Ad Hoc Teamwork using Approximate Representations

FILIPE SOUSA, Instituto Superior Técnico, Portugal

The mass production of technological systems around the world is both an economic and ecological issue we face today. It is critical that we find alternate solutions as soon as possible, to work towards a more sustainable society. An emerging field that can bring some advancements towards this goal is that of *ad hoc* teamwork, which studies how an agent can be integrated in a new team without prior knowledge of its new teammates. Such agents would be reusable in future tasks, reducing the need to create such a huge amount of agents. Recent advances in this field shown that it is possible to design agents capable of achieving high performance in this task. However, none of the existing approaches tackled this problem for large domains with partial observability.

In this paper, we present a new algorithm, Partially Observable Plastic Policy (POPP), that combines transfer learning with Deep Recurrent Q-Networks, by having an agent learn policies to play along with different types of teammates, and reusing that knowledge when faced with new teams. We chose the Half-Field Offense domain for evaluation. We experiment with different configurations, with and without partial observability, and with known and unknown teammates. Finally, we present and discuss our results, and compare them to non-recurrent approaches, namely Deep Q-Networks (DQN). We concluded that POPP was able to quickly identify most of the previously known teams, and surpassed the score rate of a DQN approach in partially observable scenarios.

Additional Key Words and Phrases: *Ad Hoc Teamwork*; Multi-agent Systems; Transfer Learning; Function Approximation; Recurrent Neural Networks

ACM Reference Format:
Filipe Sousa. 2022. Ad Hoc Teamwork using Approximate Representations. 1, 1 (October 2022), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the non-stopping progress in the world of computer science during the last few decades, a wide variety of systems were designed and deployed. Many of those systems were custom-tailored to perform a specific task in a specific environment. This means they cannot be redeployed to perform a different task in a different environment without a large cost and effort. Therefore, it would be useful to build a system that was able to dynamically adapt to different tasks, should its physical capabilities allow it to do so. However, the design of an agent with such a generalization capability is not trivial. This challenge can be somewhat simplified, though, if the task is cooperative, since a new agent has its new teammates' behaviour as an additional source of information.

Ad hoc teamwork is a field of study aiming to determine how an agent can be integrated in a group of unknown teammates "on the fly", i.e., without any prior coordination, and with poor or nonexistent communication protocols. In a teamwork task, the agent can observe its teammates' behaviour to determine and adapt to the task they are solving, without the need for humans to specify it manually.

However, a multi-agent scenario brings an additional layer of complexity to the task, since, as there is no pre-coordination between the agent and the new team, the teammates' behaviour is itself another source of uncertainty. A possible simplification of this problem occurs if the agent has worked previously in the same task, but with a different team, since it can try to extrapolate its prior knowledge when working with the new team.

Another hindrance which increases the complexity of the *ad hoc* teamwork problem is when the environment is only partially observable. This is the most realistic scenario, though, since most physical agents' sensors are imperfect (i.e., there can be noise in the data collection), and don't allow for a full representation of the world (e.g.: a 2D camera provides an incomplete representation of a 3D world).

With partial observability comes another problem: if the agent aims to perform optimally (that is, to determine the optimal policy) in a partial observable environment, it must choose how to act considering everything it saw since the beginning of the task, since the most recent observation is not enough to describe the world state. This phenomena is also called history-dependence of the optimal policy.

A final hazard that adds complexity to the *ad hoc* teamwork scenario is the sparsity of reward signals given to the agent. In many real-world tasks, only the final result matters (e.g., an agent exploring a maze), and therefore it is difficult for the agent to know whether it is performing well until the end of the task.

State of the art algorithms in the field of *ad hoc* teamwork have been developed to address some of the aforementioned issues, however, no algorithm has been created that can solve all of these problems simultaneously.

1.1 Research Question

With our work, we aim to address an *ad hoc* teamwork setting which is missing in previous literature, combining:

- **(i):** complex domains, with a continuous, high dimensional observation space;
- **(ii):** partial observability, and subsequent history-dependence of the optimal policy;
- **(iii):** sparse reward signals from the environment.

With this in mind, our research question becomes:

Is it possible to develop an autonomous agent which performs near-optimally in the *ad hoc* teamwork problem, in a complex, partially observable environment with sparse rewards?

Author's address: Filipe Sousa, filipe.miguel.sousa@tecnico.ulisboa.pt, Instituto Superior Técnico, Lisbon, Portugal.

XXXX-XXXX/2022/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.2 Contributions

We contribute to the *ad hoc* teamwork research field with the development of a new algorithm, Partially Observable PLASTIC-Policy (POPP), which uses the agent's experience working with past teams to adapt to previously unseen teammates in a continuous, high dimensional and partially observable environment with sparse rewards. In particular, the partial observability - and inherent history-dependence of the optimal policy - is dealt with using Recurrent Neural Networks (RNNs).

2 BACKGROUND

Before delving into our research problem more deeply, we will describe how we decided to model it. We modeled our problem as an Multi-agent Partially Observable Markov Decision Process (MPOMDP), which is a framework to formalize a sequential decision process under partial observability and uncertainty, targeted to a multi-agent setting. It can be described as a tuple $(I, X, A, P, Z, O, r, \mu_0)$, where:

- I is the index set of agents
- X is the set of environment states
- A^i is the set of actions available for agent i, and $A = \bigtimes_{i \in I} A^i$ is the set of joint actions
- $P : X \times A \times X \rightarrow \mathbb{R}$ is the transition probability, with $P(x, a, x')$ being the probability of the environment evolving to state x' when the team executes the joint action a on state x; $P(x, a, x')$ is more commonly written as $P(x' \mid x, a)$ to highlight its conditional nature
- Z^i is the set of possible observations for agent i and $Z = \bigtimes_{i \in I} Z^i$ is the set of joint observations
- $O : X \times A \times Z \rightarrow \mathbb{R}$ is the observation probability, with $O(x', a, z)$ being the probability of the team seeing the joint observation z when the execution of the joint action a resulted in a transition to state x'; $O(x', a, z)$ is more commonly written as $O(z \mid x', a)$ to highlight its conditional nature
- $r : X \times A \rightarrow \mathbb{R}$ is the immediate reward function, where $r(x, a)$ is the reward for taking the joint action a on state x
- $\mu_0 : X \rightarrow [0, 1]$ is the probability distribution for the initial environment state x_0

The execution of an MPOMDP is carried out as follows: at each time step t (with the environment state being x_t, unbeknownst to the team), each agent i sees an observation z_t^i; then each agent i selects an action a_t^i, resulting in the joint action a_t; upon execution of a_t, the whole team observes a reward r_t and the environment evolves to state x_{t+1}.

In order to select how to act, an agent follows a policy π, which maps the sequence of all transitions since the start of an episode to an action a. The goal of the agent is to find a policy that maximizes its expected reward. A common approach to this problem is Q-Learning, which computes a value for each state-action pair, using the update rule

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t [r_t + \gamma \max_{a' \in A} Q_t(x_{t+1}, a') - Q_t(x_t, a_t)]$$

A more advanced algorithm, which supports states with continuous features is Fitted Q-Iteration (FQI):

Algorithm .1: Fitted Q-Iteration

begin
  $D \leftarrow \{\}$ // The replay buffer
  $w_{0,0} \leftarrow 0$ // The weights $w_{k,t}$ for the estimator at time step t of episode k
  $w^{\odot} \leftarrow w_{0,0}$ // The weights for the target estimator
  $k \leftarrow 0$ // The current episode
  repeat
    $t \leftarrow 0$ // The current time step
    do
      Interact with environment and get transition $T_{k,t} = (x_{k,t}, a_{k,t}, r_{k,t}, x_{k,t+1})$
      $D \leftarrow D \cup \{T_{k,t}\}$ **if** $t \mod P = 0$ **then**
        $N \leftarrow \max(B, |D|)$
        Randomly sample mini-batch $\{(x_n, a_n, r_n, x'_n), n = 1, ..., N\}$
        $w_{k,t+1} = \text{argmin}_{w \in \mathbb{R}^M} \frac{1}{N} \sum_{n=1}^{N} ||y_{k,t,n} - \hat{Q}_w(x_n, a_n)||^2$
        where
        $y_{k,t,n} = r_n + \gamma \max_{a \in A} \hat{Q}_{w^{\odot}}(x'_n, a)$ // The target estimator
      $t \leftarrow t + 1$
      if $t \mod P^{\odot} = 0$ **then**
        $w^{\odot} \leftarrow w_{k,t}$
    while $x_{k,t}$ *is not terminal*
    $w_{k+1,0} \leftarrow w_{k,t}$ // The weights continue to the next episode
    $k \leftarrow k + 1$
  until forever

3 RELATED WORK

In this section we will explore state-of-the-art algorithms used to solve the ad hoc teamwork task. We will start by exploring the first works in this field, and find out how they culminated in the creation of the most important architectures in this field - the PLASTIC architecture - and its two main implementations - PLASTIC-Model and PLASTIC-Policy. Then, we will see how more recent approaches were designed to deal with the partial observability problem. We will finish by examining a competitive alternative to the PLASTIC architecture based on attention networks.

3.1 Early Work

The first work we will address in the field of *ad hoc* teamwork is by Stone et al. [Stone et al. 2010] and it was one of the pioneering texts to recognize the importance of deepening our research on *ad hoc* autonomous agent teams. The *ad hoc* teamwork problem is posed to the community as: "To create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of

contributing as team members." The authors go even deeper, by stating that a robust *ad hoc* team agent should be able to:

(1) "Identify the full range of possible teamwork situations that a complete ad hoc team player needs to be capable of addressing."
(2) "For each such situation, find theoretically optimal and/or empirically effective algorithms for behavior."
(3) "Develop methods for identifying and classifying which type of teamwork situation the agent is currently in, in an online fashion."

Furthermore, they propose a performance evaluation method for an *ad hoc* autonomous agent based on its capability to replace the role of a random existing teammate from a cohesive team, while trying to maximize a certain score measure.

After this, Barrett et al. [Barrett et al. 2011] published a followup work, in which the first empirical evaluation of an *ad hoc* team agent was provided. The domain used was *Pursuit*.

Two planning algorithms are tested: Value Iteration (VI), which we have seen before, and Monte Carlo Tree Search (MCTS). MCTS is an online planning algorithm which works by successively performing simulations in a search tree, to enhance its knowledge about the expected return of each possible sequence of actions. In order to perform those simulations, the algorithm needs to model the uncertainty in the environment, which, in this particular environment, exists both in the prey and the teammates' behavior. The prey, however, is known to act following a uniform distribution over the possible actions. To main challenge is to model the teammates behavior, which is done using Bayes' theorem (assuming the teammates' possible models are previously known by the agent). At each time step t, to estimate the posterior probability $P_t(m \mid a_t)$ of each model m given the joint action a_t, the likelihood of each joint action given a teammate model, $P_t(a_t \mid m)$ is multiplied by the prior distribution over the teammate models, $P(m)$, and divided by $P(a_t)$, which works as a normalization factor:

$$P_t(m \mid a_t) = \frac{P(a_t|m)\dot{P}(m)}{P(a_t)}$$

The evaluation method is adapted from the one in [Stone et al. 2010], with multiple experiments done for different combinations of the planning algorithm, the teammates' behavior and the grid size. The results shown that MCTS allows for efficient planning when compared to VI given that it has access to a known set of teammate models, even if these models are faulty, or the actual models being used by the teammates differ from the ones in the set.

Later, Barrett et al. [Barrett et al. 2012] developed the first *ad hoc* team agent capable of autonomously learning its teammates' models. They describe a novel algorithm based on *transfer learning*, adapted to cases where the observations the agent has about its (potential) teammates are limited. Transfer learning techniques consist in having an agent store the knowledge it acquired in a reinforcement learning task to reuse it in a different one.

The authors use, again, the "Pursuit" domain to test their algorithm, and the method defined in [Stone et al. 2010] to evaluate it. In the construction of the algorithm, they assume the *ad hoc* agent knows the representation of both the environment and the prey, but not that of its teammates.

The planning algorithm used is Upper Confidence bound applied to Trees (UCT), which is an MCTS algorithm that has been shown to be effective in complex domains, where the branching factor is high. To perform each simulation, the agent must make an assumption about its teammates' models, and it does so using a Bayesian approach. To select the set of models, multiple approaches are followed, but the most relevant for our case is the one where transfer learning is used. The authors empirically concluded that this transfer learning approach can enhance the *ad hoc* agent's performance even in cases where there is a small amount of data available.

3.2 The PLASTIC architecture

The Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC) architecture, by Barrett [Barrett 2015], is one of the most popular methods for *ad hoc* teamwork, due to its robustness to task and teammate diversity, and its capability to efficiently adapt to those diverse situations. It is an algorithm which assumes the *ad hoc* agent has past experience cooperating with other teammates. When faced with new teammates, the agent tries to identify the past team with a highest similarity to the current one and reuses the knowledge it got from the most similar past team to act upon the domain.

In order to better leverage information from different data sources, the author introduced a new transfer learning algorithm, "Two-Stage Transfer" (used by the PLASTIC algorithm), which tries to identify the best weighing for the importance of each data source.

The author introduces two variants of the PLASTIC architecture, which we will proceed to describe, called PLASTIC-Model and PLASTIC-Policy.

The PLASTIC-Model algorithm starts by learning transition models for a set of teammates using a supervised learning approach, i.e., an approach where the aim is to predict a target output variable given an input, after being presented with a sequence of (input, output) tuples. In PLASTIC-Model's case, the inputs are the features extracted from environment states, and the outputs are the teammates' actions. The agent performs this learning offline, thus representing its past knowledge about these teammates.

Then, these learnt transition models are combined with hand-coded models from other data sources using their novel "Two-Stage Transfer" algorithm. The resulting models are then used by the *ad hoc* agent's online planner to plan the best course of action, using the UCT algorithm we mentioned before.

To select which model is used by the planner, the agent stores a belief distribution over which of the teammate models is more likely to be the one the agent is currently working with. Since, in the presence of previously unseen teammates, this belief distribution represents, instead, the similarity between the currently observed model and the known ones, they named it *behavior distribution*, which covers both the cases of seen, and unseen teammates.

The behavior distribution for each model m is then updated using polynomial weights, as follows:

$$L(m, x, a, x') = 1 - P(a \mid x, m, x')P(m \mid x, a, x') = \frac{(1 - \eta * L(m, x, a, x'))}{c} \quad (1)$$

where $L(m, x, a, x')$ is the loss of model m for observing the transition (x, a, x'), $\eta \leq 0.5$ is a parameter bounding the maximum value for the loss, and c is a normalization constant.

Empirical evaluation shown that PLASTIC-Model can quickly adapt to previously unseen teammates in simple domains. The two main weaknesses of this algorithm are (i) that it assumes the agent has access to its teammates' actions (since it observes the joint actions a), and (ii) that UCT, being a stochastic algorithm, is prone to learning inaccurate environment models. On the other hand, more accurate planning algorithms would most likely be way too ineffective to be used online in larger domains.

3.2.1 PLASTIC-Policy. The second, and last variant we will see is PLASTIC-Policy.

Instead of learning teammate models, PLASTIC-Policy starts by directly learning policies to work with a given set of teammates, removing the hurdle of online planning. PLASTIC-Policy represents these policies in the form of a matrix of Q-values which is learnt using an FQI implementation, similar to the one we described in Algorithm .1. The estimator used by the author consists in a weighed sum of a set of binary features f_i extracted from the environment state:

$$\hat{Q}(x, a) = \sum_{i=1}^{M} w_i f_i$$

where the weights w_i are learnt by FQI and M is the number of features f_i. To build the replay buffer for each teammate in the known set, the agent performs exploratory actions in the domain, and stores in it the resulting transitions of the form (x, a, r, x'). Note that, since the agent does not need to predict its teammates' actions, a can simply be the agent's own action.

In parallel to learning the policies, the agent also learns a nearest neighbor model for each past teammate, which maps states to the next state whose previous state is the closest to the state to be mapped. We explain this in more detail in Section 4.

The agent then combines this knowledge with that from other sources using, again, the "Two-Stage Transfer" algorithm. The resulting policies are then used to act in the environment, and, to update which policy the agent follows, an analogous approach to that of PLASTIC-Model is followed.

PLASTIC-Policy was tested by Barrett et al. [Barrett and Stone 2015] in the Half-Field Offense (HFO) domain - the same we used for our experiments - making this work a very important baseline for us to follow.

After experimenting in scenarios with 2 attacking versus 2 defending agents and with 4 attacking versus 5 defending agents, the authors concluded that the algorithm quickly converged to the correct policy in both scenarios. Due to its scalability, its capability for quick adaptation to new teammates, and its portability for different *ad hoc* teamwork scenarios, PLASTIC-Policy has become one of the most popular approaches in the field.

3.3 Other architectures for the *ad hoc* teamwork problem

Chen et al. [Chen et al. 2020] developed an attention network algorithm that surpassed PLASTIC-Policy's performance in the HFO domain. The algorithm was named by the authors Achieving the Ad-hoc Teamwork by Employing the Attention Mechanism (AATEAM) and, like POPP uses RNN. However, unlike the algorithms we previously seen, AATEAM is designed to adapt to the *ad hoc* agent's new teammates in real-time, with the aid of attention-based RNN. The architecture consists of multiple networks, one for each previously known teammate type, and each network consists, mainly, of two parts, called an "encoder" and a "decoder".

The encoder, at each time step, receives a sequence with the most recent environment states and outputs an encoded value for each state. Each of these values has in consideration the input from the few previous states, since the encoder has a layer with a hidden state. The decoder uses the information outputted by the encoder, together with its hidden state, to output an action. This description is only a simplified version of the actual algorithm used in the paper. Its complete details are out of the scope of this work. Experimental results shown that AATEAM greatly over-performed PLASTIC-Policy both with known and unknown teammates.

4 THE PLASTIC-POLICY ARCHITECTURE

In this section, we will present the PLASTIC-Policy architecture in greater detail, since it will be the basis for our novel architecture. An overview of PLASTIC-Policy can be found in Figure 1. but here we present a more thorough explanation.

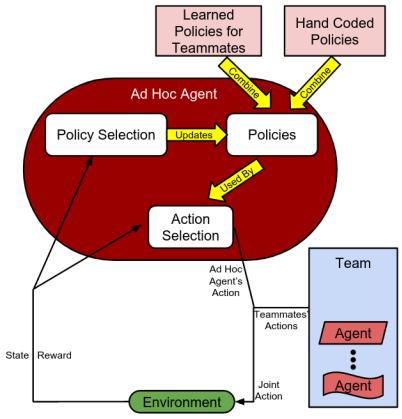


Fig. 1. Overview of the PLASTIC-Policy algorithm. Source: Adapted from [Barrett 2015].

Algorithm .2 is divided in three parts: LearnPolicies, LearnNNModels and ActInDomain. We will proceed to explaining each one of them.

4.1 LearnPolicies

First, the agent learns a set of policies Π to work with the previously encountered KnownTeammates. It does so using the FQI algorithm, as explained in Algorithm .1. We will specify the type of estimator used in Section 4.4.

Algorithm .2: The PLASTIC-Policy algorithm. Source: Adapted from [Barrett and Stone 2015].

Procedure PLASTIC-Policy(*KnownTeammates*):
  $\Pi \leftarrow$ LearnPolicies (KnownTeammates)
  $M \leftarrow$ LearnNNModels (KnownTeammates)
  ActInDomain (KnownTeammates, Π, M)

Function LearnPolicies(*KnownTeammates*):
  $\Pi \leftarrow \{\}$
  foreach *teammate* $\beta \in$ *KnownTeammates* **do**
    Learn policy π to cooperate with β
    $\Pi \leftarrow \Pi \cup \{\pi\}$
  return Π

Function LearnNNModels(*KnownTeammates*):
  $M \leftarrow \{\}$
  foreach *teammate* $\beta \in$ *KnownTeammates* **do**
    Learn nearest neighbor model m_{NN} of β
    $M \leftarrow M \cup \{m_{\text{NN}}\}$
  return M

Procedure ActInDomain(*KnownTeammates,* Π, M):
  μ = UniformDistribution(KnownTeammates)
  Initialize state x
  while x *is not terminal* **do**
    β = argmax μ
    Take action $a = \Pi_\beta(x)$ and observe r, x'
    μ = UpdateBehaviorDistribution (KnownTeammates, Π, M, μ, x, a, x')

Function UpdateBehaviorDistribution(*KnownTeammates, M, μ, x, a, x′*):
  foreach *teammate* $\beta \in$ *KnownTeammates* **do**
    $L(M_\beta, x, a, x') = 1 - P(a \mid x, M_\beta, x')$
    $\mu_\beta = \mu_\beta(1 - \eta L(M_\beta, x, a, x'))$
  Normalize μ
  return μ

4.2 LearnNNModels

Then, the agent will learn a set of nearest neighbor models M for each teammate in KnownTeammates. Learning a model consists in interacting with the environment whilst cooperating with a known teammate β, storing tuples of the form (x_t, x_{t+1}) in M_β.

4.3 ActInDomain

This is the core of the PLASTIC-Policy algorithm. Here, the agent is placed in an *ad hoc* scenario, and must cooperate with the new teammate (which might be known or unknown), leveraging the previously acquired knowledge. The agent starts by initializing a behavior distribution vector μ to a uniform distribution over the known teammates (since in our work, for simplicity, we assume the teammates follow a uniform distribution). Each entry in this vector represents how similar the current teammate's behavior is to each previously seen teammate. The agent then starts interacting with the environment, and, at each iteration, starting in state x, takes the action a prescribed by the policy corresponding to the previously encountered teammate with the highest belief distribution (in case of a tie, it selects one at random among the tied ones), and observes r and x'. With this information, it will then proceed to updating the behavior distribution μ, and it does so, using the polynomial weights update we described in Equation (1). The term $P(a \mid x, M_\beta, x')$ is computed as we described in Section 3.2.1.

4.4 PLASTIC-Policy with a DQN

In order to compare our performance with a similar, but non-recurrent approach, we implemented a version of PLASTIC-Policy using a DQN as the estimator for the FQI algorithm, and called it Deep Q-Network - PLASTIC-Policy (DQN-PP).

4.5 Introducing Recurrence in PLASTIC-Policy: POPP

We finally reach the point were we explain our novel algorithm, POPP. In essence, it consists in an implementation of PLASTIC-Policy using a Deep Recurrent Q-Network (DRQN) as the estimator for FQI. But there are some implementation details that are worthy to note.

Firstly, to choose the action during the LearnPolicies part of the algorithm, we used an ϵ-greedy policy, with ϵ being an experimental parameter.

Secondly, we used the replay buffer to store the transitions that the agent would then use to learn the nearest neighbor models (in function LearnNNModels). We could only do this, since we realized the replay buffer was never totally filled up after the end of the LearnPolicies function. In this way, we saved memory space, by avoiding the use of an additional buffer to store the $(x_t, x_{t+1}$ tuples.

Thirdly, since the environment we chose (HFO) has different valid actions for different environment states (e.g., an agent can only pass the ball to a teammate, or shoot the ball if it is currently controlling it), we changed the agent's action selection mechanism. When the agent is selecting a random (exploratory) action, it selects instead a random action among the valid ones for that state. When the agent is following the greedy action, it selects the one with the highest Q-value from among the valid ones. In this way, we prevent the agent to select invalid actions, which would hinder the learning process.

Last, but not least, since with partial observability we cannot access the full state of the environment, we use the observations $z \in Z$, instead of the states $x \in X$ to create the nearest neighbor model for each teammate.

5 EXPERIMENTAL QUESTIONS

We define a set of questions to be answered based on our experimental results:

- **1.** Can DRQN surpass a non-recurrent (DQN) architecture's performance in a complex, totally observable scenario with sparse rewards?
- **2.** Can DRQN surpass a non-recurrent (DQN) architecture's performance in a complex, partially observable scenario with sparse rewards?
- **3.** How accurately can POPP identify teams it previously encountered?
- **4.** Can POPP surpass a non-recurrent (DQN-PP) architecture in a complex, partially observable scenario with sparse rewards?
- **5.** How does POPP's performance change with the level of similarity between the current and past teammates?
- **6.** How does POPP's performance compare with that of the original teams?

6 HALF-FIELD OFFENSE

The environment we chose to evaluate our agent was HFO, which is a task that plays out in the offense half of a soccer field, where a offense team must coordinate to score a goal before time runs out, and a defense team must coordinate to prevent the offense team to score. We used the HFO implementation by Hausknecht et al. [Hausknecht et al. 2016]. Figure 2 presents a snapshot of an HFO match.

Fig. 2. Snapshot of a Half-Field Offense ongoing match. The yellow (offense team) is trying to score against the red and purple (defense) team. Source: Primary.

An HFO match ends in one of the following terminal states:

- **Goal**, if the ball enters the defense team's goal;
- **Captured by Defense**, if an agent of the defense team manages to get control of the ball;
- **Out of Bounds**, if the ball leaves the playing field through one of the four lines limiting it;
- **Out of Time**, if a set amount of time steps have passed without any of the previous terminal states being reached.

If the terminal state is "Goal", the offense team is considered the winner; otherwise, victory goes to the defense team.

6.1 Environment Parameterization

In HFO there are multiple parameters that impact the way the game plays out. In Table 1, we listed some of those parameters along with the values we chose for each of them.

Table 1. Description and chosen values for the parameters of the HFO environment

Parameter	Value
--frames-per-trial	500
--untouched-time	100
--no-sync	True

6.2 Environment Model

The HFO environment can be modeled as an MPOMDP $(I, X, A, P, Z, O, r, \mu_0)$. We will provide a high-level description for each of its components in the next few sections.

6.2.1 Agents. The index set of agents I contains indices representing all agents from the offense and defense teams. In all experiments we will present in this work, we considered 2 agents in each team, so, we have always that $|I| = 4$. We can classify the agents in I regarding their role, or their policy.

Regarding each agent's role, we can split I in two subsets, such that $I = I^{\omega} \cup I^{\delta}$, where I^{ω} contains the offense agents and I^{δ} the defense agents.

Regarding their policies, each agent is from one of the following types:

- Learning agents (DQN/DRQN), which we will call γ;
- *Ad hoc* agents, which we will call α;
- Agents for which the binary files were made available, which include those created for the RoboCup 2D Simulation League 2013 ("aut", "axiom", "cyrus", "gliders" and "helios"), and the type "base" (also known as "Agent2D"), that comes with the HFO simulator. We will call them *binary agents*, and index them as $\beta_k, k \in \mathbb{N}$, since in all our experiments there are multiple binary agents.

We made three types of experiments:

- **Learning + Binary:** $I^{\omega} = \{\gamma, \beta_2\}$;
- ***Ad hoc* + Binary:** $I^{\omega} = \{\alpha, \beta_2\}$;
- **Binary + Binary**: $I^{\omega} = \{\beta_1, \beta_2\}$;

In all our experiments, the 2 defense agents ($I^{\delta} = \{\beta_3, \beta_4\}$) were of the binary type "base".

6.2.2 State. The state of an HFO match is represented by multiple features, namely the position, velocity and angle of each entity (agents or the ball), and other minor features like whether each agent is colliding with a landmark (e.g. a goal post).

6.2.3 Actions. We chose the following set of actions for our custom agents (learning and *ad hoc* agents): Intercept, Move, Shoot, Pass and Dribble. The actions Intercept() and Move() can only be executed when the agent in unable to kick the ball, whilst Shoot(), Pass(n) and Dribble() can only be executed when the agent is able to kick the ball. Whenever an agent selects an action that cannot be executed

given the current environment state, that action has no effect, being equivalent to the agent choosing the action NoOp(), which does nothing.

Due to the sparsity of rewards (only in terminal states), we realized the agents γ were having trouble learning a policy. Therefore, we developed a way to reduce the number of time steps the learning agents see during an episode, by changing the agent's actions Intercept() and Move() to Repeat[Intercept(), N_1] and Repeat[Move(), N_2], which means that, when choosing one of these actions, the agent will execute them N_1 or N_2 times, respectively, and will not observe intermediate states.

6.2.4 Transition Probabilities. The transition probability function P results from the combination of the physics simulated by HFO with the join actions chosen by the agents.

We consider that a state transition begins right after an agent starts executing action, and ends when the agent reaches the next state after finishing executing its action, or when it reaches a terminal state. Therefore, given a transition (x, a, r, x'), starting at time step t_1 and ending at time step t_2 we have that:

- $x = x_{t_1}$ is the agent's original state in time step t_1;
- $a = a_{t_1}$ is the action the agent chose at time step t_1, and that lasted $t_2 - t_1$ time steps;
- $r = r_{t_2-1}$ is the reward the agent get for transitioning to state x_{t_2};
- $x' = x_{t_2}$ is the agent's new state in time step t_2.

6.2.5 Observations. As we discussed in Section 6.2.3, we chose to use the High Level State Feature Set as the agents' observations. The features in this set include not only simpler values, like each entity's position in the field (namely that of all agents and the ball), but also some complex to compute, but meaningful values, like the goal opening angle we described before.

For our learning and *ad hoc* agents, we empirically chose to use the feature set containing all useful features, plus one Boolean validity feature per entity (including the ball and excluding the agent itself), indicating whether or not all of the other entities' features are valid. If all features that describe an entity are valid, its corresponding validity feature takes the value 1, otherwise it takes the value -1. By default, HFO sets the value of invalid features to -2, but since that could damage our learning agent's learning process (since it could interpret -2 as a valid, but very low value for that variable), we decided to set invalid features to the value 0, and introduce the validity features we just described.

6.2.6 Observation Probabilities. Like P, the observation probability function O also results from the combination of the physics simulated by HFO with the joint actions chosen by the agents. For instance, if an opponent is blocking our agent's view of the ball or of other agent, the obstructed entity's features in our agent's observation will be invalid. However, there is an option in the HFO simulator (--fullstate), that makes agents able to access the correct values for all features. Yet, since our feature set does not contain the entities' velocities, there is still some degree of partial observability in the experiments we label as "totally observable".

6.2.7 Reward Function. We empirically chose the following reward function r:

- 0 on all time steps where the state is not terminal;
- 10 on goal;
- -10 otherwise, i.e., if the defending team catches the ball, the ball goes out of bounds or time runs out.

6.2.8 Distribution of the Initial State. Regarding μ_0, the offense agents and the ball start in random positions in the left half of the playable field; the defender starts in a random position in the right half of the field; the goalkeeper starts in the center of the goal.

7 LEARNING AGENT CONFIGURATION

In Table 2, we present the parameters we have empirically chosen for both the DQN and the DRQN agents.

Table 2. Parameters chosen for the DQN and the DRQN

Parameter	DQN value	DRQN value
Input Size	25	25
Output Size	5	5
Number of Hidden Layers	1	1
Number of Units Per Layer	12	12
Type of Units	Linear	LSTM
Activation Function	ReLU	ReLU
Optimizer	Adam	Adam
Learning Rate	0.00025	0.00025
Initial Exploration Rate	0.5	0.5
Final Exploration Rate	0.05	0.05
Initial Exploration Rate	0.5	0.5
Discount Factor	0.995	0.995
Estimator Update Period	4	4
Target Estimator Update Period	500	500
Replay Buffer Sequence Length	–	10

8 PROCEDURE AND METRICS

In this section, we present our evaluation procedure and chosen metrics for our three types of experiments: Learning + Binary, *Ad hoc* + Binary and Binary + Binary.

8.1 Learning + Binary

In this type of experiments, where $I^\omega = \{\gamma, \beta_1\}$, we aimed to evaluate the learning agent γ's capacity to learn a policy to cooperate with a binary agent β_1. For each configuration (learning agent type, teammate type and observability), we ran the agent for 80 rollouts of 500 training episodes and 50 test episodes each, adding up to a total of 40,000 train episodes and 4,000 test episodes per experiment.

In the training episodes, γ learned using the DQN and DRQN algorithms.

In the test episodes, the agents did not learn, and always chose greedy actions (the valid actions with the highest Q-values).

At the end of each rollout, we saved the agent's current neural network weights and replay buffer (to reuse in the *ad hoc* experiments, and measured the offense team's score rate in the test episodes, i.e., the fraction of test episodes from the 50 of each rollout that

ended in a goal. The results for these experiments can be found in Section 9.1.1 and Section 9.1.2.

8.2 *Ad hoc* + Binary

In this type of experiments, where $I^{\omega} = \{\alpha, \beta_1\}$, we aimed to evaluate our novel POPP algorithm. For each configuration (type of learning agent, type of the teammates the agent will find, observability), we followed the POPP algorithm as described in Section 4.5:

- we started by the policies learnt when cooperating with each known teammate (which were generated using the procedure defined in Section 8.1). Since, during each experiment, we save the agent's network and replay buffer, we had to choose which state to use. In order to achieve a balance between the policy's performance and the number of observed time steps, we chose the state with the highest score rate among the last 5 states;
- we then learnt a nearest-neighbor model for each known teammate, using the transitions stored in the replay buffer (which are all of the transitions the agent has seen, since the replay buffer never exceeded its maximum capacity);
- finally, we tested the agent in an *ad hoc* scenario, with either known, unknown, or both types of teammates. For each configuration, we ran 1000 trials of 25 episodes each.

The results for these experiments can be found in Section 9.2.

8.3 Binary + Binary

In this type of experiments, where $I^{\omega} = \{\beta_1, \beta_2\}$, we aimed to determine each binary team's average score rate, to compare their performance with that of POPP. We only considered cases where β_1 and β_2 were of the same type, since we wanted to determine the performance of the original teams.

We ran each team during 1,000 episodes, and calculated their average score rate over the course of all 1,000 episodes. We also calculated the average score rate for the set of known teams, for the set of unknown teams and for the set of all teams. The results for these experiments can be found in Section 9.2, to be compared with those for POPP.

9 RESULTS

In this section, we describe the results for the experiments defined in Section 8. To represent the standard deviation σ of a sample in the plots, we used 95% confidence intervals ($[\mu - \frac{1.96\sigma}{\sqrt{n}}, \mu + \frac{1.96\sigma}{\sqrt{n}}]$, with n being the sample size, and μ the mean value of the sample).

9.1 Learning a Policy in HFO

In this section we present the results for the experiments of the type "Learning + Binary". In each plot, each point corresponds to the average of 3 experiments as defined in Section 8.1, one for each of the following teammate types: "aut", "base" and "helios".

9.1.1 Total Observability. In Figure 3, we present the experimental results for the DQN and the DRQN with the --fullstate option of the HFO set to **True**.

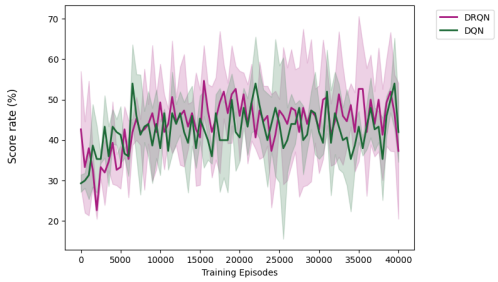


Fig. 3. Score rate for the DQN and DRQN learning agents playing with total observability. Source: Primary.

9.1.2 Partial Observability. In Figure 4, we present the experimental results for the DQN and the DRQN with the --fullstate option of the HFO set to **False**.

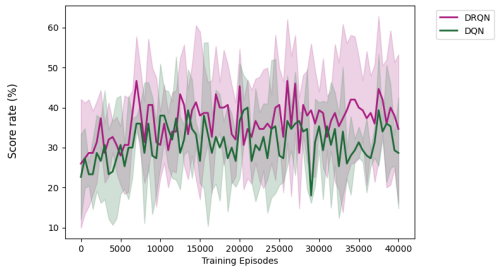


Fig. 4. Score rate for the DQN and DRQN learning agents playing with partial observability. Source: Primary.

9.2 *Ad hoc* teamwork in HFO

In this section, we present the results for our agents playing in an *ad hoc* scenario (experiments of the type "*Ad hoc* + Binary", as explained in Section 8.2), and compare them to the binary agents using their original policy (experiments of the type "Binary + Binary" Section 8.3). In all experiments, the teams the *ad hoc* agents previously knew were "aut", "base" and "helios".

In Section 9.2.1 and Section 9.1.2, the score rates presented for DQN-PP and POPP are the average of the score rates among 1000 trials. In each trial, the agent's teammate was uniformly chosen from the following types: "aut", "base", "helios", "axiom", "cyrus" and "gliders". The score rate for the binary agents is the average of the score rates of the original teams (teams with two identical agents) for the 6 types the *ad hoc* agent can encounter, where each team was run for 1000 trials.

9.2.1 Total Observability. In Figure 5, we present the experimental results for DQN-PP, POPP and binary agents with the --fullstate option of the HFO set to **True**.

9.2.2 Partial Observability. In Figure 6, we present the experimental results for DQN-PP, POPP and binary agents with the --fullstate option of the HFO set to **False**.

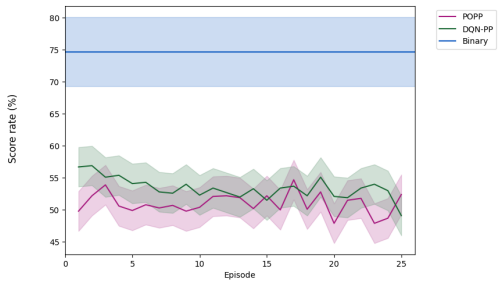


Fig. 5. Score rate for DQN-PP, POPP and binary agents playing with total observability. Source: Primary.

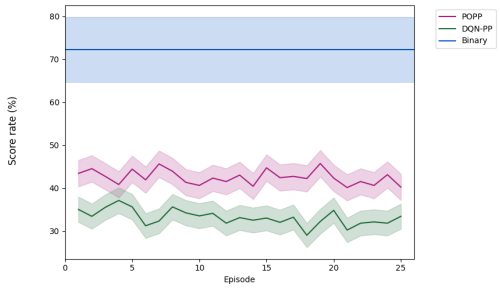


Fig. 6. Score rate for DQN-PP, POPP and binary agents playing with partial observability. Source: Primary.

9.2.3 Varying Ad Hoc Teammates with Partial Observability. In Figure 7, we present and compare the results for POPP while playing with known, and with unknown agents. with the --fullstate option of the HFO set to **False**. Each plot corresponds to the average of 1000 trials. During each trial, the agent paired up with an "aut", "base" or "helios" teammate in the "POPP (known teams)" experiment, and with an "axiom", "cyrus" or "gliders" teammate in the "POPP (unknown teams)" one. In each trial, the teammates were chosen uniformly at random from their sets.

The plots for the "Binary (known teams)" and "Binary (unknown teams)" represent the average over 1000 trials for the score rates of the original teams for the 3 types the *ad hoc* agent encountered in the "POPP (known teams)" and "POPP (unknown teams)" experiments, respectively.

We also present Figure 8, which shows how POPP's behavior distribution evolved over the course of 25 episodes, averaged over 1000 trials.

10 RESULTS DISCUSSION

We will discuss our experimental results by answering the questions we posed in Section 5.

Question 1. *Can DRQN surpass a non-recurrent (DQN) architecture's performance in a complex, totally observable scenario with sparse rewards?*

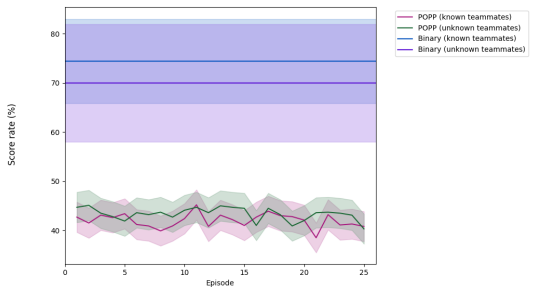
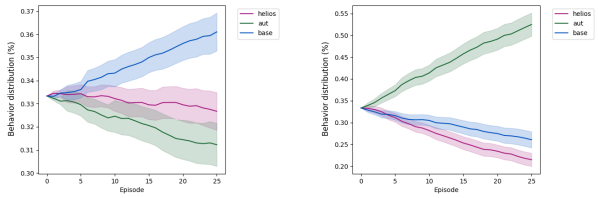
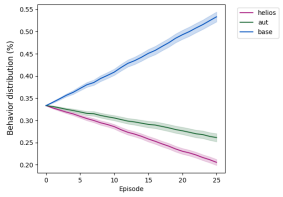


Fig. 7. Score rate for POPP playing with known and unknown teammates, and for the original binary teams. Source: Primary.



(a) Playing with "helios" teammate **(b)** Playing with "aut" teammate



(c) Playing with "base" teammate

Fig. 8. Behavior distribution for POPP when playing with known teammates. Source: Primary.

According to the results in Figure 3, the DRQN does not have a clear advantage over the DQN with full observability, which was expected, since the agents' observations contain almost all information that could be useful to decide how to act. Still, the DRQN had more frequently performance peaks than the DQN, and we conjecture that might have happened mainly due to two factors: (i) the DRQN being able to extract information about entities' speed, and (ii) the the teammate and opponents' policies being potentially non-stationary.

Question 2. *Can DRQN surpass a non-recurrent (DQN) architecture's performance in a complex, partially observable scenario with sparse rewards?*

According to the results in Figure 4, the DRQN surpassed the DQN with partial observability, which was also expected, as, in this case, knowing the history of the last few states constitutes a great advantage for the DRQN.

Question 3. *How accurately can POPP identify teams it previously encountered?*

According to Figure 8, there is some variability on how accurately POPP can identify previously encountered teammates. In Figure 8(b) and Figure 8(c), after 25 episodes, POPP identifies the correct team more than half of the times. However, by observing Figure 8(a), we can see that, when playing with team "helios", the agent has a nearly uniform distribution over the three policies even after the 25 episodes. Moreover, it considers team "base" to be the most likely teammate it is playing with.

We observed the POPP agent playing with the three teams, and realized that (i) the "base" teammate is very unpredictable, when compared with the other two, and (ii) the "base" teammate has a strategy that is more similar to that of the "helios" teammate than to that of the "aut" teammate. Our realization (ii) might explain why "base" has a higher behavior distribution than "àut" when playing with "helios", but it does not explain why "helios" does not have the highest behavior distribution. However, due to (i), the "base" teammate must have spanned a much wider range of environment transitions during the policy learning process, when compared with the other two teammates. Thus, when cooperating with "helios", the POPP agent has a higher chance of finding a transition previously encountered when cooperating with "base", due to its wider range of transitions.

Question 4. *Can POPP surpass a non-recurrent (DQN-PP) architecture in a complex, partially observable scenario with sparse rewards?*

By observing Figure 4, we can see that POPP clearly surpassed DQN-PP's performance. Moreover, their 95% confidence intervals do not intersect after episode 4, which strongly endorses our claim.

Question 5. *How does POPP's performance change with the level of similarity between the current and past teammates?*

POPP has shown to be resilient to the lack of knowledge about the teammates it encounters, since, according to Figure 7, its performance in the presence of known and unknown teammates is very similar. In fact, its performance in the presence of unknown teammates seems to be slightly higher than in the presence of known teammates, even though the known teammates' original policies had a slightly higher score rate than that of the unknown teammates. This seems to indicate that the POPP successfully transferred its learning from previous tasks to new, unseen teamwork situations.

Question 6. *How does POPP's performance compare with that of the original teams?*

POPP performed poorly when compared with the original teammates' policies. We believe that, in part, this is due to original teammates having complex policies that use more than the 5 actions we allowed the POPP agent to use. Adding to that, some teams use communication protocols (that are available in the HFO simulator), that render useless when used with our agent, since it does not support them.

We also noticed, by observing the agents playing in real-time, that POPP's learnt policies are usually conservative, consisting in passing the ball to the teammate whenever possible. In this way, it is difficult for the team to have effective offense strategies, other than the cases where POPP's teammate is well positioned to score a goal. We believe that this behavior is due to, under partial observability, some High Level actions (especially Shoot()) fail very often. This makes sense, since they implement complex behavior that depends on many features in the observation, which, when unavailable, prevent the agent from executing the action.

With this, we finished answering all 6 questions we posed in section Section 5.

We will now answer our main research question.

Is it possible to develop an autonomous agent which performs near-optimally in the *ad hoc* teamwork problem, in a complex, partially observable environment with sparse rewards?

Considering the most realistic situation, where the agent can find either known or unknown teammates (Figure 6, POPP achieved a performance of about 42% of goals scored, while, in average, the original teams scored goals in 72% of episodes. This is a large gap, so we cannot say that POPP's performance was near optimal. Yet, since we argued that there was still plenty of room for improvement, namely due to the performance gap between POPP and a more custom-tailored architecture (AATEAM), the issue with some actions failing very often, and the difficulty in identifying the correct teammate when one of the known teammates spans a wide range of transitions, we strongly believe it is possible to achieve a near-optimal performance in this scenario.

REFERENCES

Samuel Barrett. 2015. *Making Friends on the Fly: Advances in Ad Hoc Teamwork*. Studies in Computational Intelligence, Vol. 603. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-18069-4

Samuel Barrett and Peter Stone. 2015. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. *Proceedings of the AAAI Conference on Artificial Intelligence* 29, 1 (Feb. 2015). https://doi.org/10.1609/aaai.v29i1.9428

Samuel Barrett, Peter Stone, and Sarit Kraus. 2011. Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain. In *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*. Taipei, Taiwan, 567–574.

Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenveld. 2012. Learning Teammate Models for Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*. Citeseer, 57–63. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.6384&rep=rep1&type=pdf

Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. 2020. AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 05 (April 2020), 7095–7102. https://doi.org/10.1609/aaai.v34i05.6196 Number: 05.

Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. 2016. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, Vol. 3. sn.

Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. *Proceedings of the AAAI Conference on Artificial Intelligence* 24, 1 (July 2010), 1504–1509. https://doi.org/10.1609/aaai.v24i1.7529

Received 31 October 2022