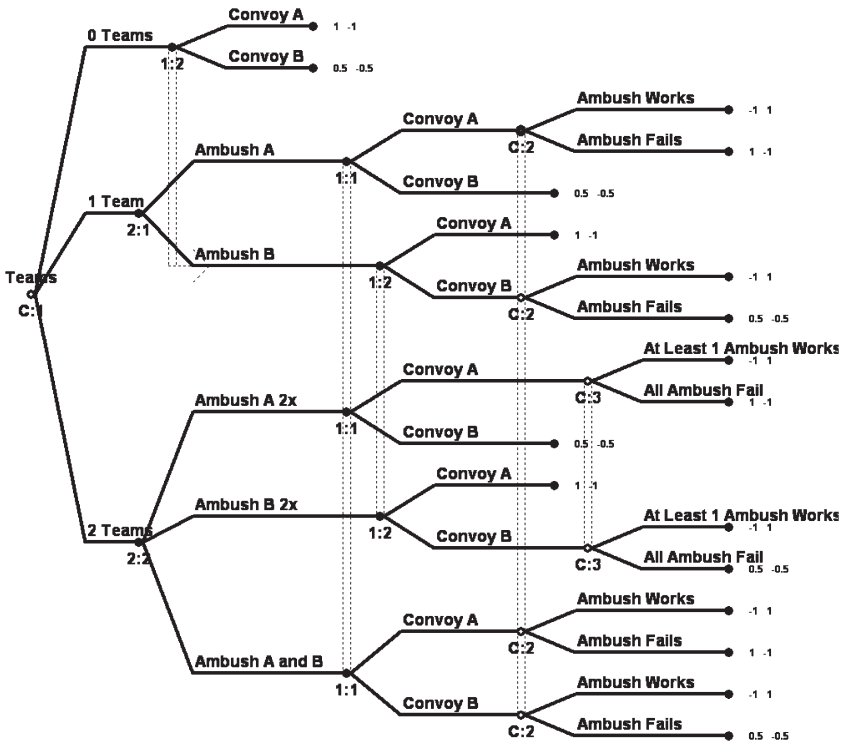


Fig. 2. Ambushed–convoy scenario when a convoy player knows whether there is an ambush waiting on road A. (Figure produced with Gambit software [7].)

the ambusher has available is 0.395, and, thus, the environment value of knowing the number of teams is $0.395 - 0.300 = 0.095$.

Since a model of the opponent that can perfectly predict whether the opponent has placed at least one ambush team on road A (the best possible performance of intelligence capability Y) has an environment value of 0.116, and an opponent model that can perfectly predict the number of teams available to the ambusher (the best possible performance of intelligence capability X) has an environment value of 0.095, the convoy driver would prefer intelligence capability Y to intelligence capability X.

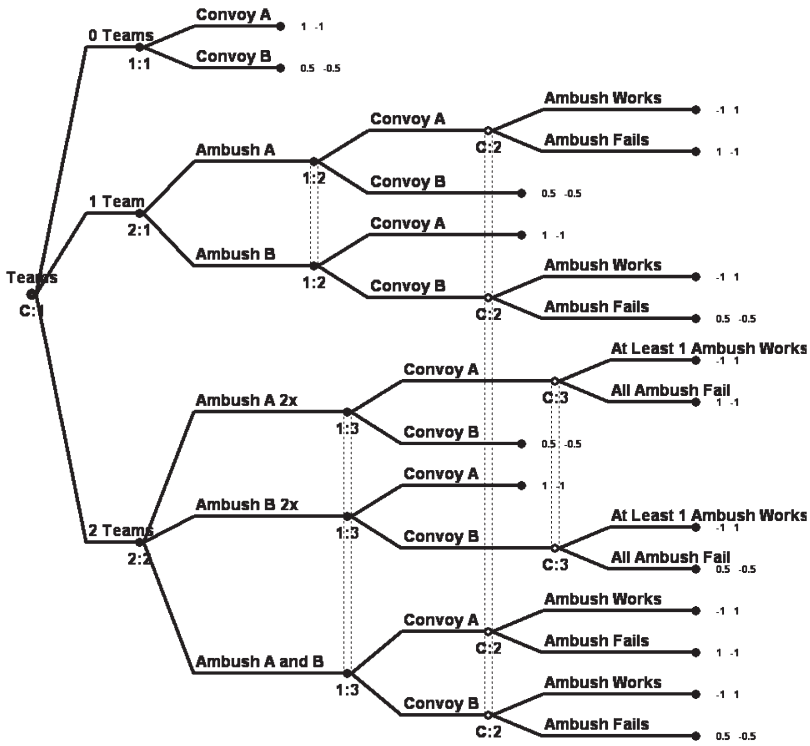


Fig. 3. Ambushed–convoy scenario with the convoy having knowledge of how many ambush teams are available. (Figure produced with Gambit software [7].)

As expected, the value of the two intelligence capabilities (if perfect) is dependent on the environment. For example, if we altered the probability of success of the ambush teams from 50% to 33%, the uninformed convoy driver would obtain a value of 0.451 due to the higher probability of slipping past the ambush attempt. Armed with intelligence capability X, the convoy driver knows the number of teams and can make better informed decisions, yielding a value of 0.535; however, knowing whether the enemy has planted an ambush on road A (using intelligence capability Y) is relatively worse in this environment and yields a value of only 0.529. In this environment, intelligence capability X has an environment value of 0.084, and Y has an environment value of 0.078, prompting the convoy driver to prefer X to Y.

B. Case Study: Environment Value in the Simultaneous-Move Strategy Game

We now apply our value-bounds-finding technique to a much richer environment: the simultaneous-move strategy war game described in the Appendix. Whereas the convoy ambush game was a finite extensive-form zero-sum game, the game about to be evaluated is an infinite extensive-form zero-sum game with perfect information about the state of the game available to both players.

In the version of the game that we use for this case study, there is no hidden information; thus, the state of the game is already known, and an oracle that reveals the opponent's state is unnecessary. Since by definition, knowing the policy of a Nash equilibrium player has no benefit for its opponent, we do not need to assess the value of the policy oracle either. For these experiments, we solely focus on the value of the action oracle. Essentially, we are trying to answer the following question: If I could purchase information about what action my opponent is about to take before he takes that action, how much is that information worth?

If we label the player whose actions are being revealed as the exposed player and the other player as the protected player, we can answer this question with the following process.

1) Determine the baseline value of each state when each player is playing the simultaneous-move game, and both player's policies are the mixed Nash equilibrium strategies for the game.
2) Determine the value of the game for the protected player when he knows what action the exposed player is going to take just before the protected player decides what action to take.
3) Subtract the protected player's value of the baseline game from the value they would achieve in the transformed game. The result is the value of the action-revealing oracle: It is the environment value of the opponent model that reveals action.

Since this game is a multiple-turn game, there are several ways to determine the value of opponent-action-revealing information. Perhaps the most intuitive way is to assume that the oracle would be available in this turn and all turns thereafter. Thus, from any state in the game, the oracle would reveal the exposed player's next action. We also make the assumption for the purposes of this analysis that the exposed player will play a stationary optimal strategy. Given a state, there is only one action that the opponent would choose, and he would choose that action every time he was in that state, independent of the history of play prior to arrival in that state.

Since the successor state is partially stochastic (often based on the outcome of the stochastic events that occur during the game phases following the selection of each player's actions), we can form a set of outcomes that could occur when the opponent takes an action and we take an action. Once each player selects an action, the outcome states depend on the transition probabilities that are solely a function of the rules of the game. By weighting the likelihood of an outcome with the transition probabilities, we can determine the expected value for a state from the value of the successor states. We use the same iterative solution method described in the Appendix to find the value of each state, starting with the terminal states and working backward to solve for the value of the predecessor states. When the values of the states converge, we know that we have a solution for the values of the states for a game of indefinite length. The difference between the algorithm used here and the one used in the Appendix is that, here, we assume that the exposed player's choice is derived from a probabilistic minimax over successor state values (instead of a solution to the linear programming problem for the purpose of finding the mixed Nash equilibrium). The protected player's response is the other component of the minimax play: It is the best response to the exposed player's chosen action.

We solved the game in this manner to determine the value of the action-revealing oracle from each state, given that the oracle will be available for playing the game from that state forward until a terminal state is reached. If we were to play a full game from the default starting state (each player has one unit and no bases), the expected value of the state (and, thus, the oracle) is 0.0050. There are 35 fair[1] starting states for this game. In these fair starting states, the average value for the improvement in the score when the action oracle is used (instead of the Nash equilibrium strategy) is 0.0808. If we collect the value differences earned by the protected player in each state and then sort them according to their value, we can generate a distribution of the state values and display the distribution as a histogram. A histogram of the values for all fair starting states is shown in Fig. 4(a), and a histogram of the values for each of the 2021 nonterminal states is depicted in Fig. 4(b). The mean value improvement for all nonterminal states is 0.0692. Another analysis we can perform with these data is to determine how much expected value could be earned from the action oracle's information if it was only available for a *single* decision. Essentially, we would like to determine the value of a state if we were allowed to use the oracle once in that state and then we were not allowed to use it for any portion of the remainder of the game. For the remainder of the game, we would use the mixed strategy Nash equilibrium policy described in the Appendix.

We compute this value by first determining the set of successor states and the associated probabilities that occur when the action oracle is available (and the minimax solution method is used). We then look up the values for just the states that were successor states when using the mixed-equilibrium solution method. By weighting these values according to their transition probabilities and summing, we get the expected value of the game if the remainder of the game was played without the action oracle. If we subtract the expected value of the successor states from the value of the initial state, the result is the difference in the value of the joint action prescribed by the action oracle and the joint action described by the mixed Nash equilibrium: It is the value of the action oracle's information for one decision from the current state. If we use the action oracle only for the first decision from the default starting state, the value gain from using the oracle is 0. The mean value of the one-decision oracle's information over all 35 fair starting states is 0.0041. Fig. 5(a) shows a histogram of the values for each of the 35 fair starting states if the action oracle was available for only the first decision in those games. For all 2021 nonterminal states in the game, the average value of the one-shot decision made with the help of the action oracle is 0.0129. Fig. 5(b) shows a histogram of the values for each nonterminal state if the action oracle was available for only a single decision at that state.

From these results, we see that, in general, the action oracle does provide a value advantage but that the advantage does depend on which state the game starts in. A secondary

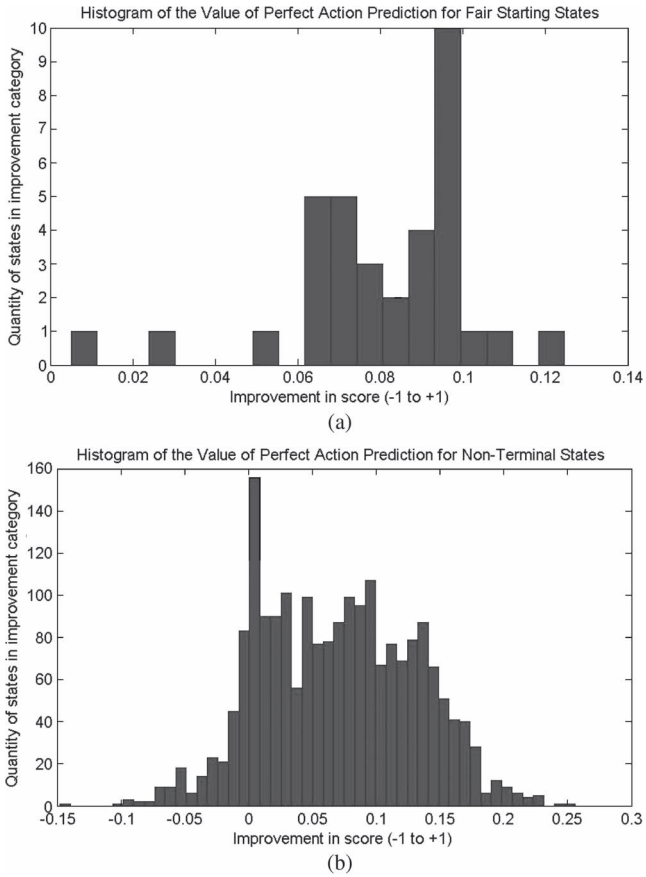


Fig. 4. When using the action oracle for the entire game, a histogram of our improvements in the value is shown for fair starting states in (a) and for all nonterminal states in (b). (a) Improvement in score from starting states when the action oracle is available on every turn. (b) Improvement in score from nonterminal states when the action oracle is available on every turn.

[1]A *fair* starting state is one in which both players have the same number of assets and the same configurations (bases and units) at each location. In other words, if the player identities were switched, a fair state has the property that the ID-reversed state is an isomorphism of the original state. Starting states must also be nonterminal. Thus, a state where all players have no assets is fair, but not a valid starting state.

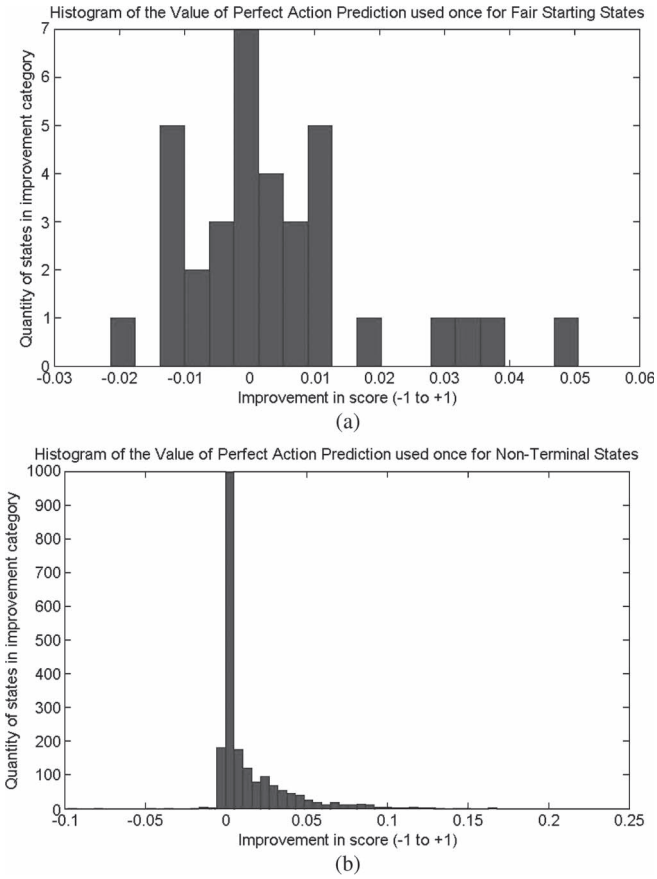


Fig. 5. When using the action oracle for only the first decision, a histogram of our improvements in the value is shown for fair starting states in (a) and for all nonterminal states in (b). (a) Improvement in score from starting states when the action oracle is available for only the first decision. (b) Improvement in score from all nonterminal states when the action oracle is available for only the first decision.

interpretation of these results is that the action oracle is more important in some states than others. In other words, the value of the information about what the opponent will do next varies depending on which state we are in.

V. Conclusion and Future Work

In this paper, we have shown that the environment value of an opponent model is equivalent to the minimum expected payoff value if the model was perfect. This environment value is equal to or greater than zero for any oracle (a perfect opponent model) because knowing better information about the game can never lead to lower valued decisions. Our novel method for finding the environment value of an opponent model can fulfill several existing needs. First, the environment-value-finding technique provides the first method for comparing different types of opponent models without having to consider the distribution of opponents that one might face. Since the environment value is a nonzero scalar, types of models can be rank-ordered by the value of the information that they would provide in a given environment. This full ordering of models can help agent developers focus their efforts on developing the most profitable opponent models for the particular conditions of the environment. Second, because the environment value of an opponent model is the maximum value earnable against a Nash equilibrium opponent, it can be used as a benchmark with which to compare the performance of actual models being used when playing against a Nash equilibrium opponent.

Unfortunately, the game-theoretic method is limited because the Nash equilibrium strategies must be calculable with available resources. When this precondition does not hold, we may be better off approximating the environment value using an estimation of the distribution over likely opponents, which is the focus of future work.

We now present a sketch of an alternative technique for approximating a value of a model when game-theoretical solutions are intractable or inappropriate, but the set of opponents is drawn from a known distribution of known opponent types. The steps for computing the value of a model for a probability distribution over known opponent types are actually a generalization of the game-theoretic method described previously. Essentially, in the game-theoretic method, we made the assumption that our opponent was rational in the game-theoretic sense and that they would always play the Nash equilibrium.

In a more general description of our approach, we assumed that the probability that our opponent was a Nash equilibrium player was 1.0, and then we used the Nash equilibrium solution to determine the behavior of our opponent without considering any other possible opponent types. In the Bayesian technique that we are about to describe, we instead use a probability distribution over multiple arbitrary opponent types.

To use this technique, we first collect and instantiate a representative opponent from each type. For each opponent type, we create an oracle that would reveal the state, action, or policy of the opponent type in a transformation of the original game where the target information remained hidden. We then create the two versions of the environment: the original one in which our agent has no access to the opponent's information, and the transformed one in which an oracle will provide one of the missing pieces of information about the opponent.

To find an approximation of the best response in the original environment, we then use each of the instantiated opponents within an M^* [6] search to find the optimal action to take within the game. The M^* search inserts a model of the opponent into the search algorithm instead of assuming a Nash equilibrium opponent. In this way, an expected-value-based search engine can compute the best action to take against the given opponent. We generate a static best response policy for each state for each opponent type. To find an approximation of the best response in the transformed environment, we simply reveal the missing information to our agent within the M^* search similarly to the way the action information was revealed in the previous section. We then compute the new optimal action to take for each state for each opponent. Then, we can calculate the difference between the value of each state (in the original and transformed environments) for each opponent. We incorporate the probability distribution (over opponent types) into the sum of the differences to estimate the improvement of a particular model type for that distribution of opponent types.

When using a Nash equilibrium opponent, we could make some guarantees about the value earnable by a perfect model of the opponent. Because a Nash equilibrium player is going to

play the most secure strategy (when the preconditions for equilibrium have been met), we can assume that for any opponent not using the Nash equilibrium technique, we can earn at least that much and possibly more if we had an oracle to consult. However, if our analysis technique does not rely on assuming our opponent will play the Nash equilibrium, this guarantee no longer holds. Two caveats must be made about the environment value of the model derived using the Bayesian technique.

1) The value improvement computed is reliable only for the distribution over opponent types used in the calculation. If the distribution of actual agents differs from the distribution used in the computation, then the environment value could be different as well.
2) The value calculated with the Bayesian technique is no longer a function that solely depends on the environment. In the Bayesian technique, the elements of the weaknesses of the opponents are included in the computation and can contribute to an increase in the apparent value of the model.

APPENDIX
SIMULTANEOUS MOVE STRATEGY GAME

Real-time strategy (RTS) games provide an interesting environment for analysis of the techniques discussed in this paper. An RTS game allows players to test their ability to outthink their opponents in several ways. These games often involve exploration of unknown territory for the purpose of locating resources (which can be used to produce buildings and units) and finding the enemies. When the enemies are located, the player must defend his buildings and resource-gathering points while simultaneously trying to eliminate the enemy presence.

Unfortunately, most commercially available RTS games provide no access to the underlying information available to each player. Furthermore, these games have a massive state space and action space, which would preclude demonstration of significant empirical results without an extremely large sample of games.

Here, we present a two-player simultaneous-move turn-based game that has a comparatively small state space and action space that provides some of the key elements of an RTS game. The game we describe has stochastic results of interactions and decision sets that have varying size, depending on what actions were previously taken by the players. The remainder of this appendix presents the game and a technique for finding equilibrium in the game.

A. Game Overview

The goal of this game is to eliminate all enemy assets (enemy bases and units). Each player starts with one unit (a mobile military force) but may build additional units if the player also owns bases (stationary unit production facilities). During a turn, a player may move any or all of his units from their current locations to any other connected locations or use units to build bases (which can then build more units). When a player's units occupy the same location as the enemy's units or encounter units when traveling from one location to another, a battle ensues, and at most, only one of the players will have surviving units. When a player's units are the only units at the location of an enemy base, they may destroy the base (but may also lose units in the process). When one side has no remaining bases or units, the game is over.

The game is played in a world composed of several locations where players' units or bases could exist. Each location is identical and is fully connected to every other location. Each player simultaneously makes decisions on what to do with their units at the beginning of the turn. The players pass their decisions (known as orders) to the game server, which then processes the orders and determines the results. The results of any interactions between the players' units are presented to the players in the following turn, along with the status of the nodes.

In a turn, a player must decide what to do with each unit he owns. The choices are as follows.

- *Idle*: remain in the current location.
- *Move (destination)*: attempt to move to a different connected location.
- *Build*: attempt to build a base (which can produce more units). To build a base, the unit must remain at the same location and survive any enemy attacks during the turn. If the unit is still present at the end of the turn, the base is constructed.
- *Destroy*: if a player's unit is present at an enemy base, the unit may attempt to destroy that base. If the unit survives any enemy attack for the whole turn, then the base will be destroyed, but there is a chance that the unit will also be destroyed.

At any location where the player owns a base, the game server will produce a new unit each turn (assuming that the player has fewer units than the maximum unit capacity γ). Players may wish to build bases and defend them during the course of the game because they are the only methods of generating new units for a player to command.

We developed a computerized version of this game. A game server keeps track of the true state of the world and passes this information to the client agents. The clients then make decisions on what actions to take and pass their desired orders back to the game server.[2] The game server processes the orders, handling any interactions between bases and units, and updates the state of the world prior to starting the next turn. If terminal conditions are achieved, the winner and the loser (or tied players) are declared. The game we analyzed was characterized by the following parameter values:

Parameter	Description	Default value
l	Number of locations in the game	4
γ	Maximum number of units owned by single player	3
δ	Defensive strength bonus	1.5

The game starts with each player having one unit. The units are randomly placed at two unique locations. Once the game

[2]Although the clients are queried in order, no orders are executed until the server has received both player's orders. Thus, the game is effectively a simultaneous action game.

starts, the server processes the clients' orders in the following sequence:

- move units;
- generate new units (at bases);
- resolve conflicts;
- destroy bases;
- build bases.

When all of the phases are complete, the server sends the updated state of the world to the players and requests their next turn orders. This cycle continues until at least one player's assets have been eliminated. The details of the stochastic interactions and formulas for computing the results of each action and conflict between the two forces are discussed in [8].

The game continues over one or more turns (through all of the phases listed above) until at least one player has no remaining assets (units or bases) at the end of a turn. If neither player has any remaining assets, then the game terminates with a tie (each player scores zero points). If one player has assets remaining when the other player has none, then the player with assets receives one point, and the other player loses one point. Thus, this game is a zero-sum game.

B. Analyzing the Game

While the game appears to be relatively simple, the parameters described above allow for 2136 unique state isomorphisms in the full-information version of this game.[3] Of these, 115 are states with terminal conditions, 57 wins, 57 losses, and 1 tie for each player. The parameters also allow for five action options per unit (idle, move to one of three other locations, and, depending on whether there is or is not an enemy base or friendly base present, possibly build a base or possibly destroy a base). With a unit capacity limitation of three units, a worst case scenario is when each player has three units to command, yielding a set of $(3^5)^2 = 59\,049$ unique joint actions if each unit is considered as a unique entity. Because of the limitation of only one player occupying a location at a time, if we consider unit action isomorphisms,[4] then the number of joint actions is greatly reduced. The worst case scenario yields 2304 joint unit action isomorphisms, while the average number of joint action isomorphisms per state is approximately 211. Fortunately, even with this rather large apparent branching factor, only one of the 2136 unique states will be arrived at from any state-and-joint-action branch. Unfortunately, the state that it arrives at is nondeterministic and is often based on the results of one or more stochastic processes occurring during the game turn.

[3]Because of the fully connected nature of the locations in this game, many physical states are isomorphic. For example, if only two locations have units present, then it does not matter *which* two locations those units are at, but just which player occupies and how many units are at each location. State isomorphisms are important for reducing computational complexity because each isomorphism represents all of the states comprising it, and once a complex computation is performed for one member of the isomorphism, all other members can be generated by mapping the results from one state to the other without repeating the computation.

[4]In a unit action isomorphism, the unique identity of a player's individual units is removed such that if a player has two units at a location and unit 1 moves to another location, while unit 2 is idle, the decision is isomorphic to the condition where unit 2 moves to that location and unit 1 is idle.

As we shall see, calculating equilibrium is computationally expensive. The remainder of this section focuses on finding an equilibrium solution to support computations of the value of a model.

To decide what course of action a player should take, we would ideally like to know the value of each of the 2136 states to each player as well as the likelihood that each joint action would lead from one of those states to another. Armed with these values and probabilities, we could calculate an equilibrium solution for risk-neutral[5] agents.

Clearly, the value of a terminal state to a player is equivalent to the score obtained in that state by that player. However, when we try to evaluate nonterminal states, the actions that might lead to a terminal state are *joint* actions, and, thus, while one player is desiring a certain joint action, the other player may find it equally undesirable and attempt to avoid their component of the joint action. Furthermore, many of the joint actions have probabilities of leading to nonterminal states (or even back to the original state), creating potential nonterminated recursions in any algorithm attempting to recursively determine the value of a state. Since there can be sequences of actions by each player that yield cycles in the visited states, the game is not guaranteed to terminate. In general, this is an undesirable situation for game-theoretic analysis.

We can estimate the values of the states for each player using a combination of game-theoretic techniques for strategy selection and value iteration for determining the value of each state in the game. We start by determining the transition probabilities for the game by running a Monte Carlo [9] simulation from each state for each joint action. Then, we iteratively solve the full-information extensive-form game by creating successively longer length games (starting from endgame and working backward). During each iteration, we update the value of a state by using Nash equilibrium to decide a player's mixture over actions (based on the expected values of the states) and then use the precalculated transition probabilities from those states to get an expected value of the outcome.

The overview of the algorithm is as follows.

- Determine the transition probabilities for each state and joint action to another state using Monte Carlo simulation.
- Determine the terminal states (where there is a definitive win, loss, or tie for the players).
- Initialize the appropriate value to every terminal state from the second player's perspective ($+1$, -1, or 0), and value all other states as 0.
- Repeat until convergence or for as many iterations as time allows.
    1) Determine which states have a positive probability of transition to a nonzero-valued state.
    2) For each of those states:
        a) Generate a normal-form game zero-sum payoff matrix for that state by collecting the joint action payoffs (sum of probability-weighted values of the states transitioned to by each joint action).

[5]Risk-neutral agents are ones who regard their game score as their utility function. Thus, a risk-neutral agent in this game desires a win equally as much as he despises a loss, and is indifferent when the game is tied.

b) Solve the state's normal-form game (by calculating the equilibrium condition) and determine the value associated with the solution (from player 2's perspective).
c) Update the state's value for the next iteration to the value calculated in the last step.

- Report the resulting values for each state.
- Report the equilibrium solution's recommendation for distribution over joint actions in each possible state.

To obtain the transition probabilities, we play the full-information version of the game from every state with every possible combination of legal player orders using Monte Carlo sampling to determine the probability that a joint action in the originating state leads to a destination state. Since both player's actions are fully specified in each case, the resulting transition probabilities are a function of only the rules of the game, not the strategy of the players. These transition probabilities are effectively a (noisy) mathematical representation of the rules of the game.

To form the payoff matrix for each state and joint action, we look up the transition probabilities associated with each possible joint action, determine all the successor states, and look up their values. For actions chosen by player 1 (i) and player 2 (j), the set of n associated transition probabilities $P_1, \ldots, P_n$, the associated states $S_1, \ldots, S_n$, and their values (from the second player's perspective) $V_1, \ldots, V_n$, we compute the payoff or utility u, i.e.,

$$u_{ij} = \sum_{k=1,\ldots,n} P_k V_k. \tag{1}$$

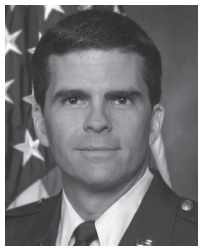
To solve the zero-sum normal form game generated when utilities for every possible joint action have been computed from a given state, we describe the conditions of the game in terms of a constrained optimization problem and use a linear program solver such as `simlp` in MATLAB. The result is a probability distribution over actions for each player and a value (from that player's perspective) for playing that strategy.

In the method above, we described how to strongly solve the simultaneous-move strategic game to generate an approximation of the strategy of a Nash equilibrium player. This type of strategy becomes the basis for optimal play when the agent is uninformed.[6]

[6]By uninformed, we mean a strategy that has no information about the specific opponent's characteristics. The uninformed agent assumes that his opponent is rational, and the equilibrium becomes desirable for describing strategies that guarantee some minimum value in the environment when all players are acting rationally.

REFERENCES

[1] J. F. Nash, "Equilibrium points in n-person games," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 36, no. 1, pp. 48–49, Jan. 1950.
[2] R. J. Aumann and A. Brandenburger, "Epistemic conditions for Nash equilibrium," *Econometrica*, vol. 63, no. 5, pp. 1161–1180, 1995.
[3] J. F. Nash, "Non-cooperative games," *Ann. Math.*, vol. 54, no. 2, pp. 286–295, 1951.
[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.
[5] C. A. Luckhardt and K. B. Irani, "An algorithmic solution of n-person games," in *Proc. 5th Nat. Conf. Artif. Intell. (AAAI)*, Philadelphia, PA, 1986, pp. 158–162.
[6] D. Carmel and S. Markovitch, "Incorporating opponent models into adversary search," in *Proc. 13th Nat. Conf. Artif. Intell.*, 1996, pp. 120–125.
[7] R. D. McKelvey, A. M. McLennan, and T. L. Turocy, Gambit: Software Tools for Game Theory2007. [Online]. Available: http://gambit.sourceforge.net
[8] B. J. Borghetti, "Opponent modeling in interesting adversarial environments," Ph.D. dissertation, Univ. Minnesota, Twin Cities, MN, Aug. 2008.
[9] N. Metropolis and S. Ulam, "The Monte Carlo method," *J. Am. Stat. Assoc.*, vol. 44, no. 247, pp. 335–341, Sep. 1949.



Brett J. Borghetti received the B.S. degree in electrical engineering from Worcester Polytechnic Institute, Worcester, MA, in 1992, the M.S. degree in computer systems from the U.S. Air Force Institute of Technology, Dayton, OH, in 1996, and the Ph.D. degree in computer science from the University of Minnesota, Twin Cities, in 2008.

He has worked at the Air Intelligence Agency, San Antonio, TX, the National Air and Space Intelligence Center, Dayton, OH, the U.S. Strategic Command, Omaha, NE, and the Training Systems Product Group, Dayton, OH. He is currently a Lieutenant Colonel in the U.S. Air Force and an Assistant Professor of computer science with the U.S. Air Force Institute of Technology, Wright-Patterson Air Force Base, Dayton, OH. He received his commission through the Air Force Reserve Officer Training Corps in 1992.