Back to the Future: Toward a Hybrid Architecture for Ad Hoc Teamwork

Hasra Dodampegama, Mohan Sridharan

Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK
hhd968@student.bham.ac.uk, m.sridharan@bham.ac.uk

Abstract

State of the art methods for ad hoc teamwork, i.e., for collaboration without prior coordination, often use a long history of prior observations to model the behavior of other agents (or agent types) and to determine the ad hoc agent's behavior. In many practical domains, it is difficult to obtain large training datasets, and it is necessary to quickly revise the existing models to account for changes in team composition or the domain. Our architecture builds on the principles of *step-wise refinement* and *ecological rationality* to enable an ad hoc agent to perform non-monotonic logical reasoning with prior commonsense domain knowledge and models learned rapidly from limited examples to predict the behavior of other agents. In the simulated multiagent collaboration domain *Fort Attack*, we experimentally demonstrate that our architecture enables an ad hoc agent to adapt to changes in the behavior of other agents, and provides enhanced transparency and better performance than a state of the art data-driven baseline.

1 Introduction

Ad hoc teamwork (AHT) refers to the problem of enabling an agent to cooperate with others on the fly (Stone et al. 2010). Figure 1 shows motivating scenarios from *Fort Attack*, a simulated benchmark for multiagent collaboration (Deka and Sycara 2020). Three guards (in green) are trying to protect a fort from three attackers (in red). An episode ends when all members of a team are killed, an attacker reaches the fort, or guards protect the fort for a sufficient time period. Each agent can move in a particular direction with a particular velocity, or shoot over a particular range. Each agent is aware of the world state (e.g., location, status of each agent) at each step, but the agents have not worked with each other before and do not communicate with each other. This example is representative of many practical multiagent collaboration scenarios (e.g., disaster rescue), and poses knowledge representation, reasoning, and learning challenges. An ad hoc agent, e.g., "1" in Figure 1, has to reason with prior commonsense domain knowledge (e.g., some domain and agent attributes, axioms governing change) and uncertainty, adaptinf its action choices to changes in the domain or in the behavior of other agents.

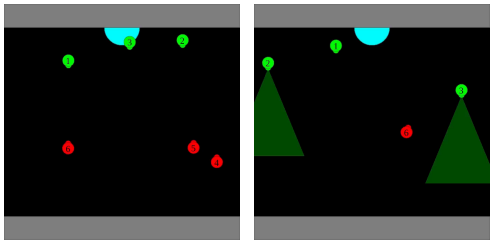

Figure 1: Screenshots from the *fort attack* environment.

There has been considerable research in AHT (Mirsky et al. 2022). Initial methods had the ad hoc agent choose from and execute encoded protocols for collaboration based on the current state. State of the art methods include a data-driven component; they learn probabilistic or deep network models and policies from a history of experiences to estimate the behavior of other agents (or agent "types") and to compute the ad hoc agent's actions. These methods do not fully leverage the available domain knowledge. Unlike existing work, our architecture draws on research in cognitive systems, moves beyond just data-driven optimization, and builds on the principles of step-wise refinement (Sridharan et al. 2019) and ecological rationality (Gigerenzer 2016) to:

- Support non-monotonic logical reasoning with prior commonsense domain knowledge, enabling an ad hoc agent to plan and execute actions to achieve desired goals, and provide relational descriptions of its decisions;
- Use reasoning to guide learning and revision of simple anticipatory models that satisfactorily mimic the behavior of other agents from limited examples; and
- Use an algorithmic model of heuristic methods to identify attributes and heuristic methods that support reliable and efficient reasoning and learning.

In the Fort Attack domain, we demonstrate that our architecture supports reliable, efficient, and transparent reasoning, learning, and adaption, and better performance than a state of the art data-driven baseline.

2 Related Work

Research in AHT has existed under different names for at least 15 years (Mirsky et al. 2022). Early work encoded spe-

cific protocols (or plays) for different scenarios and enabled an agent to choose (or plan) a sequence of protocols based on the current state (Bowling and McCracken 2005). Other work has used sampling-based methods such as Upper Confidence bounds for Trees (UCT) for determining the ad hoc agent's action selection policy (Barrett et al. 2013). UCT has been combined with biased adaptive play to provide an online planning algorithm for ad hoc teamwork (Wu, Zilberstein, and Chen 2011), and researchers have explored using Value Iteration or UCT depending on the state observations (Barrett, Stone, and Kraus 2011). Much of the current (and recent) research has formulated AHT as a probabilistic sequential decision-making problem, assuming an underlying Markov decision process (MDP) or a partially observable MDP (POMDP) (Barrett et al. 2017; Chen et al. 2020; Santos et al. 2021; Rahman et al. 2021). This has included learning different policies for different teammate *types* and choosing a policy based on the teammate types seen during execution (Barrett et al. 2017). Later work has used recurrent neural networks to avoid switching between policies for different teammate types (Chen et al. 2020).

A key component of state of the art AHT methods learns to predict the behaviors of other agents using probabilistic or deep neural network methods and a long history of prior interactions with similar agents or agent types (Barrett, Stone, and Kraus 2011; Barrett et al. 2017; Rahman et al. 2021). The predictions of other agents' actions are then used to optimize the ad hoc agent's actions. Different ideas have been introduced to make this learning more tractable. For example, recent work used sequential and hierarchical variational auto-encoders to model the beliefs over other agents, and meta-learned approximate belief inference and Bayes-optimal behavior for a given prior (Zintgraf et al. 2021). Learned policy methods have been combined with predictive models to account for the behavior of changing agents (Santos et al. 2021), and a Convolutional Neural Network-based change point detection method has been developed to adapt to changing teammate types (Ravula, Alkoby, and Stone 2019). Other work has used the observation of current teammates and learned teammate models to learn a new model (Barrett et al. 2017). Despite these innovative ideas, learning these predictive models requires substantial time, computation, and training examples, and the internal mechanisms governing the decisions of these methods are difficult to understand.

In a departure from existing work, our AHT architecture combines knowledge-based and data-driven reasoning and learning to enable the ad hoc agent to adapt to different agent behaviors and team compositions, and to provide transparency in decision making, as described below.

3 Architecture for Ad Hoc Teamwork

Figure 2 is an overview of our architecture. The ad hoc agent performs non-monotonic logical reasoning with prior commonsense domain knowledge and models of other agents' behaviors learned incrementally from limited examples, using heuristic methods to guide reasoning and learning. These components are described using the following example.

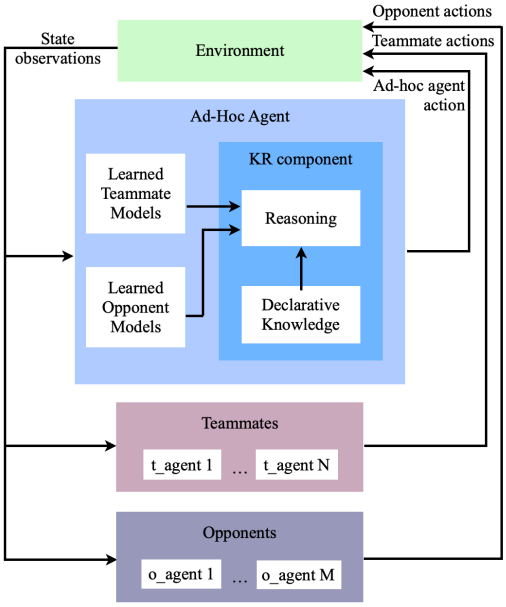

Figure 2: Architecture combining knowledge-based and data-driven heuristic reasoning and learning.

Example Domain 1 *[Fort Attack (FA) Domain]*
Consider a scenario with three guards protecting a fort from three attackers (Figure 1). One of the guards is the ad hoc agent that can adapt to changes in the team and domain. Prior commonsense domain knowledge includes relational descriptions of some domain attributes (e.g, location of agents), agent attributes (e.g., shooting capability and range), default statements (e.g., attackers typically spread and approach the fort), and axioms governing change such as: (a) an agent can only shoot others within its shooting range; and (b) an agent can only move to a location nearby. The ad hoc agent also builds and reasons with predictive models of other agents' behavior (i.e., action choices in specific states). The knowledge and models may need to be revised, e.g., an agent's behavior may change over time.

3.1 Knowledge Representation and Reasoning

In our architecture, any domain's transition diagram is described in an extension of the action language $\mathcal{AL}_d$ (Gelfond and Inclezan 2013) that supports non-Boolean fluents and non-deterministic causal laws (Sridharan et al. 2019). Action languages are formal models of parts of natural language for describing transition diagrams of dynamic systems. $\mathcal{AL}_d$ has a sorted signature with *actions*, *statics*, i.e., domain attributes whose values cannot be changed by actions, and *fluents*, i.e., attributes whose values can be changed by actions. *Basic/inertial* fluents obey laws of inertia and can be changed by actions, whereas *defined* fluents do not obey laws of inertia and are not changed directly by actions. $\mathcal{AL}_d$ supports three types of statements: *causal law*, *state constraint* and *executablility condition* (examples below).

Knowledge Representation: Any domain's representation

comprises a system description $\mathcal{D}_C$, a collection of statements of $\mathcal{AL}_d$, and history $\mathcal{H}_C$. $\mathcal{D}_C$ has a sorted signature Σ_C and axioms describing the domain's transition diagram. Σ_C defines the basic sorts, and describes the domain attributes (statics, fluents) and actions in terms of the sorts of their arguments. Basic sorts of FA domain such as ah_agent (ad hoc agent), ext_agent (external agent), $agent$, $guard$, $attacker$, dir, x_val, y_val, and $step$, are arranged hierarchically, e.g., ah_agent and ext_agent are subsorts of $agent$. Statics include relations $next\_to(x\_val, y\_val, x\_val, y\_val)$ that encode the relative arrangement of places. Domain fluents include relations:

$$in(ah\_agent, x\_val, y\_val),\ face(ah\_agent, dir),$$
$$shot(agent),\ agent\_in(ext\_agent, x\_val, y\_val),$$
$$agent\_face(ext\_agent, dir),\ agent\_shot(ext\_agent)$$

which encode the location, orientation, and status of the ad hoc agent and other agents; the last three are defined fluents. Also, actions of the FA domain include:

$$move(ah\_agent, x\_val, y\_val),\ rotate(ah\_agent, dir),$$
$$shoot(agent, agent),\ agent\_move(ext\_agent, x\_val, y\_val)$$
$$agent\_rotate(ext\_agent, dir)$$

which encode an agent's ability to move, rotate, and shoot, and to mentally simulate exogenous actions by other agents. Σ_C also includes relations $holds(fluent, step)$ and $occurs(action, step)$ to imply that a fluent is true and an action is part of a plan (respectively) at a time step. Given this Σ_C, axioms encoding the dynamics include:

$$move(R, X, Y) \textbf{ causes } in(R, X, Y)$$
$$\neg in(R, X1, Y1) \textbf{ if } in(R, X2, Y2),\ X1 \neq X2, Y1 \neq Y2$$
$$\textbf{impossible } shoot(R, A) \textbf{ if } agent\_shot(A)$$

which encode the outcome of the $move$ action (causal law), state that an agent can only be in one location at a time (constraint), and prevent the consideration of an action whose outcome has been accomplished (executability condition).

The history $\mathcal{H}_C$ of a dynamic domain is usually a record of observations, i.e., $obs(fluent, boolean, step)$, and action executions, i.e., $hpd(action, step)$ at specific time steps. We also include default statements describing the typical values of fluents in the initial state, e.g., it is initially believed that attackers spread and attack, and cannot shoot.

$$\textbf{initial default } spread\_attack(X) \textbf{ if } attacker(X)$$
$$\textbf{initial default } \neg shoot(X, ah\_agent) \textbf{ if } attacker(X)$$

This representation does not assign numerical values to degrees of belief in these defaults, but supports elegant reasoning with defaults and any specific exceptions.

Reasoning: To reason with knowledge, the ad hoc agent translates the $\mathcal{AL}_d$ description to program $\Pi(\mathcal{D}_C, \mathcal{H}_C)$ in CR-Prolog, a variant of Answer Set Prolog (ASP) (Gebser et al. 2012) that supports consistency restoring (CR) rules (Balduccini and Gelfond 2003). ASP is based on stable model semantics, represents constructs difficult to express in classical logic formalisms, and encodes *default negation* and *epistemic disjunction*, i.e., unlike "$\neg a$" that states *a is believed to be false*, "$not\ a$" only implies *a is not believed to be true*, and unlike "$p \vee \neg p$", "*p or* $\neg p$" is not tautologous. Each literal is true, false, or unknown, and the agent only believes what it is forced to believe. ASP also supports non-monotonic reasoning, i.e., revision of previously held conclusions, an essential ability in dynamic domains.

$\Pi(\mathcal{D}_C, \mathcal{H}_C)$ has the signature and axioms of $\mathcal{D}_C$, inertia axioms, reality checks, closed world assumptions for defined fluents and actions, observations, actions, and defaults from $\mathcal{H}_C$, and a CR rule for every default allowing the agent to assume that the default's conclusion is false in order to restore consistency under exceptional circumstances. For example:

$$\neg spread\_attack(X) \stackrel{+}{\leftarrow} attacker(X)$$

allows the ad hoc agent to consider the rare situation of attackers mounting a frontal attack. All reasoning tasks (e.g., planning, diagnostics, and inference) can be reduced to computing *answer sets* of Π after including suitable helper axioms, e.g., to drive the agent to search for plans to achieve a goal. Our ad hoc agent can pursue different goals, e.g., shoot a nearby attacker or move to cover an unprotected region:

$$goal(I) \leftarrow holds(agent\_shot(X), I),\ attacker(X)$$
$$holds(nearby(ag_{ah}, X), I)$$
$$goal(I) \leftarrow holds(in(ag_{ah}, X, Y), I),$$
$$holds(in\_free\_region(X, Y), I)$$

The ad hoc agent selects goals by trading off their priorities with the estimated cost of pursuing them, toward the overall objective of ensuring the fort's safety. Each answer set represents the ad hoc agent's beliefs about the world; we use the SPARC system (Balai, Gelfond, and Zhang 2013) to compute answer sets. Example SPARC programs for FA domain are in our repository (Dodampegama and Sridharan 2022).

Knowledge-based reasoning methods are often criticized for requiring comprehensive domain knowledge, but ASP has been used by an international research community to reason with incomplete knowledge, and modern ASP solvers are efficient for large knowledge bases (Erdem and Patoglu 2018). The effort involved in encoding knowledge is typically much less than training purely data-driven systems; also, most of this knowledge is encoded only once and can be revised over time (Sridharan and Meadows 2018).

3.2 Agent Models and Heuristics

A key component of our architecture is the ability to learn models of the behavior of other agents, supporting rapid, incremental updates and accurate predictions. In the FA domain, other agents include teammates (guards) and opponents (attackers). To build initial versions of these models, we handcrafted policies for four types of agents, two each of guards and attackers. These policies mimic simple strategies, e.g., attackers that spread and attack. We executed these policies and recorded states and actions of agents in a few episodes to create 10000 examples.

Ecological Rationality and Heuristics: To build predictive models, we used the *Ecological Rationality* (ER) approach,

Description of attribute	Number
x position of agent	6
y position of agent	6
distance from agent to center of field	6
polar angle of agent with center of field	6
orientation of the agent	6
distance from agent to fort	6
distance to nearest attacker from fort	1
number of attackers not alive	1
previous action of the agent	1

Table 1: Attributes considered by ad hoc agent to build simple predictive models of other agents' behavior.

which builds on Herb Simon's definition of *Bounded Rationality* (Gigerenzer 2020) and the rational theory of heuristics (Gigerenzer and Gaissmaier 2011). Unlike the focus on optimal search in many fields (e.g., finance, computing), ER explores decision making under true uncertainty (i.e., in open worlds), characterizes behavior as a function of the internal (cognitive) processes and the environment, and focuses on *adaptive satisficing*. Also, unlike the use of heuristics to explain biases or irrational behavior (e.g., of humans, in psychology), ER considers heuristics as a strategy to ignore part of the information in order to make decisions more quickly, frugally, and/or accurately than complex methods (Gigerenzer and Gaissmaier 2011). It advocates an adaptive toolbox of classes of simple heuristics (e.g., one-reason, sequential search, lexicographic), and comparative out-of-sample testing to identify heuristics that leverage the target domain's structure. This approach has provided good performance in many applications (Gigerenzer 2016) characterized by factors also observed in AHT, e.g., the need to make rapid decisions under resource constraints.

Representational Choices and Models: Based on the ER approach, we applied statistical attribute selection methods to the set of 10000 samples to identify the pose (i.e, position, orientation) of each agent, its recent action, and other key attributes defining behavior—Table 1. Since these attributes can take a wide range of values, we represented them using the principles of abstraction and refinement in ER. In particular, we considered polar coordinates and relative distance of each agent from the fort. We also built on prior work on refinement in robotics (Sridharan et al. 2019) to reason about positions at the level of coarser (abstract) regions and finer-granularity grid locations that are components of these regions, formally coupling the descriptions ($\mathcal{D}_C, \mathcal{D}_F$) through component relations and bridge axioms such as:

$$in^*(A, R) \textbf{ if } in(A, X, Y),\ component(X, Y, R)$$
$$next\_to^*(R_2, R_1) \textbf{ if } next\_to^*(R_1, R_2)$$

where location (X, Y) is in region R and superscript "*" refers to relations in $\mathcal{D}_C$. An example ASP program with $\mathcal{D}_F$ is in our repository. Our definition of refinement formally links the two descriptions, enabling the ad hoc agent to automatically choose the relevant part of the descriptions at run-time based on the goal or abstract action, and to transfer relevant information between the granularities.

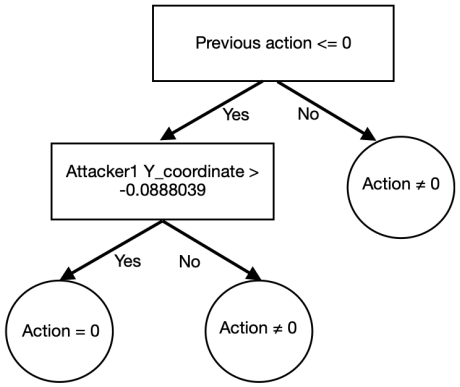

Figure 3: Fast and frugal tree for an attacker type agent.

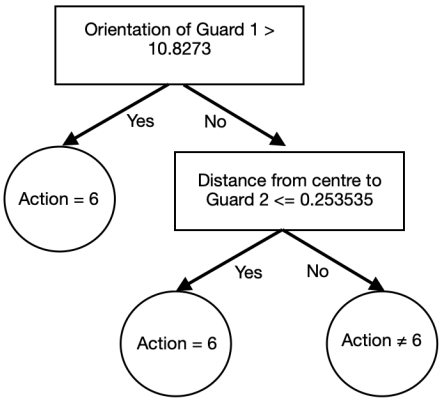

Figure 4: Fast and frugal tree for a guard type agent.

After these representational choices, we matched the FA domain's environmental factors (e.g., dynamic changes, rapid decisions with limited samples) with the adaptive heuristics toolbox to explore: an ensemble of "fast and frugal" (FF) decision trees in which each tree provides a binary class label and has its number of leaves limited by the number of attributes (Gigerenzer and Gaissmaier 2011; Katsikopoulos et al. 2021); and STEW, a regularized (linear) regression approach that exploits feature directions and is biased towards an equal weights solution (Lichtenberg and Simsek 2019). We performed statistical testing on unseen examples using ANOVA (analysis of variance) (Fisher 1992) to choose the FF trees-based models for behavior prediction. Unlike many existing methods, these predictive models can be learned and revised incrementally and rapidly. Also, consistent agreement (disagreement) with predictions of an existing model can trigger model choice (revision). Figures 3-4 show example FF trees learned for an attacker and a guard.

Transparency in Reasoning and Learning: The ability to answer causal, contrastive, and counterfactual questions about decisions and beliefs, plays a key role in human reasoning and learning, and promotes acceptability of automated decision-making systems (Anjomshoae et al. 2019;

Algorithm 1: Control Loop of Architecture

Input: N: number of games; $\Pi(\mathcal{D}, \mathcal{H})$: core ASP program, $\mathcal{M}$: behavior models of other agents; $\mathcal{P}$: other agents' policies
Output: game_stats: statistics of games

1 Create environment, load $\mathcal{P}$, initialize environment
2 **for** $i = 0$ **to** $N - 1$ **do**
3 $\mathbf{s} \leftarrow$ state of environment
4 **while** $\neg\, game\_over(\mathbf{s})$ **do**
5 $\mathbf{a}_o \leftarrow$ other_agents_action($\mathbf{s}, \mathcal{P}$)
6 $a_{ah} \leftarrow$ adhoc_agent_action($\mathbf{s}, \Pi, \mathcal{M}$)
7 $\mathbf{a} = \mathbf{a}_o \cup a_{ah}$
8 $\mathbf{s}'$ = execute($\mathbf{s}, \mathbf{a}$)
9 update_models($\mathcal{M}$)
10 **if** $game\_over(\mathbf{s}')$ **then**
11 update(game_stats)
12 initialize environment
13 **else**
14 $\mathbf{s} = \mathbf{s}'$
15 **end**
16 **end**
17 **end**
18 **return** game_stats

Algorithm 2: adhoc_agent_action

Input: $\mathbf{s}$, $\Pi(\mathcal{D}, \mathcal{H})$, $\mathcal{M}$
Output: a

1 **if** $alive(adhoc\_agent)$ **then**
2 $\mathbf{a}_o \leftarrow$ action_predictions($\mathcal{M}$)
3 $\mathbf{s}' \leftarrow$ simulate_effects($\mathbf{a}_o$)
4 zones $\leftarrow$ compute_relevance($\mathbf{s}, \mathbf{a}_o, \mathbf{s}'$)
5 ASP_program $\leftarrow$ construct_program($\mathbf{s}, \Pi$, zones)
6 answer_set $\leftarrow$ SPARC(ASP_program)
7 $a \leftarrow$ next_action(answer_set)
8 **else**
9 $a \leftarrow$ do_nothing
10 **end**
11 **return** a

Fox, Long, and Magazzeni 2017). The use of knowledge-based reasoning and simple predictive models in our architecture provides a good foundation to support the desired transparency in the decisions and beliefs of the ad hoc agent using our architecture. We build on prior work that demonstrated that any question about an action choice (or belief) can be answered by iteratively and selectively identifying axioms and literals that influence this action (belief) and have their antecedents satisfied in the corresponding answer sets (Mota, Sridharan, and Leonardis 2021). Also, existing software tools and simple predefined templates are used to parse textual questions and construct textual answers based on the relevant literals. Due to space constraints, we only provide some qualitative examples (e.g., an execution trace) of explanation generation in Section 4.3.

3.3 Control Loop

The overall control loop of our architecture is described in Algorithm 1. First, the FA game environment is set up, including the other agents' policies $\mathcal{P}$ (Line 1) that are unknown to the ad hoc agent, before each of the N games (i.e., episodes) are played (Lines 2-17). In each game, the other agents' actions are identified based on the current state and $\mathcal{P}$ (Line 5), and the ad hoc agent's action is computed by reasoning with domain knowledge and learned models ($\mathcal{M}$) of other agents' behaviors (Line 6; Algorithm 2). The actions are executed in the simulated environment to receive the updated state (Line 8). The observed state can also be used to incrementally learn and update $\mathcal{M}$, e.g., when observations do not match predictions (Line 9). The updated state is used for the next step (Lines 13-15). This process continues until the game is over; the game's statistics are stored for analysis before moving to next game (Lines 10-12).

In Algorithm 2, the selection of the ad hoc agent's action is only valid if this agent is alive (Lines 1-7). The ad hoc agent first uses the learned behavior models ($\mathcal{M}$) of the other agents (guards, attackers) to predict their next action in the current state (Line 2). It simulates the effect of this action to estimate the next state (Line 3). It uses this information to compute the relevant regions (*zones*) in the domain that need special attention (Line 4). This information is used to automatically determine the regions and related axioms to be considered, and the level of abstraction to be used for reasoning (see Section 3.1, description of refinement in Section 3.2, Line 5). The relevant information is also used to automatically prioritize a goal (e.g., getting to a suitable region) to be added to the ASP program (Line 5). The corresponding answer set provides the next action to be executed by the ad hoc agent (Line 7, returned to Algorithm 1).

4 Experimental Setup and Results

We experimentally evaluated the following hypotheses:

- **H1:** our architecture enables adaptation to different teammate and opponent types;
- **H2:** our architecture supports incremental learning of models of other agents' behavior from limited examples;
- **H3:** our architecture provides better performance than a state of the art data-driven method for AHT; and
- **H4:** our architecture supports the generation of relational descriptions of the ad hoc agent's decisions and belief.

We evaluated these hypotheses in the **FA domain**, a benchmark for multiagent systems research. Each episode (i.e., game) started with three guards protecting the fort and three attackers trying to reach the fort; the ad hoc agent was one of the guards. An episode ended when all members of a team were killed, an attacker reached the fort, or guards managed to protect the fort for a sufficient time period. Each agent had eight action choices: stay in place, move in one of four cardinal directions, turn clockwise or counterclockwise, or shoot. **Performance measures** included the number of steps in an episode, fraction of wins of a particular agent type or team, prediction accuracy of behavior models, and fraction of times a shooting guard eliminated an attacker.

4.1 Experimental Setup

For training, we hand-crafted two sets of policies for attackers and other guards: (**Policy1**) guards stay close to the fort and try to shoot attackers, while attackers spread and approach fort; (**Policy2**) guards and attackers spread and shoot opponents. We designed these policies to capture basic behavioral characteristics in the domain. To simulate training from limited examples, we used only 10000 observations of state and agents' actions to train the ensemble of FF trees. We then tested the ad hoc agent in: (**Exp1**) when other agents used handcrafted policies; and (**Exp2**) when other agents used the following built-in policies of the FA domain:

- Policy 220: guards place themselves in front of the fort and shoot continuously; attackers try to approach the fort.
- Policy 650: guards stay near the fort and try to shoot nearby attackers; attackers try to sneak in from all sides.
- Policy 1240: guards spread out, are willing to move out a bit from the fort, and try to shoot when attackers are nearby; attackers try to sneak in from all sides.
- Policy 1600: guards spread, are willing to move further out from fort, and try to shoot nearby attackers; some attackers approach and shoot the guards, while others stay back and wait for a chance to reach the fort.

Unlike the hand-crafted policies, built-in policies were based on complex deep (graph) neural networks trained over many episodes. This made evaluation challenging because *the ad hoc agent had no prior experience of agents following these policies*. We considered three baselines without our ad hoc agent: (i) **Base1** in **Exp1** with agents following hand-crafted policies; (ii) **Base2** in **Exp2** with agents following built-in policies; and (iii) **GPL**, a state of the art AHT method based on graph neural networks (Rahman et al. 2021) in **Exp3** as an extension of **Exp2**. Each data point in the results below was the average of 40 episodes. For GPL, we took the average results from their supplementary material. We used these experiments to evaluate **H1-H3**; **H2** was also evaluated on a separate dataset and **H4** was evaluated qualitatively instead of using the quantitative measures available in literature.

4.2 Experimental Results

Tables 2-3 summarize the results of **Exp1**. Compared with **Base1** that has all agents using hand-crafted policies, the average number of steps in an episode was a little less and there was a (statistically) significant improvement in the number of episodes in which the guards won, when one of the guards was an ad hoc agent. Given that the ad hoc agent used models of other agents' behavior learned from limited training examples of hand-crafted policies, the higher fraction of wins with episodes of similar length is a good outcome.

Tables 4-5 summarize the results of **Exp2**. We observed that the team of guards with an ad hoc agent using the behavior models trained from the handcrafted policies was able to adapt to the previously unseen built-in policies of the FA domain. In particular, the average number of steps in an episode was about the same or a little less for policies 220, 650, and 1240, compared with **Base2** that had all the agents using the built-in policies. At the same time, there was a statistically significant improvement in the % wins for the team of guards for these policies. The % of episodes in which the team of guards won was rather small for policy 1600, although there was a significant improvement when our ad hoc agent was one of the guards (7% from 2%). Recall that this policy was particularly challenging for the guards; some attackers tried to draw the guards away from the fort and shoot them while others stayed back and waited for an opportunity to sneak in. Notice, however, that for policy 1600, the team with our ad hoc agent was able to prolong the game for much longer compared with **Base2**. These results support **H1**, provide partial support for **H2**, and indicate the benefits of reasoning and learning guiding each other. *Videos of trials, including those with changes in team composition, are in our repository* (Dodampegama and Sridharan 2022).

Policy	**with ad-hoc agent**	**without ad-hoc agent (Base1)**
Policy1	19	22
Policy2	21	22

Table 2: Average number of steps in an episode with handcrafted policies and learned agent models (Exp1).

Policy	**with ad-hoc agent**	**without ad-hoc agent (Base1)**
Policy1	90%	85%
Policy2	67%	62%

Table 3: Average % of episodes in which guards win with handcrafted policies and learned agent models (Exp1).

To further explore **H2**, we evaluated the learned behavior models on a previously unseen examples from the handcrafted policies. Table 6 shows that the prediction accuracy ranged from $60 - 87\%$, i.e., there were errors. However, when used in conjunction with other components of our architecture, these models were sufficient to improve the performance of the team of guards compared with baselines without our ad hoc agent. These results demonstrate the benefits of not over-fitting on the training set and allowing rapid revision of the learned models; they also support **H2**.

Figure 5 summarizes the results of **Exp3** that evaluated **H3** based on the fraction of times a guard agent's shooting eliminated an attacker, i.e., *shooting accuracy*. We compared our ad hoc guard agent with an ad hoc guard agent using GPL, and a guard using **Base2**, in the context of built-in

Policy	**with ad-hoc agent**	**without ad-hoc agent (Base2)**
policy 220	22	27
policy 650	40	40
policy 1240	24	24
policy 1600	34	27

Table 4: Average number of steps per episode with previously unseen built-in policies (Exp2); ad hoc guard reduces (220) or provides comparable results (650, 1240), or extends survival (1600), compared with Base2.

Policy	with ad-hoc agent	without ad-hoc agent (Base2)
policy 220	85%	77%
policy 650	35%	20%
policy 1240	70%	52%
policy 1600	7%	2%

Table 5: Average % of episodes in which guards win with previously unseen built-in policies (Exp2).

Agent Model	Accuracy
Guard type 1	85.5%
Guard type 2	60.0%
Attacker type 1	86.9%
Attacker type 2	85.2%

Table 6: Prediction accuracy of learned behavior models.

policies. Our ad hoc guard's average shooting accuracy was significantly better than that of the ad hoc guard using GPL for three policies (220, 650, 1240); results were comparable for policy 1600. These policies are challenging, e.g., guards are at risk with policy 1600 because the attackers try to draw them out and shoot them. Also, the GPL-based agent was trained for $60 * 160000$ steps, whereas our ad hoc agent reasoned with prior domain knowledge and predictive models learned from 10000 samples to adapt to previously unseen policies. In the final set of experiments in Figure 5 ("mix"), agents other than the ad hoc agent used a random mix of available policies in each episode; once again, our architecture provided higher accuracy. These results support **H3**.

4.3 Execution Trace

As a qualitative example of providing relational descriptions of decisions, consider an exchange with an ad hoc guard agent after it shot an attacker; Figure 6 shows a snapshot.

- **Human:** "Why did you move to (3,14) in step 1?"
- **Ad hoc Agent:** "Because attacker1 was not in range and I had to move to (4,14)". This answer was based on the long(er)-term goal of getting attacker1 in range, and the associated short-term goal of getting to locations such as (4, 14) that will eventually enable it to shoot attacker1.
- **Human:** "Why did you not move to (5,13) in step 4?"
- **Ad hoc Agent:** "Because that would have put attacker1 out of range and I had to shoot attacker1". This example demonstrates the ability to answer contrastive questions, which is an important ability for learning in humans.

Similar scenarios can be created for other question types (e.g., counterfactual). Also, reasoning can generate explanations that refer to team behavior because the ad hoc agent's actions are based on domain state, predicted behavior of other agents, and its goal(s). These results support **H4**.

5 Conclusions

We described an architecture for ad hoc teamwork that combined knowledge-based and data-driven reasoning and

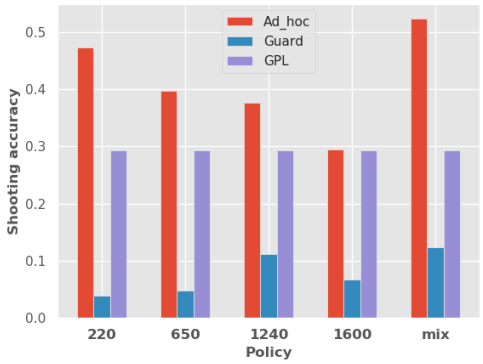

Figure 5: Our ad hoc agent's shooting accuracy was better in Exp3 compared with a guard using the built-in policy and an ad hoc agent using a state of the art data-driven method.

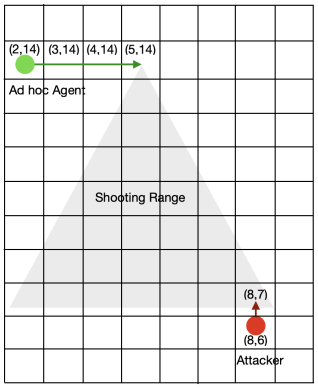

Figure 6: Part of the domain showing the ad hoc guard agent (green) moving to track and shoot an attacker (red).

learning by leveraging the principles of refinement and ecological rationality. The architecture enabled an ad hoc agent to perform non-monotonic logical reasoning with prior commonsense domain knowledge and an ensemble of fast and frugal decision trees learned from limited examples to model the behavior of other agents. In the benchmark fort attack domain, our architecture enabled an ad hoc guard agent to adapt to previously unseen teammates and opponents, rapidly revise the learned models, perform substantially better than baselines that included a state of the art data-driven method, and generate on-demand relational descriptions of its decisions. Future work will explore more complex scenarios with multiple ad hoc agents and partial observability, and use our architecture on physical robots in AHT settings.

Acknowledgments

This work was supported in part by the US Office of Naval Research award N00014-20-1-2390. All conclusions are those of the authors alone. The authors thank Ahmet Baran Ayhan and Haniyeh Ehsani for their help in exploring different heuristic methods, and thank Evgenii Balai for his guidance in making revisions to the SPARC system.

References

Anjomshoae, S.; Najjar, A.; Calvaresi, D.; and Framling, K. 2019. Explainable agents and robots: Results from a systematic literature review. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Montreal, Canada.

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

Balduccini, M.; and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*.

Barrett, S.; Rosenfeld, A.; Kraus, S.; and Stone, P. 2017. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242: 132–171.

Barrett, S.; Stone, P.; and Kraus, S. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *International Conference on Autonomous Agents and Multiagent Systems*, 567–574.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013. Teamwork with Limited Knowledge of Teammates. In *AAAI Conference on Artificial Intelligence*, volume 27, 102–108.

Bowling, M.; and McCracken, P. 2005. Coordination and Adaptation in Impromptu Teams. In *National Conference on Artificial Intelligence*, 53–58.

Chen, S.; Andrejczuk, E.; Cao, Z.; and Zhang, J. 2020. AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. In *AAAI Conference on Artificial Intelligence*, 7095–7102.

Deka, A.; and Sycara, K. 2020. Natural Emergence of Heterogeneous Strategies in Artificially Intelligent Competitive Teams. Technical report, https://arxiv.org/abs/2007.03102.

Dodampegama, H.; and Sridharan, M. 2022. Code and results for the AAAI 2023 paper. https://github.com/hharithaki/HAAHT.

Erdem, E.; and Patoglu, V. 2018. Applications of ASP in Robotics. *Kunstliche Intelligenz*, 32(2-3): 143–149.

Fisher, R. A. 1992. *Statistical Methods for Research Workers*, 66–70. New York, NY: Springer New York. ISBN 978-1-4612-4380-9.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In *IJCAI Workshop on Explainable AI*.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan Claypool Publishers.

Gelfond, M.; and Inclezan, D. 2013. Some Properties of System Descriptions of AL_d. *Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming*, 23(1-2): 105–120.

Gigerenzer, G. 2016. *Towards a Rational Theory of Heuristics*, 34–59. London: Palgrave Macmillan UK.

Gigerenzer, G. 2020. What is Bounded Rationality? In *Routledge Handbook of Bounded Rationality*. Routledge.

Gigerenzer, G.; and Gaissmaier, W. 2011. Heuristic Decision Making. *Annual Review of Psychology*, 62: 451–482.

Katsikopoulos, K.; Simsek, O.; Buckmann, M.; and Gigerenzer, G. 2021. *Classification in the Wild: The Science and Art of Transparent Decision Making*. MIT Press.

Lichtenberg, J. M.; and Simsek, O. 2019. Regularization in Directable Environments with Application to Tetris. In *International Conference on Machine Learning*.

Mirsky, R.; Carlucho, I.; Rahman, A.; Fosong, E.; Macke, W.; Sridharan, M.; Stone, P.; and Albrecht, S. V. 2022. A Survey of Ad Hoc Teamwork: Definitions, Methods, and Open Problems. Technical report, https://arxiv.org/abs/2202.10450.

Mota, T.; Sridharan, M.; and Leonardis, A. 2021. Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics. *Springer Nature CS*, 2(242): 1–18.

Rahman, M. A.; Hopner, N.; Christianos, F.; and Albrecht, S. V. 2021. Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning. In *International Conference on Machine Learning*, 8776–8786.

Ravula, M.; Alkoby, S.; and Stone, P. 2019. Ad Hoc Teamwork With Behavior Switching Agents. In *International Joint Conference on Artificial Intelligence*, 550–556.

Santos, P. M.; Ribeiro, J. a. G.; Sardinha, A.; and Melo, F. S. 2021. Ad Hoc Teamwork in the Presence of Non-Stationary Teammates. 648–660. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-86229-9.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2019. REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. *Journal of Artificial Intelligence Research*, 65: 87–180.

Sridharan, M.; and Meadows, B. 2018. Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration. *Advances in Cognitive Systems*, 7: 77–96.

Stone, P.; Kaminka, G.; Kraus, S.; and Rosenschein, J. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *AAAI Conference on Artificial Intelligence*, 1504–1509.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online Planning for Ad Hoc Autonomous Agent Teams. IJCAI'11, 439–445. AAAI Press. ISBN 9781577355137.

Zintgraf, L.; Devlin, S.; Ciosek, K.; Whiteson, S.; and Hofmann, K. 2021. Deep Interactive Bayesian Reinforcement Learning via Meta-Learning. In *International Conference on Autonomous Agents and Multiagent Systems*.