Teaching and Leading an Ad Hoc Teammate: Collaboration without Pre-Coordination

Peter Stone[a], Gal A. Kaminka[b], Sarit Kraus[b,c], Jeffrey S. Rosenschein[d], Noa Agmon[a,b]

[a] *The University of Texas at Austin*
[b] *Bar Ilan University*
[c] *The University of Maryland*
[d] *Hebrew University*

Abstract

As autonomous agents proliferate in the real world, both in software and robotic settings, they will increasingly need to band together for cooperative activities with previously unfamiliar teammates. In such *ad hoc team* settings, team strategies cannot be developed a priori. Rather, an agent must be prepared to cooperate with many types of teammates: it must collaborate without pre-coordination. This article defines two aspects of collaboration in two-player teams, involving either simultaneous or sequential decision making. In both cases, the ad hoc agent is more knowledgeable of the environment, and attempts to influence the behavior of its teammate such that they will attain the optimal possible joint utility.

Keywords: autonomous agents, multiagent systems, teamwork, game theory, k-armed bandits

1. Introduction

Imagine that you are called to participate in a search and rescue scenario, where the robots you designed are supposed to help search for survivors in a major earthquake. Alternately, suppose you are part of a Mars exploration party, where your rover is sent (as part of a team) to explore the planet. In both cases, you already deployed an old robot you designed years ago for the mission, and you want also to use a new robot built by someone else, that has more information about the environment (but perhaps has poor actuators). These two robots were designed by different parties in different decades, thus cannot communicate directly and do not use the same coordination protocols. Will you be able to utilize the information gained by the newly designed robot to make the robots perform better as a team in their mission?

This scenario is an example of an *ad hoc team* setting. Multiple agents (in this case robots) with different knowledge and capabilities find themselves in a situation such that their goals and utilities are perfectly aligned (effectively, everyone's sole interest is to help find survivors), yet they have had no prior opportunity to coordinate. In addition to the setting described above, ad hoc teams may arise among any robots or software agents that have been programmed by different groups and/or at different times such that it was not known at development time that they would need to coordinate.

This article focuses on the subclass of such settings in which we are designing a new agent that has full information about its environment, that must coordinate with an older, less aware, more reactive agent whose behavior is known. Let A be the *ad hoc* agent that we control and design, and has full information about the environment. Let B be the agent that we *cannot* control, that adapts to the environment as it perceives it, i.e., it chooses its next action based on what it observed in the environment (mainly, the actions of its teammate). A main question that arises is: can A's information be used to influence B to perform actions leading to higher team utility? Given that this is an ad hoc teamwork setting, B can be assumed to choose actions that it believes to be optimal for the team—based on its limited view of the world. However, these actions might result in poor team utility in the long run. For example, the robot with limited information about the search and rescue environment will choose to help recover one person it can sense, but will disregard numerous people it cannot currently observe.

While designing the ad hoc agent A, its behavior as an ad hoc agent must be adept at assessing the capabilities of other agents (especially in relation to its own capabilities), it must also be adept at assessing the other agents' knowledge states, and must be proficient at estimating the effects of its actions on the other agents.

In this article we address two repeated decision making settings for ad hoc agents.

1) *Simultaneous decision making*, in which agents A and B make their decisions at the same time. In this case, agent A could *lead* agent B to perform actions resulting in long-term higher team utility. This interaction between the agents is modeled using game theoretic tools, specifically, by a matrix game representation.

2) *Sequential decision making*, in which agent B selects its action after observing the outcome of A's (and possibly its own past) actions. Here, the actions chosen by agent A can *teach* agent B of the optimal action it should choose, yielding highest possible team utility in the long run. In this case, we model the interaction between the agents by a novel cooperative k-armed bandit formalism.

In both cases we can directly control the behavior of agent A, and by choosing appropriate actions this agent (indirectly) influences the behavior of agent B, whose decision-making algorithm is assumed to be known and reflect its assumption that the environment (specifically, agent A) will continue to perform similarly to what was observed so far. Computing the optimal behavior for the ad hoc agent A is done using dynamic programming algorithms, for both leading and teaching agent B. In both cases the agent's goal is the same - maximize team utility, where the utility is computed as the sum of payoffs gained by performing each action (joint action in simultaneous play, or individual actions in sequential play).

The remainder of this article is organized as follows. Sections 2 and 3 introduce detailed theoretical analysis of these ad hoc teamwork problems. First, in Section 2, we examine the case of leading ad hoc teams, in which the two agents act repeatedly and simultaneously in a situation appropriately captured by iterated matrix games. Second, in Section 3, we turn to the teaching in ad hoc teams, a scenario in which the agents alternate in their turns to make decisions, as can be captured by a novel cooperative k-armed bandit formalism. Section 4 discusses prior research most related to our specific studies and the ad hoc teamwork problem itself; and Section 5 discusses the results in the broad perspective of the general problem of ad hoc teamwork and concludes.

1.1. Problem Scope and Motivation

The challenge of ad hoc teamwork, as presented in the ad hoc teamwork introductory paper [1], is:

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.

In this article, we analyze the simplest, and in some sense most basic and fundamental, special case of the ad hoc teamwork problem.To this end, we strip away as much complexity as possible while still retaining the most essential feature of ad hoc teamwork, namely that an individual agent must determine on the fly how to cooperate with at least one other teammate. Specifically, we assume that there is *only* one teammate, and that its behavior is fixed and known.

Admittedly, allowing for the teammate's behavior to be fixed and known may seem, at first blush, to remove an essential component of the teamwork being "ad hoc." However, consider a disaster rescue scenario in which robots developed by many different people in different parts of the world converge to work together to locate and extract victims from places that are yet too dangerous for human rescue teams to enter. The behavior and capabilities of each *type* of robot may be known a priori, even if the particular combination of robots to be contributed is not. In this case, the ad hoc team agent must determine, on the fly, how to act given the current team composition. The robots certainly form a *team*: they are fully cooperative with no notion of individual self-interest separate from the team's interest. They all aim to act so as to maximize the likelihood of finding survivors, even if it means risking their own safety. More generally, any legacy agent that has been developed in the past but is no longer easily reprogrammable could become a teammate with fixed and known behavior to a newer, more intelligent agent that is capable of reasoning about ad hoc teamwork.

Throughout the article, we will consider *Agent A* to be the agent that is within our control, known as the *ad hoc agent*; whereas *Agent B*, which reacts in a fixed way, is given by the environment.

As a second example of ad hoc teamwork with fixed and known Agent B, consider the problem of robotic exploration.[1] Assume that a robot was deployed on Mars a decade ago for the sole purpose of exploring and retrieving essential information about the soil. When the robot was deployed, its designers did not know when, if, or to what extent the robot would be able to interact with other robots as a team. However, since they envisioned the possibility that other robots would be deployed at some point, its designers equipped it with basic teamwork capabilities, namely: examining the behavior of other possible robots, and making the best decision (in this case positioning for explorations) based on their observed behavior. For example, it is aware that the team utility will be greater if the two robots explore different areas. A decade later, substantially more information about Mars

[1] This will serve as an example of leading throughout the paper.

is available, and another robot is indeed sent to Mars holding that information. The mission of this new robot is not only to explore the more fruitful areas on Mars, but also to influence the exploration pattern of the initial robot such that it will travel to these areas as well. Since the older robot (Agent B) cannot communicate directly with the new robot (Agent A), the only influence can be through the actions of the new robot. If Agents A and B make decisions simultaneously, then the setting can be modeled as a simultaneous repeated matrix game, as shown in Section 2. In this case, A should choose a set of actions *leading* Agent B to the new area for exploration yielding optimal utility for the team.

On the other hand, consider a case in which Agents A and B do not act simultaneously, but they can observe their teammate's actions and change their plan for the next day accordingly. Specifically in this example, A and B need to recharge their battery in a docking station, allowing each one of them to act sequentially (one robot active while the other recharges). Additionally, B cannot be maneuvered into exploring areas that it did not know of at the time of deployment, but chooses to explore each day one of the areas that (based on its previous observation) will most likely gain highest utility. Agent A, being recently designed, also receives the chances of getting high utility from an area of exploration based on new equipment it carries with it. Agent A can make the obvious choice of exploring only the newly discovered area, but it can also use its equipment to reveal information for Agent B. In this case, A should choose each day an action that *teaches* Agent B the utilities of its available actions. As shown in Section 3, the interaction between the agents is captured by a novel cooperative k-armed bandit formalism.

The examples described above serve to emphasize the sense in which the ad hoc teamwork problem can arise even when the teammates' behaviors are fixed and known, specifically by elaborating upon the idea of interacting with legacy agents. The importance of interaction with such sub-optimal agents (that, for example, do not use learning algorithms or other intelligent means for determining optimality of their behavior) is the essence of ad hoc teamwork: not all teammates can be assumed to be equally capable. In the following sections we concentrate on technical contributions of each of the two problems: teaching and leading in ad hoc teamwork, in this simplified, known, environment.

2. Leading a Teammate: Repeated Scenarios with Simultaneous Actions

In this section, we consider the case of an ad hoc team player, *Agent A* that is interacting with a teammate, *Agent B*, with whom it cannot communicate directly, but that is capable of adapting to its teammate's behavior. Specifically, *Agent B* observes its teammate as part of the environment, and adapts its actions according to the *best response* to some fixed history window of its observation of the environment (specifically, *Agent A*'s past moves). Therefore, *Agent A*'s goal is to find the sequence of actions that will lead he team to the highest (expected) payoff in a fully cooperative setting. In the Mars rover example described in Section 1.1, we would like to find the set of actions performed by the ad hoc robot that will lead the team to explore the most beneficial areas on Mars. We discuss in this section several teammate models for Agent B: a basic case, in which it decides its actions based on the last state of the environment it observed (specifically, agent A's last action), the case in which it can store more information and choose its action based on a larger memory size, and the case in which its actions could be random.

We begin by abstracting this setting to a game-theoretic formalism in which the agents interact in a fully cooperative iterative normal form game.

2.1. Formalism

We represent the multiagent interaction of interest as a fully cooperative iterative normal-form game between two agents, *Agent A* and *Agent B*. Let the x actions available to *Agent A* be $a_0, a_1, \ldots, a_{x-1}$ and the y actions available to its teammate, *Agent B*, be $b_0, b_1, \ldots, b_{y-1}$. The immediate payoff (a real number) when A and B select actions a_i and b_j, $m_{i,j}$ is stored in row i and column j of the payoff matrix M: $M[i,j] = m_{i,j}$. In addition we define the value of the highest payoff in the matrix, which could be realized by multiple entries, to be m^*. Without loss of generality, throughout this section, we assume that $m_{x-1,y-1} = m^*$.

$M1$	b_0	b_1	b_2
a_0	25	1	0
a_1	10	30	10
a_2	0	33	40

For example, consider the payoff matrix $M1$ for a scenario in which agents A and B each have three possible actions. If both agents select action 0 (i.e., their *joint action* is (a_0, b_0)), then the joint team payoff is $m_{0,0} = 25$. Similarly if their joint action is (a_2, b_0) their joint payoff is 0. In this case, there is a unique joint action that leads to m^*: $m_{2,2} = m^* = 40$.

Assume that b_0 is *Agent B*'s default action or that, for whatever reason, the agents have been playing (a_0, b_0) in the past. This could be, for example, because *Agent B* is not fully aware of *Agent A*'s payoffs so that it cannot unilaterally identify the best joint action, or because B does not fully trust that A will play its part of the best joint action. In the Mars rover example, this could be the initial state in which the new rover found the existing rover, before it realized that the new rover is part of its environment. The question we examine is *what sequence of actions should* Agent A *take so as to maximize the team's undiscounted long-term payoff over iterative interactions using the identical payoff matrix?* The answer to this question depends on *Agent B*'s strategy. For example, if *Agent B* is non-adaptive and always selects b_0, then the best *Agent A* can do is always select a_0.

However, if *Agent B* is adaptive, *Agent A* can lead it towards the optimal joint action by taking a sequence of actions the responses to which will cause *Agent B* to abandon b_0 and choose other actions. In order to do so, it may need to accept short-term losses with respect to the current payoffs (e.g., immediate payoffs of less than 25); however in the long run these losses will be offset by the repeated advantageous payoff of (a_2, b_2).[2]

In this article, we consider a particular class of strategies that *Agent B* could be using. Though they may not be the most sophisticated imaginable strategies, they are reasonable and often studied in the literature. The fact that they are possibly suboptimal represents the philosophy that *Agent A* must be able to adapt to its teammates as they are, not as they should be. That is, we assume that we have control only over *Agent A*, not over *Agent B*.

In particular, we specify *Agent B* as being a bounded-memory best response agent with an ϵ-greedy action strategy. That is, the agent's behavior is determined by two parameters: a memory size *mem*; and a random action rate ϵ. The agent considers the most recent *mem* actions taken by its teammate (*Agent A*), and assumes that they have been generated by the maximum

[2]In principle, it is possible that the game will not continue long enough to offset these losses. We assume that the game will be repeated a large enough number of times that it will not terminate before the agents reach the best joint action in the way that we specify. In a setting where this is not the case, one would need to include the number of iterations left as a part of the state.

likelihood policy that assigns fixed probabilities to each action. For example, if $mem = 4$ and *Agent A*'s last four actions were a_1, a_0, a_1, a_1, then *Agent B* assumes that *Agent A*'s next action will be a_0 with probability 0.25 and a_1 with probability 0.75. It then selects the action that is the best response to this assumed policy with probability $1 - \epsilon$; with probability ϵ it chooses a random action. For example, for payoff matrix $M1$ in this situation, it would select b_1 with probability $1 - \epsilon$. We denote this *best response* action as $BR(a_1, a_0, a_1, a_1) = b_1$. Note that when $\epsilon = 1$, the agent acts completely randomly.

To illustrate, we begin by considering the case of $mem = 1$ and $\epsilon = 0$. For the remainder of this section, we consider the same case, in which *Agent B* always selects the action that is the best response to *Agent A*'s previous action: b_0, b_1, or b_2 depending on whether A's last action was a_0, a_1, or a_2 respectively.

Now consider *Agent A*'s possible action sequences starting from the joint action (a_0, b_0) with payoff $m_{0,0} = 25$. Because its last action was a_0, it knows that B will select b_0 on the next play. It could immediately jump to action a_2, leading to the joint action (a_2, b_0). This action will lead to an immediate payoff of $m_{2,0} = 0$, but then will cause *Agent B* to select b_2 next, enabling a payoff of 40 on the next turn and thereafter (assuming A continues to select a_2 as it should). The resulting sequence of joint actions would be $S_0 = [(a_0, b_0), (a_2, b_0), (a_2, b_2), (a_2, b_2), \ldots]$ leading to payoffs $[25, 0, 40, 40, \ldots]$.

Alternatively, *Agent A* could move more gradually through the matrix, first selecting a_1 for a joint payoff of 10 and leading B to select b_1 on its next turn. It could then shift to a_2 for a payoff of 33, followed by 40 thereafter. The resulting sequence of joint actions would be $S_1 = [(a_0, b_0), (a_1, b_0), (a_2, b_1), (a_2, b_2), (a_2, b_2), \ldots]$ leading to payoffs $[25, 10, 33, 40, 40, \ldots]$.

We define the *cost* $C(S)$ of a joint action sequence S to be the loss from playing S when compared to always playing the joint action (a_{x-1}, b_{y-1}), which leads to payoff m^* — in the case of $M1$, 40. Thus

$$C(S_0) = (40-25)+(40-0)+(40-40)+(40-40)+\cdots = 15+40+0+0+\cdots = 55$$

and

$$C(S_1) = (40-25)+(40-10)+(40-33)+(40-40)+\cdots = 15+30+7+0+0+\cdots = 52$$

In this case, S_1 is preferable to S_0, and is in fact the optimal (lowest cost) sequence starting from (a_0, b_0).

We define the *length* $L(S)$ of a joint action sequence S to be the number of joint actions prior to the first instance of the infinite sequence of joint actions that yield m^*.[3] Thus $L(S_0) = 2$ and $L(S_1) = 3$. Note that S_1 has lower cost even though it is longer. Note also that sequences that begin with a joint action (a_i, b_j) such that $m_{i,j} = m^*$ have both length 0 and cost 0.

For a given payoff matrix, we define $S_n^*(a_i, b_j)$ to be the lowest cost sequence of length n or less starting from joint action (a_i, b_j). $S^*(a_i, b_j)$ is the lowest cost such sequence of any length. Thus, for matrix $M1$, $S_2^*(a_0, b_0) = S_0$ and $S_3^*(a_0, b_0) = S^*(a_0, b_0) = S_1$.

For the special case that no sequence of a given length exists (e.g., if $n = 0$ or $n = 1$), we define $S^*(a_i, b_j) = \omega$ and $C(\omega) = \infty$. Thus, for $M1$, $C(S_0^*(a_0, b_0)) = C(S_1^*(a_0, b_0)) = \infty$, but $C(S_1^*(a_2, b_1)) = 7$ and $C(S_0^*(a_2, b_2)) = 0$.

Finally, for a given payoff matrix M, we are interested in the length of the longest optimal sequence over all the possible starting points. We define this value as $\overline{L}(M) = \max_{i,j} L(S^*(a_i, b_j))$. For example, in matrix $M1$, $L(S^*(a_0, b_0)) = L(S_1) = 3$, and there is no optimal sequence longer than 3 starting from any other cell of the matrix (as we will prove below). Thus $\overline{L}(M1) = 3$.

2.2. Finding Optimal Sequences and Analysis

In this section, we develop algorithms for finding $S^*(a_i, b_j)$ given a payoff matrix M, and we examine the question of how long these S^*'s can be. We divide the analysis based on *Agent B*'s strategy. First, in Section 2.2.1 we assume that *Agent B* has $mem = 1$ and $\epsilon = 0$ as in Section 2.1. Next in Section 2.2.2 we consider the more difficult case of $mem > 1$. Then, in Section 2.2.3 we allow *Agent B*'s actions to be random by considering $\epsilon > 0$.

2.2.1. Deterministic Teammate with 1-Step Memory

We begin by presenting an efficient algorithm for finding all of the S^*'s for a matrix M when interacting with a deterministic teammate ($\epsilon = 0$) that always selects the best response to our most recent action ($mem = 1$). Detailed in pseudocode as Algorithm 1, it uses dynamic programming, using the S_{n-1}^*'s to compute the S_n^*'s.

[3] The length of the sequence is defined for the purpose of the complexity analysis in the following sections.

The algorithm takes as input an $x \times y$ dimensional payoff matrix M and begins by initializing the optimal sequence of length 0 for every cell in the matrix according to the definition (lines 1–5). It then enters the main loop (7–21) that successively finds the best sequences of increasingly longer lengths (as indicated by the variable *len*).

A key insight that aids efficiency is that for a given a_i, the optimal sequences for b_1–b_y are the same as the optimal sequence starting from (a_i, b_0), other than the first joint action. The reason is that a_i determines *Agent B*'s next action independently from *Agent B*'s current action: in all cases, its next action will be $b_{BR(a_i)}$. Thus, *Agent A*'s task is to select its action, a_{act}, that leads to the best possible joint action of the form $(a_{act}, b_{BR(a_i)})$.

Algorithm 1 Find S^*'s (M)

1: **for** $i = 0$ to $x - 1$ **do**
2:     **for** $j = 0$ to $y - 1$ **do**
3:         $S_0^*(a_i, b_i) = \begin{cases} [(a_i, b_i), (a_i, b_i), \ldots] & \text{if } m_{i,j} = m^* \\ \omega & \text{if } m_{i,j} < m^* \end{cases}$
4:     **end for**
5: **end for**
6: $len = 0$
7: **repeat**
8:     $len = len + 1$
9:     **for** $i = 0$ to $x - 1$ **do**
10:         $S_{len}^*(a_i, b_0) = S_{len-1}^*(a_i, b_0)$
11:         **for** $act = 0$ to $x - 1$ **do**
12:             $S' = S_{len-1}^*(a_{act}, b_{BR(a_i)})$
13:             **if** $m^* - m_{i,0} + C(S') < C(S_{len}^*(a_i, b_0))$ **then**
14:                 $S_{len}^*(a_i, b_0) = \text{PREPEND}((a_i, b_0), S')$
15:             **end if**
16:         **end for**
17:         **for** $j = 1$ to $y - 1$ **do**
18:             $S_{len}^*(a_i, b_j) = \text{REPLACEHEAD}(S_{len}^*(a_i, b_0), (a_i, b_j))$
19:         **end for**
20:     **end for**
21: **until** $len = \text{UPPERBOUND}(\overline{L}(M))$

This very computation is carried out in lines 10–16, specifically for *Agent B*'s action b_0. First, it is possible that the optimal sequence of length *len*, $S_{len}^*(a_i, b_0)$ is the same as that of length *len*-1. Thus it is initialized as such (line 10). Then for each possible next action on the part of *Agent A*, denoted

a_{act}, the cost of the resulting sequence is simply the cost of the current joint action (a_i, b_0), which is $m^* - m_{i,0}$, plus the cost of the best possible sequence of length $len - 1$ that starts from $(a_{act}, b_{BR(a_i)})$. If that cost is less than the cost of the best sequence of length len found so far, then the running best sequence is updated accordingly by prepending joint action (a_i, b_0) to the sequence $S^*_{len-1}(a_{act}, b_{BR(a_i)})$ (lines 14–16).

The resulting optimal sequence is then used to determine the optimal sequence starting from all other values of (a_i, b_j) for $1 \leq j < y$ by simply replacing the first joint action in the sequence $S^*_{len}(a_i, b_0)$ with the joint action (a_i, b_j) (lines 17–19). At the end of this loop, the optimal sequence of length len starting from any joint action (a_i, b_j) ($S^*_{len}(a_i, b_j)$) is known and stored.

The computational complexity of the main loop of Algorithm 1 (lines 7–21) is quadratic in x and linear in y. Assuming x and y are of similar dimension (Agents A and B have roughly the same number of possible actions), we can call the dimensionality of M to be $d = \max(x, y)$. In that case, the main loop has complexity $O(d^2)$. Note that sequence costs $C(S)$ can be calculated incrementally in constant time as the sequences are constructed.

The only thing left to determine is how many times this main loop needs to be run. In particular, for what value of len is it no longer possible to find a better sequence than the best of length $len - 1$. We denote this value UPPERBOUND($\overline{L}(M)$). The following two theorems prove that this value is exactly $\min(x, y)$. Thus the overall computational complexity of algorithm 1 is $O(d^3)$.

First, in Theorem 2.1, we prove that there is no need to consider sequences of length greater than $\min(x, y)$: UPPERBOUND($\overline{L}(M)$) $\leq \min(x, y)$. Then, in Theorem 2.1, we show that it *is* necessary to to consider sequences up to length $\min(x, y)$:
UPPERBOUND($\overline{L}(M)$) $\geq \min(x, y)$.

Theorem 2.1. *When interacting with a teammate with* mem $= 1$ *and* $\epsilon = 0$ *based on an* $x \times y$ *dimensional payoff matrix* M, $\overline{L}(M) \leq \min(x, y)$

Proof. We argue that $\forall M, \overline{L}(M) \leq \min(x, y)$ by first showing that $\overline{L}(M) \leq x$ and then showing that $\overline{L}(M) \leq y$. Intuitively, both cases hold because an optimal sequence can visit every row and column in the matrix at most once. If there were multiple visits to the same row or column, any steps in between could be excised from the sequence to get a lower-cost sequence. The formal arguments for the two cases are quite similar, though with a couple of subtle differences.

Case 1: $\overline{L}(M) \leq x$**.** This is equivalent to proving $\forall n \geq x$, and $\forall i, j, S^*_{n+1}(a_i, b_j) = S^*_n(a_i, b_j)$. Suppose not. Then $\exists k$ and a corresponding sequence S' such that $S' = S^*_{n+1}(a_i, b_j) = \text{PREPEND}((a_i, b_j), S^*_n(a_k, b_{BR}(i)))$ with $C(S') < C(S^*_n(a_i, b_j))$. Since $S^*_n(a_i, b_j)$ is the optimal sequence of length n or less, $L(S') = n + 1$. $n + 1 > x$, so by the pigeonhole principle, $\exists q$ such that *Agent A* selects a_q more than once in S' prior to the first instance of the terminal joint action with value m^*. Assume that (a_q, b_r) appears earlier in the sequence than $(a_q, b_{r'})$. In both cases, *Agent B*'s next action in the sequence must be $BR(a_q)$. Thus after joint action (a_q, b_r), *Agent A* could have continued as it actually did after $(a_q, b_{r'})$. This revised sequence would have cost less than S', violating the assumption that $S' = S^*_{n+1}(a_i, b_j)$. Therefore $\overline{L}(M) \leq x$.

Case 2: $\overline{L}(M) \leq y$**.** Similarly, this case is equivalent to proving that $\forall n \geq y$, and $\forall i, j, S^*_{n+1}(a_i, b_j) = S^*_n(a_i, b_j)$. Suppose not. Then $\exists k$ and a corresponding sequence S' such that $S' = S^*_{n+1}(a_i, b_j) = \text{PREPEND}((a_i, b_j), S^*_n(a_k, b_{BR}(i)))$ with $C(S') < C(S^*_n(a_i, b_j))$. Since $S^*_n(a_i, b_j)$ is the optimal sequence of length n or less, $L(S') = n + 1$. $n + 1 > y$, so by the pigeonhole principle, $\exists r$ such that *Agent B* selects b_r more than once in S' after the first entry (a_i, b_j) and up to and including the first instance of the terminal joint action with value m^*.[4] Assume that (a_q, b_r) appears earlier in the sequence than $(a_{q'}, b_r)$. Then at the point when *Agent A* selected a_q leading to (a_q, b_r), it could have instead selected $a_{q'}$, and then finished the sequence as from $(a_{q'}, b_r)$ in S'. Again, this revised sequence would have cost less than S', violating the assumption that $S' = S^*_{n+1}(a_i, b_j)$. Therefore $\overline{L}(M) \leq y$.
Therefore $\forall M, \overline{L}(M) \leq \min(x, y)$. ☐

Theorem 2.2. *$\forall x, y, \exists$ $x \times y$ dimensional matrix M such that, when interacting with a teammate with* $\text{mem} = 1$ *and* $\epsilon = 0$*,* $\overline{L}(M) = \min(x, y)$*.*

Proof. To prove existence, we construct such a matrix.

Case 1: $x = y$**.** Consider the matrix $M2$ where $\delta = 10/x$. All cells on the diagonal are $100-\delta$ except for the bottom right corner, $m_{x-1,y-1} = m^* = 100$. All cells below this diagonal are $100 - 2\delta$, and all other cells are 0. We show that for $M2$, $L(S^*(a_0, b_0)) = x$. Specifically,

$$S^*(a_0, b_0) = [(a_0, b_0), (a_1, b_0), (a_2, b_1), \ldots, (a_{x-2}, b_{y-3}), (a_{x-1}, b_{y-2}), (a_{x-1}, b_{y-1})].$$

[4] This portion of the sequence still includes $n + 1$ elements, since we are ignoring the first element (a_i, b_j), but then including the first instance of the terminal joint action.

$M2$	b_0	b_1	b_2	$\cdots$	b_{y-3}	b_{y-2}	b_{y-1}
a_0	$100-\delta$	0	0	$\cdots$	0	0	0
a_1	$100-2\delta$	$100-\delta$	0		$\vdots$	0	0
a_2	0	$100-2\delta$	$100-\delta$			$\vdots$	0
$\vdots$	$\vdots$		$\ddots$	$\ddots$			$\vdots$
a_{x-3}	0	$\vdots$		$\ddots$	$100-\delta$	0	0
a_{x-2}	0	0	$\vdots$		$100-2\delta$	$100-\delta$	0
a_{x-1}	0	0	0	$\cdots$	0	$100-2\delta$	100

To see that this sequence is optimal, note that its cost is $\delta+(x-1)*2\delta < 2x\delta = 20$. Note further, that $\forall i, BR(a_i) = b_i$. Now working backwards, in order to reach the optimal joint action (a_{x-1}, b_{y-1}), *Agent A* must have selected action a_{x-1} in the iteration prior to the first appearance of (a_{x-1}, b_{y-1}) in the sequence. When that happened, if *Agent B* had selected anything other than b_{y-2} (b_{y-1} is not an option since we are considering the iteration prior to the *first* appearance of b_{y-1} in the sequence), then there would have been a payoff of 0, leading to a sequence cost of ≥ 100. Thus joint action (a_{x-1}, b_{y-2}) must appear in the optimal sequence. Similarly, considering the first appearance of this joint action, for *Agent B* to have selected b_{y-2}, *Agent A* must have selected a_{x-2} on the prior iteration. Again, any joint action other than (a_{x-2}, b_{y-3}) (here b_{y-2} is not an option for the same reason as above) leads to a payoff of 0 and a sequence cost of ≥ 100. Continuing this line of reasoning, we can see that all the cells under the diagonal must appear in the optimal sequence starting from joint action (a_0, b_0). Furthermore, adding any additional joint actions (including those on the diagonal) only raise the cost. Therefore the sequence presented above, of length x, is indeed $S^*(a_0, b_0)$. It is easy to see that no optimal sequence from any other cell is longer.[5] Thus $\forall x, \exists x \times x$ dimension matrix M such that $\overline{L}(M) = x = \min(x, y)$.

Case 2: $x < y$**.** If $x < y$ we can construct a matrix $M2'$ that includes the $x \times x$ dimensional version of $M2$ as a submatrix and contains an additional $y - x$ columns of all 0's. By the same argument as above, $S^*(a_0, b_0)$ is the same sequence as above, which is of length x: $\overline{L}(M2') = x = \min(x, y)$.

[5] To be precise, $\forall i, j, L(S^*(a_i, b_j)) = x - i$ with one exception: $L(S^*(a_{x-1}, b_{y-1})) = 0$.

Case 3: $x > y$**.** In this case, we can construct a matrix $M2'$ based on the $y \times y$ dimensional version of $M2$ that adds $x - y$ rows of all 0's. Again, $S^*(a_0, b_0)$ is the same as above and $\overline{L}(M2') = y = \min(x, y)$.
Therefore, $\forall x, y, \exists$ an $x \times y$ dimensional matrix M such that $\overline{L}(M) = \min(x, y)$. $\square$

Theorems 2.1 and 2.2 establish that the value of the call to the function UPPERBOUND in line 21 of Algorithm 1 is $\min(x, y)$.

Note that in our analysis of this case in which *Agent B* has $mem = 1$ and $\epsilon = 0$, all of the arguments hold even if there are multiple cells in the payoff matrix M with value m^*. Furthermore, Algorithm 1 computes the optimal sequence of joint actions from *all* starting points, not just a particular starting point, all in polynomial time in the dimensionality of the matrix.

2.2.2. Longer Teammate Memory

In this section we extend our analysis from the previous section to consider interacting with teammates with $mem > 1$. This case presents considerably more difficulty than the previous one in two ways. First, though the algorithm can be naturally extended, it is no longer polynomial, but rather exponential in *mem*. Second, it is no longer straightforward to compute UPPERBOUND($\overline{L}(M)$), the maximum value of $L(S^*(a_i, b_j))$. We identify a lower bound on this maximum value, but can only conjecture that it is a tight bound.

Since the algorithm and analysis is so similar to that in Section 2.2.1, rather than presenting them fully formally, we discuss how they differ from the previous case.

To begin with, we need an added bit of notation for indicating sequences. Because *Agent B*'s actions are now no longer determined by just *Agent A*'s previous action, but rather by *Agent A*'s history of previous *mem* actions, we keep track of these actions in the sequence, indicating a step as $(a_i, b_j)[h_0; h_1; \ldots; h_{mem-1}]$ where $h_0 = a_i$ is *Agent A*'s most recent action, h_1 is its prior action, etc. Then *Agent B*'s next action in the sequence must be $b_r = BR(h_0, h_1, \ldots, h_{mem-1})$ and if *Agent A*'s next action is a_q, then the next element in the sequence is $(a_q, b_r)[a_q; a_i; h_1; \ldots; h_{mem-2}]$.

For example, returning to matrix $M1$ from Section 2.1, consider the case in which *Agent B* has $mem = 3$ (and still $\epsilon = 0$ throughout this section). A valid sequence starting from $(a_0, b_0)[a_0; a_0; a_0]$ is

$$S_2 = [(a_0, b_0)[a_0; a_0; a_0], (a_2, b_0)[a_2; a_0; a_0], (a_2, b_0)[a_2; a_2; a_0], (a_2, b_2)[a_2; a_2; a_2]]$$

Note that because $BR(a_2, a_0, a_0) = b_0$, *Agent A* needs to select a_2 twice before *Agent B* will shift to b_2. $C(S_2) = 15 + 40 + 40 = 95$. As in Section 2.1, there is another valid sequence S_3 in which *Agent A* leads *Agent B* through joint actions (a_1, b_0) and (a_2, b_1) on the way to (a_2, b_2). But now, *Agent A* must select a_1 twice before B will switch to b_1 and then a_2 three times before B will switch to b_2. Thus $C(S_3) = 25 + 2 * 30 + 3 * 7 = 106$. Hence, unlike in Section 2.1, when *Agent B* has $mem = 3$, *Agent A* is best off jumping straight to a_2.

The first necessary alteration to Algorithm 1 in this case is that it is no longer sufficient to simply calculate S^*_{len} for every joint action (a_i, b_j) on each loop of the algorithm. Rather, we must now calculate such values for each joint action-history $(a_i, b_j)[h_0; \ldots; h_{mem\text{-}1}]$. Since h_0 is constrained to be the same as a_i, there are x^{mem-1} such histories for each joint action, leading to a total of $x^{mem}y$ optimal sequences computed on each main loop of the algorithm. To accommodate this alteration, we simply need to nest additional **for** loops after lines 2 and 10 of Algorithm 1 that iterate over the (exponential number of) possible histories.

The second necessary alteration to Algorithm 1 in this case is that it is no longer sufficient to simply arrive at a joint action (a_i, b_j) such that $m_i, j = m^*$. Rather, the agents must arrive at such an action with a history of *Agent A*'s actions such that if it keeps playing a_i, *Agent B* will keep selecting b_j. We define such a joint action-history to be *stable*.

To see why the concept of stability is necessary, consider matrix $M3$. A valid sequence starting from $(a_2, b_2)[a_2; a_1; a_0]$ proceeds to $(a_2, b_2)[a_2; a_2; a_1]$ if *Agent A* selects a_2. However from there, *Agent B*'s best response is b_0, not b_2. Thus the agents do not remain stably at joint action (a_2, b_2).

$M3$	b_0	b_1	b_2
a_0	0	30	50
a_1	41	20	0
a_2	99	20	100

To accommodate this situation, the only change to Algorithm 1 that is needed is that in line 3, only stable joint-action histories such that $m_{i,j} = m^*$ should be initialized to the sequence of repeated terminal joint actions. Unstable ones should be initialized to ω (along with all instances such that $m_{i,j} < m^*$, no matter what the history). We can check stability by computing the best response to all histories that result from repeating action a_i until the entire history window is full of action a_i. If any of these best responses is not b_j, then the joint action-history is not stable.

Third, the main loop of Algorithm 1 needs to be altered to accommodate the inclusion of histories. In particular, in line 12, care needs to be taken to compute S' correctly, with *Agent B*'s action being based on the best response to the current history, and the history being the result of taking action a_i from the current history. Furthermore the PREPEND and REPLACEHEAD operators must manipulate the histories (and incremental cost computations) in the appropriate, obvious ways.

Finally, and most significantly, the value of UPPERBOUND in line 21 of Algorithm 1 must be altered. Unfortunately, we only can prove a lower bound of this value and a loose upper bound ($\min(x, y) * x^{mem-1}$). We conjecture, but have not proven, that the lower bound is tight as it is in Section 2.2.1.

Theorem 2.3. *$\forall x, y, \exists\ x \times y$ dimensional matrix M such that, when interacting with a teammate with* $\text{mem} > 1$ *and* $\epsilon = 0$, $\overline{L}(M) = (\min(x, y) - 1) * mem + 1$.

Proof. (*sketch*) This theorem, which is the analog of Theorem 2.2, can be proven using a similar construction. In particular, redefining δ as $\delta = 10/((x-1) * mem + 1)$, the same matrix $M2$ serves as our existence proof. Consider the optimal sequence starting from (a_0, b_0) with history full of a_0's. In that case, *Agent A* needs to select action a_1 *mem* times before *Agent B* will switch to b_1. Similarly, it then needs to select a_2 *mem* times before B will switch to b_2, and so on until A has selected each of the actions a_1–a_{x-1} *mem* times. The additional one is for the initial action (a_0, b_0) which appears only once in the sequence. As before, any joint actions with payoff 0 will lead to a higher sequence cost than this entire sequence, and any additional joint actions also increase the cost.

Also as before, the cases of $x \neq y$ are covered by simply adding extra rows or columns of 0's to $M2$ as needed. □

$M5$	b_0	b_1	b_2	b_3	b_4
a_0	98	0	96	97.2	0
a_1	96	98	0	0	0
a_2	0	96	98	97.2	0
a_3	0	0	0	96	100

In [2], we conjectured that the lower bound from Theorem 2.3 was tight. That is, we conjectured that it was always the case that $\overline{L}(M) \leq (\min(x, y) - 1) * mem + 1$. The intuition was that neither *Agent A* nor *Agent B* would ever select any one action more than *mem* times without foregoing some repetitions of its other actions. However, we now know that there are counterexamples to that conjecture. For example, consider the 4×5 matrix, $M5$.[6] If *Agent B*'s *mem* $= 2$ (and its $\epsilon = 0$), the optimal sequence from (a_0, b_0) starting with history $[a_0; a_0]$ ends at (a_3, b_4) and has length 8: $L(S^*(a_0, b_0)[0; 0; 0]) = 8 > (\min(x, y) - 1) * mem + 1 = 7$. Specifically, in S^* *Agent A* selects a_1 twice, then a_2 twice, but then returns to a_0 before selecting a_3 thereafter. Due to this example, and others like it, we revise our previous conjecture as follows.

Conjecture 2.1. *When interacting with a teammate with* mem > 1 *and* $\epsilon = 0$ *based on an* $x \times y$ *dimensional payoff matrix* M, $\overline{L}(M) \leq (y - 1) * \text{mem} + 1$.

Proving or disproving this conjecture is left as an important direction for future work. It may also be possible to find a tighter bound, particularly for matrices such that $y > x$. An additional important direction for future work is developing heuristics for more efficiently finding the S^*'s when *mem* > 1. Unfortunately, the problem is NP hard — see Appendix A for a proof. The exponential runtime in *mem* of the algorithm for finding the S^*'s is of practical significance. Our algorithm finds all the best sequences for a 60×60 matrix in less than 30 seconds of user time on a 1GHz laptop (calculated by the Unix `time` command) when *mem* $= 1$, but it can only handle an 18×18 matrix in that time when *mem* $= 2$, a 9×9 matrix when *mem* $= 3$, 6×6 when *mem* $= 4$, and 4×4 when *mem* $= 5$. For larger matrices than those listed, java ran out of heap space with the default settings, often after running for more than 10 minutes.

2.2.3. Teammate Randomness

Until this point, we have assumed that *Agent B* acts deterministically: *Agent A* could predict *Agent B*'s next action with certainty based on its own previous actions. In this section we relax that assumption by allowing B's ϵ to be greater than 0.

Once again, Algorithm 1 needs to be changed minimally to accommodate this case, so we just describe the changes. In fact, here, the only change necessary is that costs of joint actions be computed as expected values in comparison to the expected value of the optimal joint action.

The expected value of a joint action $EV(a_i, b_j) = (1-\epsilon)m_{i,j} + \frac{\epsilon}{y}(\sum_{k=0}^{y-1} m_{i,k})$. m^* is then defined to be the maximum expected value of a joint action in M. The cost of a sequence $C(S)$ is then the sum of the differences between

[6] Thanks to Leonid Trainer for this example.

m^* and the expected values of the joint actions in the sequence. After these changes in notation, which simply generalize our previous notation (all prior definitions hold for the case when $\epsilon = 0$), the only change necessary to Algorithm 1 is in line 13: the term $m_{i,0}$ must be replaced by $EV(a_i, b_0)$. The notion of stable joint action-histories remains unchanged from Section 2.2.2.

Note that as ϵ changes, both the optimal sequence of joint actions and the "target" joint actions (the ones that lead to expected value of m^*) can change. For example, consider the 4×4 matrix, $M4$. If *Agent B*'s $mem = 3$, then if its $\epsilon = 0$, the optimal sequence from (a_0, b_0) starting with history $[a_0; a_0; a_0]$ ends at (a_3, b_3) and has length 10: $L(S^*(a_0, b_0)[0; 0; 0]) = 10$. When $\epsilon = 0.1$, and $\epsilon = 0.3$ the optimal lengths are 8 and 3 respectively, still ending at (a_3, b_3). When $\epsilon = 0.4$, the optimal sequence is of length 3, but now ends at (a_2, b_2). All of these sequences have different costs.

$M4$	b_0	b_1	b_2	b_3
a_0	25	0	0	0
a_1	88	90	99	80
a_2	70	98	99	80
a_3	70	70	98	100

The intuitive reason for these changes is that as ϵ increases, it is no longer sufficient to reach a good cell in the matrix, but rather *Agent A* must aim for a good row: any value in the row is possible to be the payoff of the joint action. For this reason, with high ϵ, the row corresponding to a_2 is preferable to that corresponding to a_3 (the sum of the values is higher).

The analysis of the algorithmic runtime remains mostly unchanged. For efficiency, the expected values of joint actions can be cached so that they only need to be computed once. However ϵ does have some effects on the value of UPPERBOUND in line 21 of the algorithm. For $\epsilon < 1$, Theorems 2.1–2.3 all hold, though δ in the example matrix $M2$ needs to be generalized to $\delta = \frac{20(1-\epsilon)}{((x+1)*mem)(2-2\epsilon+\frac{\epsilon}{y})}$. However when $\epsilon = 1$, UPPERBOUND$(\overline{L}(M)) = 1$: *Agent A* can always jump immediately to the action that leads to the row with the highest expected value, which will be attained by all joint actions in that row. It is not clear whether ϵ has any effect on Conjecture 2.1.

2.3. Empirical Results

All variations of the algorithm presented in Section 2.2 are fully implemented. In this section, we present some brief empirical results from running them in various settings that shed some light on the nature and prevalence of our problem of interest.

In particular, we consider how frequently action sequences of various lengths appear in random matrices. At first blush, it may seem that when interacting with an agent with $mem = 1$, matrices for which there $\exists(a_i, b_j)$ such that $L(S^*(a_i, b_j)) > 2$ (such as $M1$ in Section 2.1) would be relatively rare in practice.

To test this hypothesis, we generated random $x \times y$ matrices such that $m_{x-1,y-1} = 100$ and all other values $m_{i,j}$ are generated uniformly randomly from $[0, 100]$. Table 1 shows the distribution of $\overline{L}(M)$ for $x \times x$ matrices when *Agent B*'s $mem = 1$ or 3. For matrices larger than 7×7, the $mem = 3$ case takes more than a day to run on a modern laptop, so we stop at that point. Matrices such that $x \neq y$ did not show any interestingly different patterns.

mem=1	1	2	3	4	5	6	7	8	9	10
3×3	104	852	44							
4×4	12	825	158	5						
5×5	3	662	316	19	0					
6×6	0	465	489	45	1	0				
7×7	0	349	565	81	5	0	0			
8×8	0	236	596	159	8	1	0	0		
9×9	0	145	640	193	20	2	0	0	0	
10×10	0	72	636	263	29	0	0	0	0	0

mem=3	1	2	3	4	5	6	7	8	9	10	11
3×3	98	178	344	340	28	8	4	0	0	0	0
4×4	15	76	266	428	134	60	21	0	0	0	0
5×5	1	19	115	408	234	145	71	7	0	0	0
6×6	0	0	22	282	272	222	164	27	11	0	0
7×7	0	0	5	116	293	282	220	55	17	10	1

Table 1: Distribution of $\overline{L}(M)$ for 1000 randomly generated matrices of various sizes. *top*: *Agent B*'s $mem = 1$. No entries are shown for values that we know to be impossible from Theorem 2.1. *bottom*: $mem = 3$. No values greater than 11 were found.

From these results we see that even in 3×3 matrices with $mem = 1$, it is not uncommon for *Agent A* to need to reason about the cost of various sequence lengths: In 44 of 1000 cases, there is at least one joint action from which *Agent A* is best off not jumping immediately to action a_2. In 104 of the cases, all optimal sequences are of length 1, which occurs exactly when b_2 is the best response to all of A's actions: $\forall 0 \leq i < x, BR(a_i) = b_{y-1}$ (as expected, this occurrence becomes less common as the matrix size increases). In the other 852 cases, *Agent A* is best off switching to a_2 immediately, leading to longest sequences of length 2.

Though matrices such that $\overline{L}(M) > 2$ are not uncommon, it is also noticeable that matrices with optimal sequences of lengths close to the theoretical maximum do not occur naturally as the matrix size increases. A carefully selected construct such as $M2$ in Section 2.2 is required to find such sequences.

2.4. Simultaneous Action Summary

A brief summary of the results from this section on repeated scenarios with simultaneous actions is as follows, both as a table, and also with slightly more explanation of each item, as a bulleted list.

	Efficiency of finding optimal action sequence	Maximum length of optimal sequence
Deterministic Teammate, 1-Step Memory	$O(d^3)$	$d = \min(x, y)$
Deterministic Teammate, Longer Memory	NP Hard	Open problem; Between $(\min(x, y) - 1) * mem + 1$ and $\min(x, y) * x^{mem-1}$
Random Teammate	Same as deterministic	Same as deterministic

Deterministic teammate with 1-Step memory:

- Can find optimal action sequence efficiently: $O(d^3)$
- Maximum length of optimal sequence: $\min(x, y)$

Longer teammate memory:

- Cannot find optimal action sequence efficiently: $NPhard$
- Maximum length of optimal sequence: open problem — between $(\min(x, y) - 1) * mem + 1$ and $\min(x, y) * x^{mem-1}$

Random teammate:

- Same as deterministic teammate: depends on teammate memory size, with same bounds above applying.

3. Teaching a Teammate: Sequential Scenarios with Differing Abilities

Section 2 explored the scenario in which Agent B is fixed and known and the two agents repeatedly take *simultaneous* actions. This section maintains the assumption that Agent B is fixed and known, but now considers the case in which the teammates interact in a sequential turn-taking scenario, as motivated in Section 1.1. This scenario can be formalized as a finite-horizon cooperative k-armed bandit [3] in a way that, to the best of our knowledge, has never been considered before in the literature. The formalism can be applied to any multiagent decision-making setting that shares the essential characteristics of the scenario described above, and can also be generalized to *ad hoc teamwork* settings.

In this section, we characterize the conditions under which certain actions are potentially optimal in such a finite-horizon, cooperative k-armed bandit, and we present a dynamic programming algorithm that solves for the optimal action when the payoffs come from a discrete distribution. For Gaussian distributions we present some theoretical and experimental results and identify an open problem. While k-armed bandits are often used to study the exploration versus exploitation challenge, nobody has previously considered a multiagent cooperative setting in which the agents have different knowledge states and action capabilities. Thus our formalization is simultaneously a practical method for multiagent team decision-making, and a novel contribution to the literature on k-armed bandits.

3.1. Formalism

The k-armed bandit problem [3] is a much-studied problem in sequential decision making. The basic setting is as follows. At each time step, a learning agent selects one of the k arms to pull. The arm returns a payoff according to a fixed, but generally unknown, distribution. Similar to the problem of leading teammates presented in section 2.1, the agent's goal is to maximize the team utility, specifically, to *maximize the sum of the payoffs it receives over time.* The k-armed bandit is a classic setting for studying the exploration vs. exploitation problem: at any given time, the agent could greedily select the arm that has paid off the best so far, or it could select a different arm in order to gather more information about its distribution. It is also the basis for reinforcement learning theory, representing the stateless action selection problem [4].

In order to study the ad hoc team problem laid out in this section we extend the standard setting to include two distinct agents, known as the *teacher* (*agent A*) and the *learner* (*agent B*), who select arms alternately, starting with the teacher. We initially consider a bandit with just three arms such that the teacher is able to select from any of the three arms, while the learner is only able to select from among the two arms with the lower expected payoffs. We consider the fully cooperative case such that the teacher's goal is to maximize the expected sum of the payoffs received by the two agents over time (the teacher is risk neutral). Specifically, we make the following assumptions:

- The payoff distributions of all arms are fully known to the teacher, but unknown to the learner.
- The learner can only select from among the two arms with the lower expected payoffs.
- The results of all actions are fully observable (to both agents).
- The number of rounds (actions per agent) remaining is finite and known to the teacher.

We assume that the learner's behavior (Agent B) is fixed and known: it acts greedily, always selecting the arm with the highest observed sample average so far. Any arm that has never been pulled is assumed to have a sample average of ∞. Thus, the learner always prefers selecting an arm that has not been selected previously. If there is more than one such arm, it selects randomly from among them. This assumption reflects optimism in the face of uncertainty on the part of the learner (optimistic initialization).

The teacher must then decide whether to do what is best in the short term, namely pull the arm with the highest expected payoff; or whether to increase the information available to its teammate, the learner, by pulling a different arm. Note that if the teacher were acting alone, trivially its optimal action would be to always pull the arm with highest expected payoff. Referring to the Mars rover example from Section 1.1, the new rover should decide whether to explore alone areas that are more beneficial for the mission and disregard the existing robots' whereabouts, or try to influence the area the old robot is exploring, by choosing to explore a less beneficial zone. The arms here refer to the possible zones the robots can explore with their possible benefits to the team.

By these assumptions, the learner is both less capable and less knowledgeable than the teacher, and it does not understand direct communication from the teacher. It is tempting to think that we should begin by improving the learner. But in the ad hoc team setting, that is not an option. The learner "is what it is" either because it is a legacy agent, or because it has been programmed by others. Our task is to determine the teacher's best actions *given* such learner behavior.

We use the following notation for the three arms. The learner selects between Arm_1 and Arm_2, while the teacher can additionally choose Arm_*. While we consider two different forms of distributions for the payoffs, throughout the section we use the following notation:

- μ_i is the expected payoff of Arm_i ($i \in \{1, 2, *\}$).
- n_i is the number of times Arm_i has been pulled (observed) in the past.
- m_i is the cumulative payoff from all the past pulls of Arm_i.
- $\bar{x}_i = \frac{m_i}{n_i}$ is the observed sample average so far.
- r is the number of rounds left.

Throughout the section we assume that $\mu_* > \mu_1 > \mu_2$. If μ_* is not the largest, then the teacher's choice is trivially to always select the arm with the largest expected payoff. The ordering of Arm_1 and Arm_2 is without loss of generality. In this setting, the question we ask is, which arm should the teacher pull, as a function of r and all the n_i, $\bar{x}_i$, and Arm_i payoff distributions (including μ_i)?

We will consider two different forms of payoff distributions for the arms. First, in the simpler "discrete" case, each Arm_i returns either a 1 or a 0 with probability p_i. Thus $\mu_i = p_i$ and m_i is the number of times the arm has yielded a payoff of 1. In this case, we derive a polynomial memory and time algorithm for determining the teacher's optimal action in any situation. The analysis generalizes naturally to any discrete distribution. Second, in the more difficult "normal" case, each Arm_i returns a value from a Gaussian distribution with standard deviation σ_i (and mean μ_i). In this case, we can only determine the optimal action efficiently when $r = 1$, though the optimal action can be estimated numerically when $r > 1$.

We begin with theoretical results that hold for any type of distribution in Section 3.2. We then present the complete solution to the discrete case in Section 3.3 followed by our analysis of the normal case in Section 3.4.

3.2. Arbitrary Distribution Arms

In this section, we present theoretical results that apply regardless of the forms of the distributions of the payoffs from the three arms.

3.2.1. The teacher should consider pulling Arm$_1$

First, to understand that the problem specified in Section 3.1 is not trivial, we show that there are situations in which the teacher should not greedily optimize its short-term payoff by pulling Arm$_*$, but rather should increase the amount of information available to the learner by pulling Arm$_1$.

In fact, even with just one round remaining ($r = 1$), it is not difficult to construct such a case. For example, suppose that $\mu_* = 10, \mu_1 = 9, \mu_2 = 5, \bar{x}_1 = 6, \bar{x}_2 = 7, n_1 = n_2 = 1$. Suppose further that the distribution of payoffs from Arm$_1$ is such that the probability of obtaining a value greater than 8 is $\eta > \frac{1}{2}$. Thus with probability η, after an agent selects Arm$_1$, its sample average will be greater than $\bar{x}_2$.

Should the teacher select Arm$_*$, then the learner will select Arm$_2$ (because $\bar{x}_1 < \bar{x}_2$), yielding an expected total payoff during the round of $\mu_* + \mu_2 = 15$. On the other hand, should the teacher select Arm$_1$, there is a greater than 50% chance that the learner will select Arm$_1$ as well. The expected payoff is then $\mu_1 + \eta\mu_1 + (1 - \eta)\mu_2 > 9 + \frac{9}{2} + \frac{5}{2} = 16$.

Therefore there are situations in which it is better for the teacher to pull Arm$_1$ than Arm$_*$. This article is devoted to characterizing exactly what those situations are.

3.2.2. The teacher should never pull Arm$_2$

Second, we argue that the teacher should only consider pulling Arm$_*$ or Arm$_1$. On the surface, this result appears obvious: why should the teacher pull Arm$_2$ just to prevent the learner from doing the same? In fact, there is a relatively straightforward proof that applies when $\bar{x}_1 < \bar{x}_2$ (similar to our proof of Theorem 3.2 below). However the proof of the fully general result that includes the seemingly simpler case that $\bar{x}_1 > \bar{x}_2$ is surprisingly subtle. We sketch the proof below. The full proof appears in Appendix B.

Theorem 3.1. *It is never optimal for the teacher to pull Arm$_2$.*

Proof sketch. The proof uses induction on r.
Base case: $r = 1$. If the teacher starts by pulling Arm$_2$, the best expected value the team can achieve is $\mu_2 + \mu_1$. Meanwhile, if it starts with Arm$_*$, the worst the team expects is $\mu_* + \mu_2$. This expectation is higher since $\mu* > \mu_1$.

Inductive step: Assume that the teacher should never pull Arm_2 with $r-1$ rounds left. Let π^* be the optimal teacher action policy that maps the states of the arms (their μ_i, n_i, and $\bar{x}_i$) and the number of rounds left to the optimal action: the policy that leads to the highest long-term expected value. Consider the sequence, S, that begins with Arm_2 and subsequently results from the teacher following π^*. To show: there exists a teacher action policy π' starting with Arm_* (or Arm_1) that leads to a sequence T with expected value greater than that of S. That is, the initial pull of Arm_2 in S does not follow π^*.

The underlying idea is that the sequence T should start with the teacher pulling Arm_* repeatedly, and tracking the values obtained by the learner to see if it can ever discern what the sequence S would have looked like after some number of rounds (it *simulates* sequence S). This may not be possible, for example if sequence S begins with a pull of Arm_1, whereas after the initial pull of Arm_2 in T, the values are such that Arm_1 is never pulled.

If the teacher ever *does* get to the point that all of the learner's pulls of Arm_1 and Arm_2 in T can be used in simulating S, then the teacher can mimic S from that point until it runs out of rounds (we can prove that the simulation necessarily ends with fewer rounds executed in S than in T). Then nothing that would have happened after the mimicking ended (that is that *will* happen in S) could have higher expected value than all the extra pulls of Arm_* that came before the mimicking started in T.

If, on there other hand, there is never a point that all the pulls of Arm_1 and Arm_2 can be used in the simulation, then sequence T must have more pulls of Arm_* and fewer pulls of Arm_2 than sequence S (which itself requires some care to prove rigorously).

Either way, the sequence T has higher expected value than sequence S, so the initial pull of Arm_2 in S was suboptimal. □

Thus, when the teacher decides to teach the learner, it does so by pulling Arm_1. Pulling Arm_* can be thought of as exploiting, or maximizing short-term payoff. In the remainder of this section, we sometimes refer to the teacher pulling Arm_1 as "teaching," and pulling Arm_* as "not teaching."

3.2.3. Never teach when $\bar{\boldsymbol{x}}_1 > \bar{\boldsymbol{x}}_2$

Third, we show that the teacher's choice is clear whenever $\bar{x}_1 > \bar{x}_2$. That is, if the current sample average of Arm_1 is greater than that of Arm_2 such that the learner will choose Arm_1 next, then the teacher should always choose Arm_*: it should not teach.

Theorem 3.2. *When $\bar{x}_1 > \bar{x}_2$, it is always optimal for the teacher not to teach (to pull Arm$_*$).*

Proof. When $r = 1$, the theorem is clearly true: the expected reward for the round when not teaching is already the maximum possible: $\mu_* + \mu_1$. When $r > 1$ the argument is a simpler version of the proof to Theorem 3.1. Consider the sequence S that begins with Arm$_1$ and then follows the optimal policy π^* thereafter. Compare it with the sequence T that results from the teacher pulling Arm$_*$ in the first two rounds, then mimicking sequence S thereafter: following π^* as if there were one more round remaining than is actually remaining. Since the first two values in S are equivalent to the learner's first two values in T (it will begin with Arm$_1$ because $\bar{x}_1 > \bar{x}_2$), the sequences are identical other than the teacher's first two pulls of Arm$_*$ in T and the last action of each agent in S. Thus the expected value of $T - S \geq (\mu_* + \mu_*) - (\mu_* + \mu_1) > 0$. Since S is the best the teacher can do if it starts with Arm$_1$, and T is a lower bound on how well it can do otherwise, the teacher should never pull Arm$_1$ when $\bar{x}_1 > \bar{x}_2$. □

3.2.4. Do not teach when $\boldsymbol{n_1 = 0}$ *and/or* $\boldsymbol{n_2 = 0}$

When starting a new task such that the learner has no experience with any of its arms, the teacher should pull Arm$_*$: it should not teach. The proof proceeds similarly to the proof of Theorem 3.2. In fact, the proof generalizes to the statement that the teacher should never do what the student is about to do anyway.

3.3. Discrete Distribution Arms

In Section 3.2, we presented theoretical results that do not depend in any way on the form of the distributions governing the payoffs from the various arms: the teacher should never pull Arm$_2$, and it should only consider Arm$_1$ when $\bar{x}_1 < \bar{x}_2$. In this section and the next, we analyze when exactly the teacher should select Arm$_1$, which depends on the exact distributions of the payoffs. We first restrict our attention to *binary* distributions such that each Arm$_i$ returns a 1 with probability p_i, and a 0 otherwise. Referring to the Mars rover example, this case is equivalent to a "success" and "failure" in the exploration mission in the zone: was the robot able to produce valuable information today in its exploration mission, or not? Here, $\mu_i = p_i$, and m_i is the number of times the arm has yielded a payoff of 1 thus far. In this setting we can solve for the optimal teacher action using finite horizon dynamic programming. The algorithm generalizes to any discrete distribution.

3.3.1. $\bar{x}_1 < \bar{x}_2, r = 1$

To develop intuition, we begin by considering what the teacher should do when $r = 1$ (one action remaining for each agent). As shown in Section 3.2, the teacher should never teach when $\bar{x}_1 > \bar{x}_2$.

When $\bar{x}_1 < \bar{x}_2$ (i.e., $\frac{m_1}{n_1} < \frac{m_2}{n_2}$), there are two conditions that must hold for it to be worthwhile for the teacher to teach. First, it must be the case that pulling Arm_1 could change the learner's action from Arm_2 to Arm_1; and second, it must be the case that the expected cost of teaching is less than the expected benefit of teaching. Specifically, we need the following to hold:

1. $\frac{m_1+1}{n_1+1} > \frac{m_2}{n_2}$
2. $p_* - p_1 < p_1(p_1 - p_2)$

The right hand side of the second inequality is the probability that Arm_1 will yield a 1 multiplied by the difference in expected values between Arm_1 and Arm_2.

Note that we can also explicitly calculate the expected values of both not teaching (EV_{nt}) and teaching (EV_t). $\text{EV}_{nt} = p_* + p_2$ and $\text{EV}_t = p_1 + p_1^2 + (1 - p_1)p_2$.

3.3.2. *Algorithm*

Building on the intuition from Section 3.3.1, this section presents our fully-implemented polynomial memory and time dynamic programming algorithm for determining the teacher's optimal action with any number of rounds left. It takes as input initial values for m_1, n_1, m_2, n_2, and r, which we denote as M_1, N_1, M_2, N_2, and R respectively, and it outputs whether the teacher's expected value is higher if it teaches by pulling Arm_1 or if it exploits by pulling Arm_*.

The dynamic programming algorithm works backwards from smaller to bigger values of r, computing the expected value of the optimal action from any possible values of m_1, n_1, m_2, and n_2 that could be reached from the initial values.

First, consider the values that m_1, n_1, m_2, and n_2 can take on when there are r rounds left.

- Because both agents can pull Arm_1 any number of times, with r rounds left (after $R - r$ rounds have passed), n_1 can range from N_1 (if Arm_1 was never selected) to $N_1 + 2(R - r)$.

- Any number of the $n_1 - N_1$ times that Arm_1 was pulled, m_1 could have increased by 1. Thus m_1 can range from M_1 to $M_1 + (n_1 - N_1)$.

- Because only the learner pulls Arm_2, it will be pulled at most once per round. But the range of n_2 depends on the value n_1, because the learner only pulls Arm_2 when it does not pull Arm_1. Thus n_2 can range from $N_2+\max(0, R-r-(n_1-N_1))$ to $N_2+(R-r)-\max(0, n_1-N_1-(R-r))$.

- Similarly to m_1, m_2 can range from M_2 to $M_2 + (n_2 - N_2)$.

The algorithm, detailed as pseudocode in Algorithm 2, is structured as nested `for` loops using these ranges. For each reachable combination of values, the algorithm computes the teacher's optimal action (Arm_1 or Arm_*), denoted $\mathrm{Act}[\cdot]$; and the expected long-term value of taking that action, denoted $\mathrm{Val}[\cdot]$: the expected sum of payoffs for the optimal action and all future actions by both the teacher and the learner.

First, in Line 1, the expected value with zero rounds remaining is defined to be 0 since there are no more actions to be taken. Then, in the body of the nested `for` loops (Lines 7–45), the expected values of both teaching by pulling Arm_1 (EV_t) and not teaching by pulling Arm_* (EV_{nt}) with r rounds remaining are computed based on the stored values for the possible resulting states with $r-1$ rounds remaining.

The values of these possible resulting states are denoted as EV_{abcd} where a, b, c, and d denote the increments to m_1, n_1, m_2, and n_2 respectively between rounds r and $r-1$ (Lines 7–17). For example, Line 25 computes the expected value for not teaching when $n_1, n_2 > 0$ and $\frac{m_1}{n_1} > \frac{m_2}{n_2}$. In the current round, the teacher exploits (does not teach) by pulling Arm_* and the learner pulls Arm_1, leading to an expected return of $p_* + p_1$. This value is then added to the expected value of the resulting state with $r-1$ rounds remaining. Due to the learner's action, the value of n_1 is incremented by 1. With a probability of p_1, this action returns a payoff of 1, causing m_1 to be incremented as well. With a probability of $1-p_1$, m_1 is not incremented. Thus the expected value after the current round is $p_1\mathrm{EV}_{1100} + (1-p_1)\mathrm{EV}_{0100}$. Note that there are special cases for the situations in which n_1 and/or n_2 are 0 corresponding to the assumed learner behavior as specified in Section 3.1.

Once the expected values of teaching and not teaching have been computed, they are compared in Line 38, and the $\mathrm{Act}[\cdot]$ and $\mathrm{Val}[\cdot]$ entries are set according to the result. Finally, the appropriate action with R rounds remaining is returned (Line 50). Note that by storing the optimal actions along the way (Act[·]), the algorithm eliminates the need to do any additional computations in the future as the number of rounds remaining (r) decreases to 1. For all possible results of the teacher's and learner's actions, the optimal teacher action in all future rounds is already stored.

3.3.3. Algorithm analysis

In this Section we analyze the memory and runtime properties of Algorithm 2, specifically showing that it is polynomial in R in both respects.

First, notice that both the memory and the runtime complexity is determined by the number of iterations through the nested `for` loop. Each iteration through the loop requires that one expected value and one optimal action be stored; and the computation within the loop is constant with respect to r.

Thus the relevant quantity is the number of combinations of values m_1, n_1, m_2, n_2, and r can take in the body of the loop. Looking at their ranges as laid out at the beginning of Section 3.3.2, it is clear that this number is bounded above by $2R * 2R * R * R * R = 4R^5$. Therefore both the memory and runtime complexities of this algorithm for computing the optimal teacher action with R rounds remaining for any starting values of the other variables are $O(R^5)$.

Although the algorithm runs iteratively, using dynamic programming, in principle we can convert the stored data structure into closed form computations of both teaching and not teaching. This conversion is based on the probabilities of the various possible outcomes of the pulls of the arms. However the closed form equations will be dependent upon m_1, n_1, m_2, and n_2.

3.3.4. Other Discrete Distributions

The algorithm and analysis to this point in this section all deal with the *binary* case in which each arm returns either 1 or 0 on each pull: 1 for a success and 0 for a failure. However, the algorithm and analysis extend trivially to distributions in which the success and failure payoffs from each arm differ from 1 and 0 and differ across the arms. The key property is that each arm has a success payoff that is realized with probability p_i and a (lower) failure payoff that is realized otherwise. Either or both of the payoffs can even be negative, representing an action penalty. In order to adapt the algorithm, the calculations of the expected values in lines 18–37 need to be changed to reflect the revised payoffs, and the calculations of the sample average (e.g. in Line 24), need to reflect the revised payoffs by multiplying

Algorithm 2 TeachOrExploit(M_1, N_1, M_2, N_2, R)

```
Require: p_1, p_2, p_*
Define Val[m_1, n_1, m_2, n_2, 0] = 0, ∀m_1, n_1, m_2, n_2
for r = 1 to R do
    for n_1 = N_1 to N_1 + 2(R − r) do
        for m_1 = M_1 to M_1 + (n_1 − N_1) do
            for n_2 = N_2 + max(0, R − r − (n_1 − N_1)) to N_2 + (R − r) − max(0, n_1 − N_1 − (R − r)) do
                for m_2 = M_2 to M_2 + (n_2 − N_2) do
                    EV_1100 = Val[m_1 + 1, n_1 + 1, m_2, n_2, r − 1]
                    EV_0100 = Val[m_1, n_1 + 1, m_2, n_2, r − 1]
                    EV_0011 = Val[m_1, n_1, m_2 + 1, n_2 + 1, r − 1]
                    EV_0001 = Val[m_1, n_1, m_2, n_2 + 1, r − 1]
                    EV_2200 = Val[m_1 + 2, n_1 + 2, m_2, n_2, r − 1]
                    EV_1200 = Val[m_1 + 1, n_1 + 2, m_2, n_2, r − 1]
                    EV_0200 = Val[m_1, n_1 + 2, m_2, n_2, r − 1]
                    EV_1111 = Val[m_1 + 1, n_1 + 1, m_2 + 1, n_2 + 1, r − 1]
                    EV_1101 = Val[m_1 + 1, n_1 + 1, m_2, n_2 + 1, r − 1]
                    EV_0111 = Val[m_1, n_1 + 1, m_2 + 1, n_2 + 1, r − 1]
                    EV_0101 = Val[m_1, n_1 + 1, m_2, n_2 + 1, r − 1]
                    if n_1 = 0 and n_2 = 0 then
                        EV_nt = p_* + .5(p_1(1+EV_1100) + (1 − p_1)EV_0100) + .5(p_2(1+EV_0011) + (1 − p_2)EV_0001)
                    else if n_1 = 0 then
                        EV_nt = p_* + p_1(1+EV_1100) + (1 − p_1)EV_0100
                    else if n_2 = 0 then
                        EV_nt = p_* + p_2(1+EV_0011) + (1 − p_2)EV_0001
                    else if m_1/n_1 > m_2/n_2 then
                        EV_nt = p_* + p_1 + p_1 EV_1100 + (1 − p_1)EV_0100
                    else
                        EV_nt = p_* + p_2 + p_2 EV_0011 + (1 − p_2)EV_0001
                    end if
                    if n_2 = 0 then
                        EV_t = p_1 + p_2 + p_1 p_2 EV_1111 + p_1(1 − p_2)EV_1101 + (1 − p_1)p_2 EV_0111 + (1 − p_1)(1 − p_2)EV_0101
                    else if m_1/(n_1+1) > m_2/n_2 then
                        EV_t = 2p_1 + p_1 p_1 EV_2200 + 2p_1(1 − p_1)EV_1200 + (1 − p_1)(1 − p_1)EV_0200
                    else if (m_1+1)/(n_1+1) < m_2/n_2 then
                        EV_t = p_1 + p_2 + p_1 p_2 EV_1111 + p_1(1 − p_2)EV_1101 + (1 − p_1)p_2 EV_0111 + (1 − p_1)(1 − p_2)EV_0101
                    else
                        EV_t = p_1(1 + p_1(1+EV_2200) + (1 − p_1)EV_1200) + (1 − p_1)(p_2(1+EV_0111) + (1 − p2)EV_0101)
                    end if
                    if EV_nt > EV_t then
                        Act[m_1, n_1, m_2, n_2, r] = Arm_*
                        Val[m_1, n_1, m_2, n_2, r] = EV_nt
                    else
                        Act[m_1, n_1, m_2, n_2, r] = Arm_1
                        Val[m_1, n_1, m_2, n_2, r] = EV_t
                    end if
                end for
            end for
        end for
    end for
end for
Return Act[M_1, N_1, M_2, N_2, R]
```

m_1 and m_2 appropriately and computing the weighted averages with $n_1 - m_1$ and $n_2 - m_2$ respectively.

The results can also be generalized from binary distributions to any discrete distribution. In this case the algorithm includes extra nested `for` loops for each possible outcome of pulling an arm (not just two per arm). The exponent of the space and runtime complexities of the algorithm is increased accordingly, but the algorithm remains polynomial.

3.3.5. Numerical Results and Experiments

With the aid of the algorithm presented in Section 3.3.2, we tested several conjectures experimentally. In this section we consider the following questions:

1. Are there any patterns in the optimal action as a function of r when all other parameters are held constant?
2. How sensitive is the expected value computation to the relationship between $m_1, n_1, m_2, n_2, p_1, p_2$, and p_*?
3. When Algorithm 2 is run, how many of the states tend to have Arm_1 (teaching) as the optimal action?

First, consider the effect of increasing the number of rounds remaining to be played, r. Intuitively, as r increases, the more time there is to benefit from teaching. For example, consider the case in which $p_* = .5, p_1 = .4$, and $p_2 = .16$. Suppose that the learner has observed Arm_1 being pulled 3 times, one of which successfully gave a payoff of 1 ($m_1 = 1, n_1 = 3$) as well as Arm_2 being pulled 5 times, two of which succeeded ($m_2 = 2, n_2 = 5$).

In this case, with one round left the teacher should not teach: although condition 1 from Section 3.3.1 holds, condition 2 does not. In particular the probabilities are such that the cost of teaching ($.5 - .4 = .1$) is not outweighed by the expected benefit of teaching ($.4 * (.4 - .16) = .096$). However, when $r = 2$, there is enough time for the learner to take advantage of the added knowledge. In this case, the expected value of teaching, $\text{EV}_t = 1.3544$ is greater than that of not teaching, $\text{EV}_{nt} = 1.32$.

Though this result matches intuition, there are also cases such that increasing r changes the optimal action from teaching to not teaching. In fact, with $r = 3$ or 4 and all other values above unchanged, the optimal action of the teacher is again not to teach. For $r > 4$ (at least up to 16), the optimal action is to teach. However, there are even cases such that increasing r from 1 to 2 leads to a change in optimal action from teaching to not teaching. We will revisit this phenomenon in Section 3.4.3 in the context of arms with Gaussian distributions. The intuition is that with just one round remaining, there is a small enough cost to teaching that the teacher ought to try to get the learner to forgo Arm_2 even though the chances of succeeding are small;

but with two rounds remaining, the learner's initial selection of Arm_2 will almost surely be sufficient for it to "teach itself" that it should select Arm_1 on the next round. This scenario is exemplified by the following parameters: $p_* = .076075, p_1 = .076, p_2 = .075, m_1 = 3020, n_1 = 40000, m_2 = 910, n_2 = 12052$.[7] In this case, both constraints from Section 3.3.1 are satisfied, thus the optimal action when $r = 1$ is Arm_1 (teach). However when $r = 2$, $\text{EV}_t = .302228 < \text{EV}_{nt} = .303075$: the optimal teacher action is Arm_*.

Second, note that the optimal action is very sensitive to the exact values of all the parameters. For example, when $p_* = .5, p_1 = .4, p_2 = .16, r = 4, m_2 = 2$, and $n_2 = 5$, (the same parameters considered at the beginning of this section), the teacher's optimal action can differ even for identical values of $\bar{x}_1$. When $m_1 = 1$ and $n_1 = 3$, the optimal action is not to teach (Arm_*), but when $m_1 = 2$ and $n_1 = 6$, the optimal action is to teach (Arm_1) — even though $\bar{x}_1$ is $\frac{1}{3}$ in both cases. Similarly small changes in any of the other parameter values can change the teacher's optimal action.

Third, we consider how many of the states tend to have Arm_1 (teaching) as the optimal action when running Algorithm 2. For example, when $p_* = .5, p_1 = .4, p_2 = .16, m_1 = n_1 = m_2 = n_2 = 1$, solving for the optimal action with 15 rounds to go (r=15) leads to 81600 optimal actions computed (iterations through the `for` loops), 80300 of which are not to teach (Arm_*). In general, it seems that at least 90% of the optimal actions are Arm_*, even when the ultimate correct action is to teach, and usually significantly more than that. This observation perhaps suggests that in the Gaussian case below, when the optimal action cannot be solved for so easily, the default heuristic should be not to teach. We examine this hypothesis in Section 3.4.3.

3.4. Normal Distribution Arms

In Section 3.3, we focused on arms with discrete payoff distributions. However in general ad hoc team settings, action payoffs may come from continuous distributions. In this section we turn to the case in which the distributions are Gaussian. Now, in addition to the expected value μ_i, which is the mean of the distribution, arms are characterized by a standard deviation, σ_i.

There are two main reasons that this case is more complicated than the discrete case. First, rather than a discrete set of possible future states, there

[7]Note that this scenario is not particularly unlikely: $\frac{m_1}{n_1} \approx p_1, \frac{m_2}{n_2} \approx p_2$.

are infinitely many possible outcomes from each pull. Second, in contrast to the constraints laid out in Section 3.3.1 for when it is worthwhile to teach, in the Gaussian case the μ's and the $\bar{x}$'s (which correspond to the p's and the m's and n's in the binary case) interact in the same inequality, rather than constituting independent constraints.

Both of these complications are readily illustrated even with $r = 1$. We thus begin by analyzing that case in Section 3.4.1. Recall that all the results from Section 3.2 still apply in this case. For example, it is only worth considering teaching when $\bar{x}_1 < \bar{x}_2$. We then consider the case when $r = 2$ in Section 3.4.2 and present some empirical data in Section 3.4.3. In contrast to the discrete case, we do not have an algorithm for exactly computing the optimal action when $r > 1$. In principle it can be estimated numerically, though with increasing inefficiency as r increases.

3.4.1. $\boldsymbol{\bar{x}_1 < \bar{x}_2, r = 1}$

In order to analyze this case, we make use of the cumulative distribution function (CDF) of the normal distribution, denoted as $\Phi_{\mu,\sigma}(v)$. Exactly as in the binary case, with one round left, the teacher should teach when the expected cost of teaching, $\mu_* - \mu_1$, is less than the probability that teaching will successfully cause the learner to switch its choice from Arm_2 to Arm_1, $\Phi_{\mu_1,\sigma_1}(y)$, multiplied by the benefit of successful teaching, $\mu_1 - \mu_2$. Here y is the minimum return from Arm_1 that would cause the sample average of Arm_1 to surpass that of Arm_2: $\frac{m_1+y}{n_1+1} = \bar{x}_2$.

Therefore, the teacher should pull Arm_1 if and only if

$$1 - \Phi_{mu_1,\sigma_1}(\bar{x}_2(n_1+1) - \bar{x}_1 n_1) > \frac{\mu_* - \mu_1}{\mu 1 - \mu_2} \tag{1}$$

(recall that $\bar{x}_1 = \frac{m_1}{n_1}$ by definition). Otherwise, the teacher should pull Arm_*. We can then compute the expected value of the optimal action as:

- If $\bar{x}_1 > \bar{x}_2$, $\text{EV}_{nt} = \mu_* + \mu_1$
- Else, if the optimal action is to teach, $\text{EV}_t = \mu_1 + \mu_2 \Phi_{mu_1,\sigma_1}(\bar{x}_2(n_1+1) - \bar{x}_1 n_1)) + \mu_1(1 - \Phi_{mu_1,\sigma_1}(\bar{x}_2(n_1+1) - \bar{x}_1 n_1))$
- Else $\text{EV}_{nt} = \mu_* + \mu_2$.

Since there are readily available packages, for example in Java, for computing $\Phi_{\mu_1,\sigma_1}(y)$, this result can be considered a closed form solution for finding the optimal teacher action and its expected value when $r = 1$.

3.4.2. $\bar{\boldsymbol{x}}_{\mathbf{1}} < \bar{\boldsymbol{x}}_{\mathbf{2}}, \boldsymbol{r} \geq \mathbf{2}$

In contrast, when $r > 1$, there is no such closed form method for finding the optimal action. Rather, integrals over functions need to be estimated numerically. For example, consider the case in which $r = 2$. In this case, EV_{nt} and EV_t can be estimated numerically by sampling from the arms' distributions and using the results to compute a sample EV based on the appropriate case from the expected value computation from Section 3.4.1. The resulting sample EV's can then be averaged. Doing so is akin to computing the value of a double integral (since the definition of Φ also includes an integral). As r increases, the inefficiency of this process compounds: for each sample, and at each round, it is necessary to estimate the values of both EV_{nt} and EV_t so that the optimal action from that point can be determined. In a sense, the value of a nested integral, with a total of r levels of depth, needs to be computed. Alternatively, the continuous distribution can be approximated with a discrete distribution and then solved as in Section 3.3. To date, we have not been able to characterize anything more formal or concrete about this case. Instead we discuss some conjectures and heuristics in the following section.

3.4.3. Numerical Results and Experiments

Even if we cannot practically determine in general what the teacher's optimal action is, it may be possible to find some reasonable heuristics. To this end, in this section we consider the following questions, the first of which is parallel to the first question considered in Section 3.3.5:

1. Are there any rules or patterns in the optimal action as a function of r (when all other parameters are held constant)?
2. How do various teacher heuristics compare to one another in performance?

First, just as in the binary case, intuition suggests that increasing r should make it more beneficial to teach since there is more time for the added information to be used by the learner. However again, we can find a counterexample even with $r = 1$ and 2.

Consider the case in which $(\mu_*, \sigma_*) = (10, 0), (\mu_1, \sigma_1) = (9, 2)$, and $(\mu_2, \sigma_2) = (7, 2)$. Suppose that the learner has observed Arm_1 being pulled once when it got a payoff of 6.99 ($\bar{x}_1 = 6.99, n_1 = 1$), and it observed Arm_2 once for a payoff of 8 ($\bar{x}_2 = 8, n_2 = 1$).

With these values it is barely *not* worth it for the teacher to teach with $r = 1$. That is, with these values, Inequality 1 is not satisfied, but if $\bar{x}_1$ were 7.01, then it would be satisfied. Thus we know with certainty that the teacher's optimal action is Arm_*.

When $r = 2$, we can determine experimentally what the teacher's optimal action is by averaging the results of multiple trials when the teacher starts by teaching vs. not teaching and then acting optimally in the last round. In this case, when averaging over 2000 samples, the teacher reliably does better teaching (34.4 average return over the last 2 rounds) than when not teaching (34.2). Though the numbers are close and have high variance within a set of 2000 samples, the result is robust across multiple sets of 2000 samples.

When doing these experiments, we can gain a deeper understanding by considering the average situation after the teacher and learner have each taken one action, such that there is one more round remaining. First, consider the case in which the teacher does not teach with two rounds remaining. Thus it selects Arm_* and the learner selects Arm_2. Though the teacher's action has no impact on the relationship between $\bar{x}_1$ and $\bar{x}_2$ for the final round, the learner's action does. In one set of 2000 samples, the status after the first round was as follows:

- $\bar{x}_1 > \bar{x}_2$: 29.5%
- $\bar{x}_1 < \bar{x}_2$, Inequality 1 true (worth teaching): 39.2%
- $\bar{x}_1 < \bar{x}_2$, Inequality 1 false (not worth teaching): 31.4%

Weighting all three cases by their frequency, the total average expected value during the last round was 17.737.

On the other hand, when the teacher selects Arm_1 with two rounds remaining, we see the following breakdown after the first round:

- $\bar{x}_1 > \bar{x}_2$: 64.0%
- $\bar{x}_1 < \bar{x}_2$, Inequality 1 true (worth teaching): 14.1%
- $\bar{x}_1 < \bar{x}_2$, Inequality 1 false (not worth teaching): 22.0%

Again weighting the three cases by their frequency, the total average expected value during the last round was 18.322.

So in this case, after teaching in the second last round, the expected value of the last round is higher than when not teaching in the second last round.

Most of this advantage comes because it is more likely that $\bar{x}_1 > \bar{x}_2$ prior to the final round. This advantage makes up for the slight cost of teaching in the initial round.

Though perhaps typical, it is not always the case that increasing r increases the benefit of teaching. Just as we found in the binary case in Section 3.3.5, in the Gaussian case it is also possible that increasing r from 1 to 2 and holding all other parameters constant could cause a switch from teaching being optimal to not teaching being optimal.

For example, consider the case in which $(\mu_*, \sigma_*) = (2.025, 0)$, $(\mu_1, \sigma_1) = (2, 1)$, and $(\mu_2, \sigma_2) = (1, .0001)$. Suppose that $\bar{x}_1 = 3, n_1 = 1$, and $\bar{x}_2 = 3.4, n_2 = 1$. Inequality 1 holds because the cost of teaching, $\mu_* - \mu_1 = .025$, is less than the potential benefit, $\mu_1 - \mu_2 = 1$, times the probability that teaching will succeed, $1 - \Phi_{\mu,\sigma}(.38) = .036$. Thus the optimal action when $r = 1$ is Arm_1.

However with two rounds remaining, the optimal action is Arm_*. Again considering sets of 2000 samples, the expected value of teaching is reliably 8.85 (4.025 of which comes from the last round), while that of not teaching is 8.70 (3.750 from the last round). Intuitively in this case, teaching is generally unlikely to help, and is also generally unnecessary: the learner will "teach itself" that Arm_1 is better than Arm_2 when it selects Arm_2 the first time. However with just one round remaining, it is worth it for the teacher to take a chance that teaching will help because even though the odds are low, so is the cost.[8]

Second, in addition to being of theoretical interest, the phenomenon that increasing r can cause teaching to be less worthwhile also has practical import, in particular in the context of considering possible heuristics for the teacher when $r > 1$. Specifically, we tested the following three heuristic teacher strategies under a variety of conditions:

1. Never teach;
2. Teach iff $\bar{x}_1 < \bar{x}_2$;
3. Teach iff it would be optimal to teach if $r = 1$ and all other parameters were unchanged.

Heuristic 3 would be particularly appealing were it the case that increasing r always made teaching more worthwhile. As it is, we found that none of

[8]Thanks to Daniel Stronger for this example.

these heuristics consistently outperforms the others.

Specifically, we compared the three heuristics under the six possible relationships of $\mu_1, \mu_2, \bar{x}_1$, and $\bar{x}_2$ subject to the constraint that $\bar{x}_1 < \bar{x}_2$ (e.g. $\bar{x}_1 < \bar{x}_2 < \mu_1 < \mu_2$, or $\mu_1 < \bar{x}_1 < \mu_2 < \bar{x}_2$). For each comparison, we sampled μ_1 and μ_2 uniformly at random from $[0, 10]$, setting the lower of the two draws to be μ_2; sampled σ_1 and σ_2 uniformly at random from $[0, 1]$; set $n_1 = n_2 = 1$; and drew m_1 and m_2 from their respective distributions until the required relationship between $\mu_1, \mu_2, \bar{x}_1$, and $\bar{x}_2$ was satisfied. Holding all of these values constant, we then tested all three heuristics for 9 different values of r ranging from 2 to 500.[9] Each test consisted of 10 trials, with the results being averaged. We then repeated the entire process with new draws of $\mu_1, \mu_2, \bar{x}_1$, and $\bar{x}_2$ five times for each of the six relationships.

An analysis of these results revealed that each heuristic outperforms the other two under some circumstances. Finding more sophisticated heuristic and/or principled teacher strategies that perform consistently well is one of the main open directions of future work in the context of this research.

3.5. More than Three Arms

To this point, we have assumed that the learner has only two arms available and the teacher has only one additional arm. In this section we generalize to the case in which there are more than three arms total.

Observe that adding additional arms that are only available to the teacher does not change anything. Only the best such arm (the one with the greatest expected value) should ever be considered by the teacher. We continue to call that arm Arm_*; the others can be ignored entirely.

Thus, we focus on the case in which there are additional arms available to both the teacher and the learner: Arm_1, Arm_2, $\ldots$, Arm_z such that $\mu_1 > \mu_2 > \ldots > \mu_z$. In brief, the results we presented in Sections 3.2–3.4 all extend naturally to this more general case.We generalize the notation from Section 3.1 in the obvious ways.

3.5.1. It can be beneficial for the teacher to pull Arm_1–Arm_{z-1}

Now it is not only Arm_1 that the teacher needs to consider teaching with. For instance, consider any Arm_c, $1 \leq c < z$. By way of intuition, suppose that the arms that are better in expectation than Arm_c are only barely so,

[9] 2,3,4,5,10,20,50,100, and 500.

and that their current sample averages ($\bar{x}$'s) are much less than $\bar{x}_c$. Suppose further that the learner would currently select Arm_{c+1} ($\bar{x}_{c+1}$ is higher than any of the other $\bar{x}$'s). It can then be best for the teacher to target elevating Arm_c's sample average so as to make it the learner's next choice.

Extending the example from Section 3.2.1, let $r = 1, \mu_* = 10, \mu_1 = 9.1, \mu_c = 9, \mu_{c+1} = 5, \bar{x}_c = 6, \bar{x}_{c+1} = 7, n_c = n_{c+1} = 1$. Let all the other sample averages $\bar{x}_i = -100, n_i = 1$. The remaining expected values can be anything subject to the constraint that $\mu_i > \mu_{i+1}$. As in Section 3.2.1, suppose that the distribution of payoffs from Arm_c is such that the probability of obtaining a value greater than 8 is $\eta > \frac{1}{2}$. Thus with probability η, after an agent selects Arm_c, its sample average will be greater than $\bar{x}_{c+1}$. Suppose further that none of the distributions of Arm_1–Arm_{c-1} are such that the probability of obtaining a value greater than 114 (as would be needed to raise the sample average over 7) is small.

Carrying through as in Section 3.2.1, it is clear that the teacher pulling Arm_c yields a higher expected team value than pulling Arm_* or any other arm. Thus the learner needs to consider pulling at least Arm_* and Arm_1–Arm_{z-1}.

3.5.2. The teacher should never pull Arm_z

The proof of Theorem 3.1 that the teacher should never pull the arm with the worst expected value extends to the case with more than two leaner arms, but becomes even slightly more subtle. The key is to consider Arm_1–Arm_{z-1} as a single arm with an irregular distribution. Since pulling Arm_z does not affect the sample averages of any of the other arms, the sequence of draws from Arm_1–Arm_{z-1} is constant regardless of whether or not there are points in time at which Arm_z appears to be best ($\bar{x}_z$ is highest). Thus throughout the proof, the v values can represent the sequence of pulls from Arm_1–Arm_{z-1}, and $S_1(n)$ and $T_1(n)$ can represent the number of pulls of those arms in the two sequences, while $S_2(n)$ and $T_2(n)$ can represent the number of pulls of Arm_z. At the end of case 2 of the proof, there will be at least one extra pull of Arm_z in sequence S corresponding to a pull of Arm_* in sequence T.

For the remainder of this section, we continue to refer to pulling Arm_* as "not teaching," but now must specify with which arm when referring to "teaching."

3.5.3. Never teach with Arm$_i$ when $\boldsymbol{\bar{x}_i > \bar{x}_j, \forall j \neq i}$

The proof of Theorem 3.2 from Section 3.2.3 generalizes directly to the following statement. The teacher should never take the action that the learner would take next on its own if the teacher were to pull Arm$_*$.

3.5.4. Do not teach when $\boldsymbol{n_1 = n_2 = \ldots = n_z = 0}$

This result carries through from Section 3.2.4. The teacher is best off selecting Arm$_*$ while the learner selects each arm for the first time, rather than selecting one of those arms itself and shortening the period of time that it takes the learner to do so. Nothing can happen in the final rounds to compensate for the lost chances to get an expected value of μ_* at the beginning.

3.5.5. No other distribution-independent constraints

Other than the constraints Sections 3.5.2–3.5.4, any action could be optimal for the teacher. For example, there *are* situations in which the teacher should teach with Arm$_j$ even when $\exists i < j$ s.t. $\bar{x}_i > \bar{x}_j$. That is, pulling Arm$_2$ may be optimal, even when $\bar{x}_1 > \bar{x}_2$.

This last fact is perhaps somewhat surprising. It arises when $r \geq 2$ and $\exists k > j$ s.t. $\mu_k << \mu_j$ and $\bar{x}_k > \bar{x}_j$ (the learner mistakenly believes that Arm$_k$ is better than Arm$_j$, when in fact it is *much* worse). Then it can be better to ensure that Arm$_j$ is pulled as many times as possible, to minimize the chance that Arm$_k$ is ever pulled. For example, if $\bar{x}_1 > \bar{x}_z > \bar{x}_2$, but the distributions of Arm$_1$ and Arm$_2$ are such that there is a chance that Arm$_1$'s sample average will dip below Arm$_z$'s, but Arm$_2$'s sample average could be first elevated above Arm$_z$'s, then it could be optimal for the teacher to teach with Arm$_2$. Similarly for any other arm other than Arm$_z$ itself.

More concretely, consider arms with binary distributions in which $p_* = .101$, $p_1 = .1$, $p_2 = .095$, and $p_3 = .0001$. Assume further that $m_1 = 1$, $n_1 = 3$, $m_2 = 1$, $n_2 = 4$, $m_3 = 7$, and $n_3 = 24$, so that $\bar{x}_1 > \bar{x}_3 > \bar{x}_2$. In this case, when there are 2 rounds remaining ($r = 2$), the expected value of selecting Arm$_2$ is higher (.3215) than the expected value of selection Arm$_*$ (.3202). We know that the teacher shouldn't select Arm$_3$ ever, nor in this case Arm$_1$, since that is the arm that the learner would select next on its own.

Similarly, one can construct an example using arms with normal distributions.[10] Let the $(\mu_*, \sigma_*) = (10, 0)$, $(\mu_1, \sigma_1) = (9, 100)$, $(\mu_2, \sigma_2) = (8, 2)$, and $(\mu_3, \sigma_3) = (-10^1 0, 1)$. Furthermore, assume that $n_1 = n_2 = n_3 = 1$ and $\bar{x}_1 = 5.02$, $\bar{x}_2 = 5$, and $\bar{x}_3 = 5.01$. Again in this case, if $r = 2$, it is best to pull Arm_2 so as to minimize the probability that the learner will ever pull Arm_3.

One commonality between the above two examples, is that it would be quite unlikely to ever get into the state described from having pulled the arms listed. That is, given that $\mu_3 = -10^{10}$, it's extremely unlikely that $\bar{x}_3$ would ever be 5.01. However, it's also possible to construct an example in which the starting state is quite likely. For the purpose of this example, we'll use simple discrete distributions of the arms (neither binary nor normal). Assume the following distributions of the arms:

Arm_*:	always yields a payoff of 1	$\mu_* = 1$		
Arm_1:	50% chance of 10 or -9	$\mu_1 = .5$	$n_1 = 2$	$\bar{x}_1 = .5$
Arm_2:	50% chance of 1 or -1	$\mu_2 = 0$	$n_2 = 1$	$\bar{x}_2 = -1$
Arm_3:	50% chance of -10^6 or 0	$\mu_3 = -500,000$	$n_3 = 1$	$\bar{x}_3 = 0$

In this case, the $\bar{x}$'s all have a 50% chance of arising after the listed number of pulls. And once again, if $r = 2$, it is best to pull Arm_2 so as to minimize the probability that the learner will ever pull Arm_3.

3.5.6. Discrete distributions, $\bar{x}_1 < \bar{x}_i$ for some $i, r = 1$

The results from Section 3.3.1 generalize directly. In particular, let Arm_i be the learner's arm with the highest sample average $\bar{x}_i$. The teacher should consider teaching with any Arm_j, $j < z, j \neq i$ such that:

1. $\frac{m_j+1}{n_j+1} > \frac{m_i}{n_i}$
2. $p_* - p_j < p_j * (p_j - p_i)$

Those are the arms with higher expected value than Arm_*. From among those arms, it should select the Arm_j with the highest expected value EV=$p_j + p_j^2 + (1 - p_j)p_i$.

3.5.7. Discrete distributions, algorithm

Similarly, the algorithm generalizes directly. Expected values and optimal actions must now be calculated for all reachable values of m_1–m_z and n_1–n_z. Since the teacher could teach with any arm other than Arm_z, the ranges of the

[10]Thanks to Reshef Meir for this example.

variables m_1–m_{z-1} and n_1–n_{z-1} match those of m_1 and n_1 in Section 3.3.2. The range of m_z matches that of m_2 in Section 3.3.2, and n_z is similar to n_2, except that the two occurrences of $n_1 - N_1$ (both inside "max" operators) need to be changed to $\sum_{i=1}^{z-1} n_i - N_i$.

Beyond that, the inner loop need only be extended to compute and compare the expected values of all z possible teacher actions, in all cases storing the maximum such value.

3.5.8. Discrete distributions, algorithm analysis and generalization

Both the memory and runtime bounds of the extended algorithm generalize naturally to $O(R^{2z+1})$. The extended algorithm generalizes to arbitrary success and failure payoffs exactly as in Section 3.3.4.

3.5.9. Normal distributions, $\boldsymbol{\bar{x}_1 < \bar{x}_i}$ for some $\boldsymbol{i, r = 1}$

Exactly as the results from Section 3.3.1 generalize as described in Section 3.5.6, the results from Section 3.4.1 generalize as well. Specifically, let Arm_i be the learner's arm with the highest sample average $\bar{x}_i$. The teacher should consider teaching with any Arm_j, $j < z, j \neq i$ such that the equivalent of Inequality 1 is satisfied:

$$1 - \Phi_{mu_j,\sigma_j}(\bar{x}_i(n_j + 1) - \bar{x}_j n_j) > \frac{\mu_* - \mu_j}{\mu 1 - \mu_i} \tag{2}$$

Those are the arms with higher expected value than Arm_*. From among those arms, it should select the Arm_j with the highest expected value EV= $\mu_j + \mu_i \Phi_{mu_j,\sigma_j}(\bar{x}_i(n_j + 1) - \bar{x}_j n_j)) + \mu_j(1 - \Phi_{mu_j,\sigma_j}(\bar{x}_i(n_j + 1) - \bar{x}_j n_j))$.

3.5.10. Normal distributions, $\boldsymbol{\bar{x}_1 < \bar{x}_i}$ for some $\boldsymbol{i, r \geq 2}$

Similarly to Section 3.4.2, we do not have any closed form solution to this case.

3.6. Sequential Action Summary

A brief summary of the results from this section on sequential (turn-taking) scenarios with differing abilities is as follows.

Arms with any payoff distributions:

- $\boldsymbol{\bar{x}_1 > \bar{x}_2}$: do not teach
- $\boldsymbol{n_1 = 0}$ and/or $\boldsymbol{n_2 = 0}$: do not teach

Arms with discrete payoff distributions:

- Polynomial algorithm for optimal teacher action

Arms with normal payoff distributions:

- $\bar{x}_1 < \bar{x}_2, r = 1$: closed form solution for optimal teacher action
- $\bar{x}_1 < \bar{x}_2, r \geq 2$: only numerical solutions

4. Related Work

The broad context for this research is ad hoc teams in which teammates need to work together without any prior coordination. This perspective is complementary with most prior treatments of agent teamwork. For example, frameworks such as STEAM [5], and BITE [6] define explicit coordination protocol and languages. SharedPlans [7] specifies the intentions the members of the team must all adopt and about which they all must be mutually aware. In applications such as the annual RoboCup robot soccer competitions, entire teams of agents are designed in unison, enabling explicit pre-coordination via structures such as "locker room agreements" [8].

The concept of ad hoc *human* teams has arisen recently in military and industrial settings, especially with the rise of outsourcing. There have also been autonomous agents developed to help support human ad hoc team formation [9, 10, 11]. This work relies on an analysis of the sources of team variability, including member characteristics, team characteristics, and task characteristics [10]. In addition, software agents have been used to support the *operation* of human teams [12], and for distributed information gathering from distinct, otherwise independent information sources [13].

There are only a few examples of prior research that we are aware of that take a perspective similar to our ad hoc team perspective. The most closely related examples have been referred to as *pickup teams* [14] and *impromptu teams* [15]. Both pickup teams and impromptu teams are defined in the same spirit as our ad hoc teams. However both focus on tightly coordinated tasks in which there are well-defined roles for the various agents, and therefore a higher degree of common knowledge. Pickup teams, as defined in [14] build on market-based task allocation schemes to enable heterogeneous robots to work together on highly synchronized actions. The work is implemented in a treasure hunt domain. Similarly, impromptu teams assume that the teammates, other than the impromptu player, are all members of a coherent team that actively consider the impromptu player as a part of the team. Their approach is based on a “playbook” formalism that defines roles and behaviors for each team player. That work is implemented in a robot soccer domain.

In this article, we define ad hoc teamwork very broadly, in a way that is able to accommodate the assumptions made by both pickup teams and impromptu teams, as well as scenarios that include many types of teammates. Our definition of ad hoc teamwork encompasses role-based and tightly-coupled tasks as well as loosely-coupled tasks with agents that barely interact. It also covers many types of teammates: those with which the ad hoc team player can communicate and those with which it cannot; those that are more mobile and those that are less mobile; those with better sensing capabilities and those with worse capabilities. Following on this broad definition, we then focus in on a particularly fundamental type of ad hoc teamwork, namely settings with just one teammate that has fixed and known behavior. We consider both a simultaneous, repeated action scenario (in Section 2) and a sequential, turn-taking scenario in which the agents have different action capabilities (Section 3).

Another piece of prior work that takes a perspective similar to ours is that of Brafman and Tennenholtz [16] in which they consider a teacher agent and a learner agent repeatedly engaging in a joint activity. While the learner has no prior knowledge of this activity, the teacher understands its dynamics. As in our models, the teacher's goal is also to lead the learner to adopt a particular behavior.

They focus on settings in which the agents play a 2×2 matrix game. While the teacher knows the matrix, the learner does not know the payoff function, although he can perceive the payoff he receives. For example, the teacher may try to teach the learner to cooperate in the Prisoner's dilemma game. Unlike our k-armed bandit model, Brafman and Tennenholtz consider only situations in which the outcome of their agents' actions is deterministic. This limitation makes teaching considerably easier. Brafman and Tennenholtz also mainly considered situations where teaching is not costly: the goal of their teacher is to maximize the number of times that the learner chooses the “right” action. Thus in some sense, the teacher is not “embedded” in the environment. For this problem they propose an optimal teaching policy using MDPs. For the more challenging situations where teaching is costly, as in our model, they propose a teaching policy that is evaluated via experimentation in a simple coordination game.

A recent study by Wu et al. [17] investigates the problem of online planning for ad hoc teamwork, and examine it as an optimization problem. Assuming they have access to drawing samples of team actions, they learn possible teammate's actions, modeled by a Multiagent Markov Decision Process (MMDP). This model allows the agent to choose a best response to the teammate's action. Their goal, similar to our work, is to maximize the team's joint utility. Their assumption that samples of teammates' actions are available in a simulated environment makes it impossible to use their methods in the problems described in this article, in which learning (or leading) is costly.

Liemhetcharat and Veloso [18] suggest a new method for modeling the performance of a team of agents using *synergy graphs*. In a team of heterogenous agents, the performance of several agents that are teamed up is not necessarily based only on their individual capabilities, but on how they interact as a team (synergy). The synergy graphs model this interaction. Based on its structure, a subgroup of the agents, that are most appropriate for performing a task, is chosen. Modeling interaction between team members in ad hoc teamwork can also benefit from using this synergy measure. However, in their work, Liemhetcharat and Veloso are interested in *building* an optimal team (or subteam), and not in influencing the given team to perform as well as possible (without the ability to choose specific team members for the mission).

Related to the concept of teacher/learner is also the work by Zilles et al. [19]. In their work, they seek to be sample efficient in the learning process of the learner by knowing that the samples are given by a cooperative teacher. Unlike the work presented here, they focus their control over the learner rather than on the teacher, i.e., they do not answer the question on how to better teach a cooperative agent in ad hoc teamwork, but how to better utilize information coming from a knowledgeable, cooperative source.

Also somewhat related is the recent work of Zhang et al. [20] on "environment design." Here, the controlling agent can alter aspects of the environment for a learning agent in an MDP so as to influence its behavior towards a particular goal. Once again, the controlling agent is not itself embedded in the environment and taking actions itself.

Finally, our own recent work has explored role-based approaches to ad hoc teamwork [21]; ad hoc teamwork to influence a flock of simple agents [22]; and empirical studies of ad hoc teamwork [23], including experiments with learning teammate models from observations [24].

Though there has been little other work on the ad hoc teamwork problem itself, the specific scenarios we consider touch upon vast literatures in iterated game theory and in k-armed bandits. Nonetheless, our work introduces new ways of looking at both types of formalisms. In the remainder of this section, we focus in on work that relates to each type of formalism separately.

4.1. Repeated Scenarios with Simultaneous Actions

Our work in Section 2 builds on existing research in game theory and in opponent modeling. Game theory [25] provides a theoretical foundation for multiagent interaction, and though originally intended as a model for human encounters (or those of human institutions or governments) has become much more broadly applied over the last several decades. In particular, the field of multiagent systems within artificial intelligence has adopted game theory as one of its primary tools for modeling interaction among automated agents, or interaction in mixed human-automated agent encounters [26].

There is a vast research literature covering iterated play on normal form game matrices, the overall framework that we explore in Section 2. Some of that research focuses on automated players, while other work focuses on human players. Many of these papers have examined the specific questions of what, and how, agents can *learn* when repeatedly playing a matrix game; special emphasis has been given to developing learning algorithms that guarantee convergence to an equilibrium in self-play, or that converge to playing best response against another player that is using one of a fixed set of known strategies.

For example, Powers and Shoham [27] considered multiagent learning when an agent plays against bounded-memory opponents that can themselves adapt to the actions taken by the first agent. They presented an algorithm that achieved an ϵ-best response against that type of opponent, and guaranteed a minimum payoff against any opponent. A small selection of other research on multiagent learning includes Jürgens' work on Bayesian learning in repeated games [28], Conitzer and Sandholm's work [29] on a learning algorithm that converges in self-play, Young's examination of the kinds of learning that lead to a Nash equilibrium or other types of equilibria [30], Littman's multiagent reinforcement learning algorithm [31], and Chakraborty and Stone's [32] presentation of an algorithm that aims for optimality against any learning opponent that can be modeled as a memory-bounded adversary. Shoham et al. provide a survey of multiagent reinforcement learning [33].

There are also a large number of articles in the economics and game theory literature on repeated matrix games, also often focused on issues related to reaching equilibria. Hart and Mas-Colell [34] presented an adaptive procedure that leads to a correlated equilibrium among agents playing a repeated game, while Neyman and Okada [35] considered two-player repeated games in which one agent, with a restricted set of strategies, plays against an unrestricted player (and considered the asymptotic behavior of the set of equilibrium payoffs).

Axelrod [36] conducted several well-known computer tournament experiments on repeated play of the Prisoner's Dilemma, pitting computer programs playing various strategies against one another. These strategies were evaluated on the basis of their overall success in the tournaments, as well as other factors (e.g., given a population that is playing some strategy, what is that population's resistance to invasion by a competing strategy, assuming that winning strategies reproduce more successfully).

A popular game theoretic model that may lead agents to converge to an equilibrium is that of fictitious play [37], in which agents play best response under the assumption that their opponents have a unchanging (though possibly mixed) strategy. At each step, each agent imagines that others will play as they have played up to this point, and responds according to the empirical frequency of those opponents' past play. Young [38, 39] explored a related concept called "adaptive play", which similarly models a dynamic process whereby agents, each employing bounded-memory best-response algorithms based upon a random sample of past plays of the game, may gradually move towards an equilibrium (the specific choice of equilibrium by a population of agents may be affected by small amounts of noise, which are part of the adaptive play model).

Much of the research above focused specifically on automated agent repeated play; similar questions have been taken up by researchers who have considered repeated play among humans. For example, a seminal paper by Nyarko and Schotter [40] investigated the beliefs that humans have as they repeatedly play a constant-sum two-person game; the authors elicited the players' beliefs during play, and factored those beliefs into the model of how players chose their moves.

All of the research mentioned above differs in fundamental ways from the work presented in this article. First, our model assumes that the agents are cooperative; we are not considering general payoff matrices that model opponent rewards, nor zero sum games. Second, we are not examining the learning behavior of our agent (or agents), but rather are assuming that one agent is playing some variant on a best-response strategy, and its partner is fashioning its play accordingly, for their mutual benefit. This lack of symmetry between agents' algorithms distinguishes our model from that of, for example, the fictitious play model as well as Young's adaptive play model. In addition, we are exploring different aspects of the interaction than do those models.

More closely related to our current work is research by Claus and Boutilier [41] that, first of all, considers cooperative agents with identical payoffs, and then considers how (using reinforcement learning) these agents can converge to the maximal payoff. That research considers the *dynamics* of the convergence (e.g., speed of convergence), and the sliding average rewards that agents accrue as they explore their payoffs. What distinguishes our work is its emphasis on the path through matrix payoffs imposed by a reasoning *Agent A*, faced with a best-response *Agent B* as its partner. The process of movement through the matrix is deliberate and optimal, the path "searched-for," based on knowledge of partner behavior, rather than the Q-learning techniques explored by Claus and Boutilier.

Indeed, the algorithms in this article make an explicit assumption that the teammate observing the agent is playing a best-response policy to the observed actions of the agent. In doing so, the agent is actually planning its actions *intending* for them to be observed and interpreted. *Intended plan recognition* (in contrast to *keyhole recognition*) is the term used when the observed agent knows that it is being observed, and is acting under the constraints imposed by this knowledge [42].

Much of the work on planning for intended recognition settings has focused on natural language dialogue systems. Here, one agent plans its utterances or speech acts intending for them to be interpreted and understood in specific ways. Seminal work in this area was carried out by Sidner [43] and later Lochbaum [44], who have focused on collaborative dialogue settings. However, unlike our work, their focus is on the interpretation (the recognition), rather than on the planning of observed actions. Lochbaum later investigated planning [45], but here the focus was on natural language, and did not involve any notion of game-theory.

The SharedPlans framework [46, 7, 47] summarizes the set of beliefs and intentions needed for collaborative activity, and provides the rationale for the process of revising beliefs and intentions. Partial SharedPlans allows agents, as in an ad hoc team, to differ not only in their beliefs about the ways to perform an action and the state of the world, but also in their assessments of the ability and willingness of an individual to perform an action. However, while SharedPlans specifies a logical framework which provides guidelines informing agent design, it does not provide detailed algorithms for specific cases, such as the cases covered in this article.

Because our Algorithm 1 is—to a limited extent—reasoning about the teammate reasoning about itself, it is in fact engaged in a special case of *recursive modeling*. Among the first to consider such deep nesting were Vidal and Durfee (in particular, their Recursive Modeling Method—RMM [48]) and Gmytrasiewicz and Durfee (e.g., [49]). The first focused on algorithms that allow the agent to decide how deep to continue the recursive modeling, such that it does not spend precious resources on recursive modeling that does not provide gains. The latter focused on efficient representations that allow rational modeling of others, including recursion. Ultimately, however, it is the case that it is *not* always beneficial to engage in deeper nesting of models [50]. We thus choose to leave this issue open for future investigation. Specifically, an interesting question is what happens when the teammate is also trying to select actions that would cause the agent to shift policies. In this case, our agent would have to address 3-level recursive modeling.

Han et al. [51] examined a closely related problem of controlling the collective behavior of self-organized multi-agent system by one agent. They consider self organized teams of physically interacting agents, concentrating on flocking of birds, where their goal is to design an agent, denoted as a *shill agent*, that will be able to gradually change the heading of the entire team to a desired heading. They evaluate the system in terms of physical capabilities of the shill agent and the team (velocity, initial heading) and provide theoretical and simulation results showing that it is possible, under some conditions, for one agent to change the heading of the entire team. Different from our approach, they do not consider game theoretic evaluation of the individual actions and their impact on the team behavior, nor do they examine uncertain behavior.

4.2. Sequential Action Scenarios with Differing Abilities

In the context of our k-armed bandit instantiation of ad hoc teams from Section 3, our research is characterized by cooperative agents with asymmetric information and asymmetric capabilities which are acting in an uncertain environment in which both agents are embedded in the environment (their actions affect the team's payoff) but the agents cannot communicate directly. To the best of our knowledge, no prior research meets all of the above characteristics. Here we mention the most closely related work that has some of these characteristics.

As in the matrix game setting, some of this related work has been done within the context of multiagent reinforcement learning, a generalization of k-armed bandits in which there are multiple states where the actions have different effects. For example, Lin [52] describes an approach to integrating teaching with reinforcement learning in which the learner is given some successful action trajectories. In the survival task studied by Lin, teaching did not make a significant improvement, but this approach appeared beneficial with learning robots [53]. The teacher in Lin's model is not embedded in the environment and it does not face the dilemma of exploitation versus teaching. Similarly, most other work on imitation learning or learning by demonstration similarly considers scenarios in which the teacher, sometimes a human, is not embedded in the environment, but rather tries to train the learner to improve its individual actions, e.g., [54, 55, 56, 57].

There are two sources of incomplete information in cooperative reinforcement learning: whether the agents can observe the state of the environment and whether they are able to observe the reward obtained by the other agents. Schneider et al. [58] considered distributed reinforcement learning, in which agents have complete information about the state of the environment, but only observe their own reinforcement reward. They investigate rules that allow individual agents to share reinforcement with their neighbors. Peshkin et al. [59] considered the complementary problem in which the agents receive a shared reward but have incomplete information about the world state. They propose a gradient-based distributed policy search method for cooperative games.

Schaerf et al. [60] study the process of multiagent reinforcement learning in the context of load balancing of a set of resources when agents cannot observe the reward obtained by others. They show that when agents share their efficiency estimation of the different resources (as in our model) the system efficiency may not improve, and might even be harmed. The reason for this findings is that Schaerf et al.'s agents compete over the resources. Thus, having a better picture of the system leads to all of them competing over the "good" recourses and thus decreasing the overall performance of the system. They conclude that a better load-balancing mechanism is needed when communication is possible.

There are many other approaches for cooperative multiagent learning (see surveys at [61, 62, 63]). But to the best of our knowledge, none covers any work with cooperative agents with asymmetric information and asymmetric capabilities which are acting in an uncertain environment in which the teacher is embedded in the environment but the agents cannot communicate.

The k-armed bandit problem has been extensively studied (see a survey at [64]), but also in this literature we are not familiar with any work that considered a teacher and a student with asymmetric capabilities and information who aim to maximize the joint reward. There are several models that have been considered in which players can observe the choices or the outcomes of other players. Such models have been used for modeling experimentation in teams. In these settings, as in ours, a set of players choose independently between the different arms. The reward distributions of each arm is fixed, but characterized by parameters that are initially unknown to the players. Most of the works consider the case where each player tries to maximize its own expected reward and thus if the outcome of other players are observable a free riding problem is created since each wants the others to try the risky arms (e.g., [65, 66]).

Aoyagi [67] studies a model of a two-armed bandit process played by several players, where they can observe the actions of other players, but not the outcome of these actions. He proved that under a certain restriction on the probability of distribution of the arms, the players will settle on the same arm in any Nash equilibrium of the game. This shows that each agent learns from the behavior of the other agents, even if communication is not possible.

A study in which the agents are cooperative is presented in [68]. They study a two-armed bandit situation with multiple players where the risky arm distributes lump-sum payoffs according to a Poisson process. They show that if the agents try to maximize the average expected payoff then the efficient strategy is one with a common cut-off for which if the belief about the risky arm is above the cut-off all the agents will choose the risky arm. Otherwise, all of them will choose the other arm.

Situations in which the agents do not have symmetric roles are studied in the context of the principal-agent problem where the arms of the bandit are analogous to different effort levels of the agent and the principal would like the agent to choose the highest level effort [69]. The principal has the option to obtain the true value of each arm. It is shown that, if the information acquisition decision is observable by the agent, in every refined equilibrium, the principal delays information acquisition until the agent's beliefs become pessimistic enough. If this decision is unobservable, the timing of the information acquisition is indeterminate. This setting is much different than ours because of the conflicting utilities of the principal and the agent.

Multi-player multi-armed bandit problems have been also used to model the challenges facing users of collaborative decision-making systems such as reputation systems in e-commerce, collaborative filtering systems, and resource location systems for peer-to-peer networks. Here the main challenge is deciding which player to trust [70]. We assume that the learner sees the actual outcomes of the teacher and no issues of trust arise.

There are several additional approaches taken in game-theoretic research that have potential relevance to our overall scenario of collaboration in ad-hoc settings, although they remain outside the scope of our current work.

Cooperative (coalitional) game theory is concerned with groups of self-interested agents that work together to increase their utility; much of the research in this area is concerned with how a group's "profit" from joint activity can be divided among its members in a way that motivates them to remain in the group. The models used differ from those explored in this paper, but future work could profitably explore connections between these areas. Classic foundational work in this area includes [71], but there continues to be important research in recent years exploring new models of coalitional games (including from a computational perspective) [72].

Finally, there are classic game theory solution concepts that appear to have relevance in future research on ad hoc teams. For example, Aumann's notion of "strong Nash equilibrium" [73], a Nash equilibrium where no coalition can cooperatively deviate in a way that benefits all members assuming that non-member actions are fixed (i.e., an equilibrium defined in terms of all possible coalitional deviations, rather than all possible unilateral deviations), could be applied to interactions among agents in ad hoc encounters. In addition, Aumann's later solution concept of "correlated equilibrium" [74], where agents do not want to deviate from a strategy recommended by (or associated with) the value of a public signal (assuming that others do not deviate), could also be applied to ad hoc cooperation.

5. Summary and Discussion

The main contributions of this article are in the contexts of two specific instantiations of ad hoc teamwork chosen to represent the simplest, most fundamental cases. Specifically, we focused our attention on cases with a single teammate that exhibits fixed and known behavior, and then examined two variations on this theme. First, in Section 2, we considered simultaneous, repeated action settings by adopting the iterated matrix game formalism.

Second, in Section 3, we considered a turn-taking scenario by adopting, and adapting, the k-armed bandit formalism.

In both cases, we proved several theorems regarding situations in which we know which actions are or cannot be optimal for the ad hoc team agent. In both cases, we supplemented our theoretical results with some experiments analysis designed to test the aspects of the problems that were not analyzable theoretically.

First, we introduced (Section 2) a novel game theoretic formulation for modeling ad hoc teamwork for simultaneous decision making. We focused on the case in which an intelligent agent interacts repeatedly in a fully cooperative setting with a teammate that responds by selecting its best response to a fixed history of actions, possibly with some randomness. Based on its teammate's behavior, the intelligent agent can *lead* it to take a series of joint actions that is optimal for their joint long-term payoff. The length of this series was proven to be linear in the minimal number of actions of agent A or B when B's memory is of size 1, leading to a polynomial time complexity for determining the optimal set of actions for the ad hoc agent. When B bases its decisions on a longer memory size, this time complexity cannot be guaranteed. Specifically, we have shown that determining the maximal size of an optimal series of joint actions is NP hard.

We then presented (Section 3) a multiagent cooperative k-armed bandit for modeling sequential decision making in ad hoc teamwork. Here, the agents have different knowledge states and different action capabilities. We have studied in detail the task of a *teacher* that knows the payoff distributions of all of the arms as it interacts with a learner that does not know the distributions, and that can only pull a subset of the arms. The teacher's goal is to maximize the expected sum of payoffs as the two agents alternate actions. At any point, it can either exploit its best available action or increase the learner's knowledge by demonstrating one of the learner's actions. Within the specific scenario examined in this article, we proved several theorems regarding situations in which we know which actions are or cannot be optimal for the teacher. We then narrowed our focus to two different types of probability distributions for the arms. For discrete distributions, we presented a polynomial memory and time algorithm for finding the teacher's optimal action. When the arms have Gaussian distributions, we can only find the optimal action efficiently when there is one round left. In both cases we augment the theoretical results with some experimental analysis using our fully-implemented algorithms.

Our analysis—both in matrix game representation and in the k-armed bandit— opens up various exciting directions for future research. In both models of ad hoc teamwork, it is assumed that the ad hoc agent is well aware of the its teammate behavior (although little of our analysis relies on the fact that agent B is following a specific policy). Examining unknown behavior is a key factor in ad hoc teamwork, that should be addressed in the future. Similarly, leading and teaching more sophisticated agents—those that may explore independently—is also an important future direction. Our current approaches are limited to leading or teaching one teammate. Facing multiple teammates in ad hoc settings is a fundamental problem that will open various interesting research directions in the future, that include, other than the simplest, yet challenging, case of multiple agents as described in this article, also multiple possible teammate behavior, uncertainty in teammate behavior and more (note that initial results for leading multiple teammates in ad hoc settings can be found in [75]). In addition, our proposed algorithm for leading a teammate is exponential in the teammate's memory size, making solutions to interaction scenarios with more than a few possible actions per agent intractable. Heuristics enabling a streamlining of this algorithm would be very useful.

Many other generalizations to this cooperative k-armed bandit are possible. For example, we have verified that at least some of our results can be extended to the discounted, infinite horizon case [76]. Specifically, we verified that in the 3-arm case, the teacher should still consider pulling Arm_1, but should never pull Arm_2, and that it should never pull Arm_1 when $n_1 = 0$ and/or $n_2 = 0$. The results for more than three arms from Section 3.5 were also verified in the discounted, infinite horizon case. One could also consider arms with additional types of distributions, or types of distributions that differ among the arms (e.g. some discrete and some Gaussian). Additionally, our algorithm for computing the optimal teaching algorithm is exponential in the number of arms. Exploring possible approximation algorithms could be beneficial.

In the broader context, this research is just one step towards the long-term goal of creating a fully capable ad hoc team player. In order to achieve this goal, many more studies of this magnitude will be needed that consider situations in which, for example, there are more than two teammates, the teammates can communicate directly, the teammates' behaviors are not fully known, or some teammates have *more* knowledge and/or capabilities than our agent. We intend to follow up on these challenges in our future research and hope that this research will inspire others to also work towards the eventual creation of fully general ad hoc team players.

Appendix A. NP-hardness of finding S^*'s when *mem* > 1

In Section 2.2.2, we examined the complexity of finding the optimal (lowest cost) path through a matrix when *Agent B*'s *mem* > 1. Here we prove that the problem is NP-hard by a reduction from the Hamiltonian Path problem:[11] Given an n-node unweighted, undirected graph G, an initial node and a destination node, is there a simple path from initial to destination of length n? That is, can we visit each node exactly once? This decision problem is NP-complete.

Here we will show that if it were possible to find S^* for a given matrix M with *Agent B*'s *mem* > 1 (as defined in Section 2) in polynomial time, then it would also be possible to find a Hamiltonian path in polynomial time. To do so, we assume that we are given an n-node graph G such that $G_{ij} = 1$ if and only if there is an edge in G connecting nodes i and j. Otherwise, $G_{ij} = 0$. We construct a matrix M in a particular way such that there is a path through the matrix of cost (as per Section 2) no more than a target value of $n * (n^4 - 1)$, if and only if there is a Hamiltonian Path in graph G. Note that we focus on NP-completeness of the decision problem, which establishes NP-hardness of the optimization problem (since the optimal cost path through the matrix answers the question of whether or not there exists a path with cost less than $n * (n^4 - 1)$. Note also that, as required, the construction of the matrix can be done in time polynomial in all the relevant variables.

We let *Agent B*'s *mem* $= n$ and we construct Matrix M as follows.

- *Agent A* has $(n-1)*n+2$ actions. The first action is a "start" action, and *Agent B*'s memory is initialized to n copies of that action. Each of the next $(n-1)*n$ actions represents a combination (i, t) of a node i in the graph and a time step $t \geq 2$. M's payoffs will be constructed so that if the sequence satisfying the maximum cost requirement in M (if any) includes action (i, t), then the corresponding Hamiltonian path passes through node i on timestep t. Finally, there is a "done" action to be taken at the end of the path.

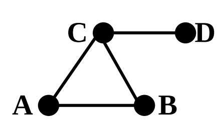
[11] Thanks to Michael Littman for the idea behind this proof.

- *Agent B* has $n * n + n + 1$ actions. The first $n * n$ actions are similar to *Agent A*'s: one for each combination of $j \in G$ and $t \geq 1$. If the satisfying sequence through M includes *Agent B* taking action (j, t), then the Hamiltonian path visits node j at time t. The next n actions are designed as "trap" actions which *Agent B* will be induced to play if *Agent A* ever plays two actions corresponding to the same node in the graph: actions (i, s) and (i, t). There is one trap action for each node, called action j. Finally, the last action is the "done" action to be played at the end of the sequence.

- M's payoffs are constructed as follows, with the nodes named as indicated in the bullets above. The initial node in the Hamiltonian path (the one visited on time step 1) is called "initial."

a)	$M[(i,t+1),(j,t)]$	$=$	1	if $G_{ij} = 1$
b)	$M[(i,t+1),(j,t)]$	$=$	$-n^5$	if $G_{ij} = 0$
c)	$M[(i,t),(i,t)]$	$=$	tn	
d)	$M[(i,t),(j,s)]$	$=$	$-n^5$	if $t \geq s$
e)	$M[(i,t),(j,s)]$	$=$	0	if $t < s$
f)	$M[(i,t),i]$	$=$	$tn - \frac{1}{3n}$	
g)	$M[(i,t),j]$	$=$	0	
h)	$M[(i,t),\text{done}]$	$=$	0	
i)	$M[\text{start},(\text{initial},1)]$	$=$	1	
j)	$M[\text{start},\text{initial}]$	$=$	$\frac{1}{2}$	
k)	$M[\text{start},\text{done}]$	$=$	$-n^4$	
l)	$M[\text{start},j]$	$=$	0	for all actions j other than initial and done
k)	$M[\text{done},(j,n)]$	$=$	1	
l)	$M[\text{done},(j,t)]$	$=$	$-n^5$	if $t < n$
m)	$M[\text{done},\text{done}]$	$=$	n^4	

For example, for this 4-node graph, with A given as the initial node of a potential Hamiltonian path,



the resulting matrix M would be constructed as follows (with $n = 4$).

M	A,1	A,2	A,3	A,4	B,1	B,2	B,3	B,4	C,1	C,2	C,3	C,4	D,1	D,2	D,3	D,4	A	B	C	D	done
start	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{1}{2}$	0	0	0	$-n^4$
A,2	$-n^5$	$2n$	0	0	1	0	0	0	1	0	0	0	$-n^5$	0	0	0	$2n-\frac{1}{3n}$	0	0	0	0
A,3	$-n^5$	$-n^5$	$3n$	0	$-n^5$	1	0	0	$-n^5$	1	0	0	$-n^5$	$-n^5$	0	0	$3n-\frac{1}{3n}$	0	0	0	0
A,4	$-n^5$	$-n^5$	$-n^5$	$4n$	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	$-n^5$	0	$4n-\frac{1}{3n}$	0	0	0	0
B,2	1	0	0	0	$-n^5$	$2n$	0	0	1	0	0	0	$-n^5$	0	0	0	0	$2n-\frac{1}{3n}$	0	0	0
B,3	$-n^5$	1	0	0	$-n^5$	$-n^5$	$3n$	0	$-n^5$	1	0	0	$-n^5$	$-n^5$	0	0	0	$3n-\frac{1}{3n}$	0	0	0
B,4	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	$-n^5$	$4n$	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	$-n^5$	0	0	$4n-\frac{1}{3n}$	0	0	0
C,2	1	0	0	0	1	0	0	0	$-n^5$	$2n$	0	0	1	0	0	0	0	0	$2n-\frac{1}{3n}$	0	0
C,3	$-n^5$	1	0	0	$-n^5$	1	0	0	$-n^5$	$-n^5$	$3n$	0	$-n^5$	1	0	0	0	0	$3n-\frac{1}{3n}$	0	0
C,4	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	$-n^5$	$4n$	$-n^5$	$-n^5$	1	0	0	0	$4n-\frac{1}{3n}$	0	0
D,2	$-n^5$	0	0	0	$-n^5$	0	0	0	1	0	0	0	$-n^5$	$2n$	0	0	0	0	0	$2n-\frac{1}{3n}$	0
D,3	$-n^5$	$-n^5$	0	0	$-n^5$	$-n^5$	0	0	$-n^5$	1	0	0	$-n^5$	$-n^5$	$3n$	0	0	0	0	$3n-\frac{1}{3n}$	0
D,4	$-n^5$	$-n^5$	$-n^5$	0	$-n^5$	$-n^5$	$-n^5$	0	$-n^5$	$-n^5$	1	0	$-n^5$	$-n^5$	$-n^5$	$4n$	0	0	0	$4n-\frac{1}{3n}$	0
done	$-n^5$	$-n^5$	$-n^5$	1	$-n^5$	$-n^5$	$-n^5$	1	$-n^5$	$-n^5$	$-n^5$	1	$-n^5$	$-n^5$	$-n^5$	1	$-n^5$	$-n^5$	$-n^5$	$-n^5$	n^4

Following a path through the matrix that corresponds to a Hamiltonian path (if one existed) would give payoffs of 1 at every step until reaching m^* (n^4) and staying there forever. Thus the cost of the n-step path would be $n * (n^4 - 1)$.

Because there is no positive payoff in the matrix greater than n^2, any path longer than n steps must have a cost of at least $(n+1)(n^4 - n^2) = n^5 + n^4 - n^3 - n^2 > n^5 - n = n * (n^4 - 1)$. In other words, if there is a path through the matrix corresponding to a Hamiltonian path in the graph, then any longer path through the matrix must have higher cost.

Furthermore, the matrix is carefully constructed such that any diversion from the path corresponding to a Hamiltonian path either will get a payoff of $-n^5$ on at least one step (which by itself makes the target cost impossible to reach), will prevent us from getting one of the 1's, or else will make it so that the path to (done,done) will require more than n total steps. In particular, if *Agent A* ever takes two actions that lead *Agent B* to select a trap action, then *Agent B* will not take a different action until the $n+1$st step after the first action that led to the trap, causing the path to (done,done) to be at least $n+2$ steps long. By this construction, it follows trivially also that if there existsk a Hamiltonian path in G, then there is a path of cost $\leq n * (n^4 - 1)$ in the matrix.

In this context, the purpose of the numbers in the graph, as indicated by the list of items a)–m) above can be understood as follows.

a) These payoffs are the 1's for each "correct" step in the path

b) These large negative payoffs prevent taking a step when there is no corresponding edge in the graph.

c) These payoffs lure *Agent B* to do what *Agent A* did last.

d) These payoffs prevent *Agent A* from skipping to an action corresponding to a later time step.

e) These payoffs ensure that it is still attractive for *Agent B* to copy *Agent A*'s last move.

f) These payoffs are chosen carefully so that it *Agent B* doesn't move to a trap action after *Agent A* takes just a single action corresponding to a given node, but if it ever takes two such actions, then *Agent B* will be lured into the trap.

g) The payoffs for other trap actions are 0.

h) The payoff for selecting done only comes at m^*.

i) The payoff that induces *Agent B* to take its initialize action on the first step.

j) A payoff that prevents *Agent A* from taking an action corresponding to the initial node ever again (lest *Agent B* take the trap action).

k) This payoff prevents *Agent B* from taking the done action until all memory of *Agent A* taking the start action is past, i.e. after at least n=*mem* steps.

l) These payoffs play no special role.

m) These payoffs are for taking the last step on the Hamiltonian path (reaching the destination node).

n) These payoffs ensure that if *Agent A* takes the done action before step n, then the cost is already higher than the target of $n * (n^4 - 1)$.

Therefore, if we could find the optimal sequence through any matrix in polynomial time, then we could use this ability to also solve the Hamiltonian path problem. That is, finding S^* when $mem > 1$ is NP-hard. □

Appendix B. Proof of Theorem 3.1

Theorem Appendix B.1. *It is never optimal for the teacher to pull Arm$_2$.*

Proof. By induction on the number of rounds left, r.
Base case: $r = 1$. If the teacher starts by pulling Arm$_2$, the best expected value the team can achieve is $\mu_2 + \mu_1$. Meanwhile, if it starts with Arm$_*$, the worst the team expects is $\mu_* + \mu_2$. This expectation is higher since $\mu* > \mu_1$.
Inductive step: Assume that the teacher should never pull Arm$_2$ with $r-1$ rounds left. Let π^* be the optimal teacher action policy that maps the states of the arms (their μ_i, n_i, and $\bar{x}_i$) and the number of rounds left to the optimal action: the policy that leads to the highest long-term expected value. Consider the sequence, S, that begins with Arm$_2$ and subsequently results from the teacher following π^*. To show: there exists a teacher action policy π' starting with Arm$_*$ (or Arm$_1$) that leads to a sequence T with expected value greater than that of S. That is, the initial pull of Arm$_2$ in S does not follow π^*.

In order to define such a policy π', we define $S_1(n)$ and $S_2(n)$ as the number of pulls of Arm$_1$ and Arm$_2$ respectively after n total steps of S. As shorthand, we denote $S(n) = (S_1(n), S_2(n))$.

Similarly, define the number of pulls of Arm$_1$ and Arm$_2$ after n steps of T (e.g. when using π') as $T(n) = (T_1(n), T_2(n))$.

Next, define the relation $>$ such that $T(n) > S(m)$ iff $T_1(n) \geq S_1(m)$ and $T_2(n) \geq S_2(m)$ where at least one of the inequalities is strict. That is $T(n) > S(m)$ if at least one of the arms has pulled more times after n steps in T than after m steps in S, and neither arm has been pulled fewer times.

Finally, we define the concept of the teacher *simulating* sequence S based on the knowledge of what values would have resulted from each of the actions, starting with the teacher's pull of Arm$_2$ at step 1.[12] It can only do that as long as it has already seen the necessary values — otherwise it does not know what the state of the sample averages would be when it is the learner's turn to act. After n steps of the sequence T, let the number of steps that it can

[12]Such simulation relies on an assumption that the payoffs from an arm are queued up and will come out the same no matter when the arm is pulled: they are not a function of the times at which the arm is pulled, or the payoffs from any other arms. However, our argument still holds if the payoffs are time-dependent and/or dependent on other arms as long as the teacher has no knowledge of the nature of this dependency.

simulate in the S sequence be $\mathrm{Sim}(n)$. Specifically, $\mathrm{Sim}(n)$ is the largest value m such that $T(n) \geq S(m)$.

By way of illustration, let the values that will be obtained from the first pulls of Arm_2 be $u_0, u_1, u_2, \ldots$ and let those that will be obtained from the first pulls of Arm_1 be $v_0, v_1, v_2, \ldots$. Consider the following possible beginning of sequence S where pulls of Arm_* are marked with a^*, n is the step number, the teacher's actions are in the row marked "T" and the learner's actions are in the row marked "L" (note that by the induction hypothesis, the teacher never pulls Arm_2 after the first step).

n:	1	2	3	4	5	6	7	8	9	10	...
Teacher:	u_0		v_1		a*		a*		v_4		...
Learner:		v_0		v_2		u_1		v_3		v_5	...

In this sequence, $S(0) = (0,0), S(1) = (0,1), S(2) = (1,1), S(3) = (2,1), S(4) = S(5) = (3,1)$, etc.

Meanwhile, suppose that the teacher's first action in sequence T is Arm_* and the learner's first action is Arm_1, leading to v_0. Then $T(0) = T(1) = (0,0)$ and $T(2) = T(3) = (1,0)$.

Until the learner sees a pull from Arm_2 in sequence T, it cannot simulate any steps of S: $\mathrm{Sim}(1) = \mathrm{Sim}(2) = \mathrm{Sim}(3) = 0$. If the teacher's second action in T is Arm_* and learner's 2nd action is Arm_2, then in the example sequence above, $\mathrm{Sim}(4) = 2$.

We are now ready to define the teacher's policy π' for generating T. Let n be the total number of actions taken so far. Then:

1. If $n = 0$, $T(n) > S(\mathrm{Sim}(n))$ or $\mathrm{Sim}(n)$ is odd, then select Arm_*;
2. Else ($T(n) = S(\mathrm{Sim}(n))$ and $\mathrm{Sim}(n)$ is even), select the next action of S (i.e. the action π would select if there were $r - \frac{\mathrm{Sim}(n)}{2}$ rounds left).

Note that by the definition of Sim, it is always the case that $T(n) \geq S(\mathrm{Sim}(n))$. Further, note that at the beginning we are in step 1 of the strategy: $T(2) = (1,0) > (0,0) = S(\mathrm{Sim}(2))$. It remains to show that the sequence T resulting from using this policy π' has an expected value greater than that of S. We prove this in two cases.

Case 1: There is a least n, call it n', such that $T(n) = S(\mathrm{Sim}(n))$ and $\mathrm{Sim}(n)$ is even.

Until that point, the teacher keeps pulling Arm_*. We can thus show that $\mathrm{Sim}(n') < n'$ as follows. After n' steps, there are exactly $\frac{n'}{2}$ u's and v's in the

T sequence ($T_1(n')+T_2(n') = \frac{n'}{2}$). But after n' steps, there are at *least* $\frac{n'}{2}+1$ u's and v's in the S sequence ($S_1(n')+S_2(n') \geq \frac{n'}{2}+1$) because the first value is a u and all the learner's actions are u's or v's. Thus the simulation of S always lags behind T in terms of number of steps simulated: $\mathrm{Sim}(n') < n'$.

Note that if it is ever the case that $T(n) = S(\mathrm{Sim}(n))$ and $\mathrm{Sim}(n)$ is odd (it is the learner's turn to act in S), then the teacher will pull Arm_* once more after which the learner will do what it would have done in sequence S after $\mathrm{Sim}(n)$ steps. That will cause both $T(n)$ and $S(\mathrm{Sim}(n))$ to increment by the same amount, and $\mathrm{Sim}(n)$ to be even. Thus in the subsequent round, the teacher will switch to step 2 of its strategy.

Once the teacher has switched to step 2 of its strategy, then it will continue using that step: sequence T will follow S exactly for its remaining $2r - n'$ steps. To see that, observe that in each round, $T(n)$ and $S(n)$ will increment by the same amount, and $\mathrm{Sim}(n)$ will increment by exactly 2, thus remaining even.

Now compare the sequences T and S. Up until the point of step n' in T and $\mathrm{Sim}(n')$ in S, the only difference between the sequences is that there are $n' - \mathrm{Sim}(n')$ extra pulls of Arm_* in T. There then follow $2r - n'$ steps in the two sequences that are identical. The final $n' - \mathrm{Sim}(n')$ steps in S include at least one pull of Arm_1 or Arm_2 (the learner's first action). Thus the expected value of $T - S$ (the difference between the sum of their expected values) is at least $\mu_* - \mu_1 > 0$.

Case 2: It is never the case that $T(n) = S(\mathrm{Sim}(n))$ and $\mathrm{Sim}(n)$ is even. Then the teacher continues playing Arm_* throughout the T sequence (r times).

First, by the same argument as above, since the teacher always pulls Arm_*, it is always the case that $\mathrm{Sim}(n') < n'$.

Next, we argue that $T_2(2r) = S_2(\mathrm{Sim}(2r))$. That is, after $\mathrm{Sim}(2r)$ steps, the next step in S is a pull of Arm_2 (because $\bar{x}_2 > \bar{x}_1$). Otherwise, S could be simulated another step further by consuming another v value from T. We show this by induction on the number of steps in the T sequence i, showing that it is always the case that $T_2(i) = S_2(\mathrm{Sim}(i))$.

This equation holds at the beginning (e.g. when $i = 2$): $T(2) = (1, 0)$, $S(\mathrm{Sim}(2)) = (0, 0)$, so $T_2(2) = S_2(\mathrm{Sim}(2)) = 0$.

Now assume $T_2(i-1) = S_2(\mathrm{Sim}(i-1))$. There are three possibilities for the next action in T. If it is a pull of Arm_* or Arm_1, then $T_2(i) = T_2(i-1)$ and $\mathrm{Sim}(i) = \mathrm{Sim}(i-1) \Longrightarrow S_2(\mathrm{Sim}(i)) = S_2(\mathrm{Sim}(i-1))$, so the condition still holds. If it is a pull of Arm_2, then $T_2(i) = T_2(i-1)+1$ and $S_2(\text{Sim}(i)) = S_2(\text{Sim}(i-1)) + 1$ because the new u value can be used to continue the simulation of S by at least one step, and there are no additional u's in T to increase $S_2(\text{Sim}(i))$ any further. Therefore $T_2(i) = S_2(\text{Sim}(i))$.

Note that in general, $S_1(\text{Sim}(i))$ could be much greater than $S_1(\text{Sim}(i-1))$: there could be several v values from T that are then able to be used for simulating S. But if all of the available v's from T are used, we get that $T(i) = S(\text{Sim}(i))$, which violates the Case 2 assumption and puts us into Case 1 above (or will put us there one round later if $\text{Sim}(i)$ is odd).

Thus we have shown that after all $2r$ steps of T, the next action in the simulated version of S (step $\text{Sim}(2r) + 1$) must be Arm_2.

Finally, we compare the expected values of T and S. As above, there are several values in common between the two sequences, namely exactly the u's and v's from T that were used to simulate the first $\text{Sim}(2r)$ steps of S (as well as possibly some pulls of Arm_*). Let the sum of these u and v values be called COMMON.

Now consider the values of T and of S that are not in common: those values from T that were not used to simulate S, and those values in S that come after the simulation ended (after step $\text{Sim}(2r)$), plus all of the pulls of Arm_*. All of these "uncommon" values in T are from Arm_* and Arm_1. In fact, exactly r of the values are from Arm_* and exactly $T_1(2r) - S_1(\text{Sim}(2r))$ of them are from Arm_1. The uncommon values from S include at most $r-1$ from Arm_* (because the first teacher action was Arm_2), and at least one from Arm_2 (step $\text{Sim}(2r) + 1$).

Thus the expected values of the two sequences satisfy the following inequalities.

$\text{EV}(T) \geq r * \mu_* + [T_1(2r) - S_1(\text{Sim}(2r))] * \mu_1 +$ COMMON
$\text{EV}(S) \leq (r-1) * \mu_* + [T_1(2r) - T_1(\text{Sim}(2r))] * \mu_1 + \mu_2 +$ COMMON
Thus $\text{EV}(T)$-$\text{EV}(S) \geq \mu_* - \mu_2 > 0$.

Therefore in both cases, the expected value of sequence T exceeds that of sequence S. Since S is the best the teacher can do if it starts with Arm_2, and T is a lower bound on how well it can do otherwise, the teacher should never pull Arm_2. $\square$

Acknowledgements

Thanks to Michael Littman and Jeremy Stober for helpful comments pertaining to Section 2. Thanks to Yonatan Aumann, Vincent Conitzer, Reshef

Meir, Daniel Stronger, and Leonid Trainer for helpful comments pertaining to Section 3. Thanks also to the UT Austin Learning Agents Research Group (LARG) for useful comments and suggestions. This work was partially supported by grants from NSF (IIS-0917122, IIS-0705587), DARPA (FA8650-08-C-7812), ONR (N00014-09-1-0658), FHWA (DTFH61-07-H-00030), Army Research Lab (W911NF-08-1-0144), ISF (1357/07, 898/05), Israel Ministry of Science and Technology (3-6797), and the Fulbright and Guggenheim Foundations.

Bibliography

References

[1] P. Stone, G. A. Kaminka, S. Kraus, J. S. Rosenschein, Ad hoc autonomous agent teams: Collaboration without pre-coordination, in: Proceedings of the Twenty-Fourth Conference on Artificial Intelligence, 2010.

[2] P. Stone, G. A. Kaminka, J. S. Rosenschein, Leading a best-response teammate in an ad hoc team, in: E. David, E. Gerding, D. Sarne, O. Shehory (Eds.), Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets, 2010, pp. 132–146.

[3] H. Robbins, Some aspects of the sequential design of experiments, Bulletin of the American Mathematical Society 58 (5) (1952) 527–535.

[4] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[5] M. Tambe, Towards flexible teamwork, Journal of Artificial Intelligence Research 7 (1997) 81–124.

[6] G. A. Kaminka, I. Frenkel, Integration of coordination mechanisms in the bite multi-robot architecture, in: IEEE International Conference on Robotics and Automation (ICRA'07), 2007.

[7] B. J. Grosz, S. Kraus, Collaborative plans for complex group actions, Artificial Intelligence 86 (1996) 269–358.

[8] P. Stone, M. Veloso, Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork, Artificial Intelligence 110 (2) (1999) 241–273.

[9] J. Just, M. Cornwell, M. Huhns, Agents for establishing ad hoc cross-organizational teams, in: IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004, pp. 526–30.

[10] R. Kildare, Ad-hoc online teams as complex systems: agents that cater for team interaction rules, in: Proceedings of the 7th Asia-Pacific Conference on Complex Systems, 2004.

[11] J. A. Giampapa, K. Sycara, G. Sukthankar, Toward identifying process models in ad hoc and distributed teams, in: K. V. Hindriks, W.-P. Brinkman (Eds.), Proceedings of the First International Working Conference on Human Factors and Computational Models in Negotiation (HuCom 2008), Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands, 2008, pp. 55–62.

[12] H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, M. Tambe, Electric elves: Applying agent technology to support human organizations, in: International Conference of Innovative Application of Artificial Intelligence, 2001.

[13] K. Sycara, K. Decker, A. Pannu, M. Williamson, D. Zeng., Distributed intelligent agents, IEEE Expert 11 (6).

[14] E. Jones, B. Browning, M. B. Dias, B. Argall, M. M. Veloso, A. T. Stentz, Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks, in: International Conference on Robotics and Automation, 2006, pp. 570 – 575.

[15] M. Bowling, P. McCracken, Coordination and adaptation in impromptu teams, in: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI), 2005, pp. 53–58.

[16] R. I. Brafman, M. Tennenholtz, On partially controlled multi-agent systems, Journal of Artificial Intelligence Research 4 (1996) 477–507.

[17] F. Wu, S. Zilberstein, X. Chen, Online planning for ad hoc autonomous agent teams, in: Proceedings of the Twenty-Second International Joint

Conference on Artificial Intelligence, Barcelona, Spain, 2011.
URL `http://rbr.cs.umass.edu/shlomo/papers/WZCijcai11.html`

[18] S. Liemhetcharat, M. Veloso, Modeling and learning synergy for team formation with heterogeneous agents, in: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012), 2012.

[19] S. Zilles, S. Lange, R. Holte, M. Zinkevich, Models of cooperative teaching and learning, Journal of Machine Learning Research 12 (2011) 349–384.

[20] H. Zhang, Y. Chen, D. Parkes, A general approach to environment design with one agent, in: International Joint Conference on Artificial Intelligence, 2009.

[21] K. Genter, N. Agmon, P. Stone, Role-based ad hoc teamwork, in: Proceedings of the Plan, Activity, and Intent Recognition Workshop at the Twenty-Fifth Conference on Artificial Intelligence (PAIR-11), 2011.

[22] K. Genter, N. Agmon, P. Stone, Ad hoc teamwork for leading a flock, in: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), 2013.

[23] S. Barrett, P. Stone, S. Kraus, Empirical evaluation of ad hoc teamwork in the pursuit domain, in: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), 2011.

[24] S. Barrett, P. Stone, S. Kraus, A. Rosenfeld, Learning teammate models for ad hoc teamwork, in: AAMAS Adaptive Learning Agents (ALA) Workshop, 2012.

[25] K. Leyton-Brown, Y. Shoham, Essentials of Game Theory: A Concise, Multidisciplinary Introduction, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2008.

[26] N. Nisan, T. Roughgarden, E. Tardos, V. V. Vazirani (Eds.), Algorithmic Game Theory, Cambridge University Press, 2007.

[27] R. Powers, Y. Shoham, Learning against opponents with bounded memory, in: IJCAI'05, 2005, pp. 817–822.

[28] E. Jürgen, Bayesian learning in repeated normal form games, Games and Economic Behavior 11 (2) (1995) 254–278.

[29] V. Conitzer, T. Sandholm, Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents, in: Proceedings of the 20th International Conference on Machine Learning, 2003, pp. 83–90.

[30] H. P. Young, The possible and the impossible in multi-agent learning, Artificial Intelligence 171 (7) (2007) 429–433.

[31] M. L. Littman, Friend-or-foe Q-Learning in general-sum games, in: Proceedings of the Eighteenth International Conference on Machine Learning, 2001, pp. 322–28.

[32] D. Chakraborty, P. Stone, Online multiagent learning against memory bounded adversaries, in: Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases, 2008, pp. 211–226.

[33] Y. Shoham, R. Powers, T. Grenager, Multi-agent reinforcement learning: a critical survey, in: AAAI Fall Symposium on Artificial Multi-Agent Learning, 2004.

[34] S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium, Econometrica 68 (5) (2000) 1127–1150.

[35] A. Neyman, D. Okada, Two-person repeated games with finite automata, International Journal of Game Theory 29 (2000) 309–325.

[36] R. Axelrod, The Evolution of Cooperation, Basic Books, New York, 1984.

[37] G. W. Brown, Iterative solutions of games by fictitious play, in: T. C. Koopmans (Ed.), Activity Analysis of Production and Allocation, Wiley, New York, 1951.

[38] H. P. Young, The evolution of conventions, Econometrica 61 (1993) 57–84.

[39] H. P. Young, Individual Strategy and Social Structure: An Evolutionary Theory of Institutions, Princeton University Press, Princeton, New Jersey, 1998.

[40] Y. Nyarko, A. Schotter, An experimental study of belief learning using elicited beliefs, Econometrica 70 (3) (2002) 971–1005.

[41] C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in: In Proceedings of the Fifteenth National Conference on Artificial Intelligence, AAAI Press, 1998, pp. 746–752.

[42] S. Carrbery, Techniques for plan recognition, User Modeling and User-Adapted Interaction 11 (2001) 31–48.

[43] C. L. Sidner, Plan parsing for intended response recognition in discourse, Computational Intelligence 1 (1).

[44] K. E. Lochbaum, An algorithm for plan recognition in collaborative discourse, in: ACL, 1991, pp. 33–38.

[45] K. E. Lochbaum, A collaborative planning model of intentional structure, Computational Linguistics 24 (4) (1998) 525–572.

[46] B. J. Grosz, C. L. Sidner, Plans for discourse, in: P. R. Cohen, J. Morgan, M. Pollack (Eds.), Intentions in Communication, MIT Press, Cambridge, MA, 1990, pp. 417–445.

[47] B. J. Grosz, S. Kraus, The evolution of SharedPlans, in: M. Wooldridge, A. Rao (Eds.), Foundations and Theories of Rational Agency, 1999, pp. 227–262.

[48] J. M. Vidal, E. H. Durfee, Recursive agent modeling using limited rationality, in: Proceedings of the First International Conference on Multi-Agent Systems, AAAI/MIT press, 1995, pp. 125–132.
URL `http://jmvidal.cse.sc.edu/papers/vidal95.pdf`

[49] P. J. Gmytrasiewicz, E. H. Durfee, Rational coordination in multi-agent environments, Journal of Autonomous Agents and Multi-Agent Systems 3 (4) (2000) 319–350.

[50] E. H. Durfee, Blissful ignorance: Knowing just enough to coordinate well, in: Proceedings of the First International Conference on Multi-Agent Systems, 1995, pp. 406–413.

[51] J. Han, M. Li, L. Guo, Soft control on collective behavior of a group of autonomous agents by a shill agent, Systems Science and Complexity 19 (1).

[52] L. ji Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Machine Learning 8 (3/4) (1992) 293–321.

[53] L.-J. Lin, Self-improving reactive agents: Case studies of reinforcement learning frameworks, in: From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior, 1991.

[54] C. G. Atkeson, A. W. M. Y, S. S. Z, Locally weighted learning for control, Artificial Intelligence Review 11 (1997) 75–113.

[55] D. Pomerleau, ALVINN: An autonomous land vehicle in a neural network, in: Advances in Neural Information Processing Systems 1, Morgan Kaufmann, 1989.

[56] D. Grollman, O. Jenkins, Dogged learning for robots, in: International Conference on Robotics and Automation (ICRA 2007), Rome, Italy, 2007, pp. 2483–2488.
URL `http://www.cs.brown.edu/~cjenkins/papers/dang_ICRA_2007%.pdf`

[57] L. Csató, M. Opper, Sparse online gaussian processes, Neural Computation.

[58] J. Schneider, W.-K. Wong, A. Moore, M. Riedmiller, Distributed value functions, in: In Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufmann, 1999, pp. 371–378.

[59] L. Peshkin, K. eung Kim, L. Kaelbling, N. Meuleau, L. P. Kaelbling, Learning to cooperate via policy search, in: In UAI, 2000, pp. 489–496.

[60] A. Schaerf, Y. Shoham, M. Tennenholtz, Adaptive load balancing: A study in multi-agent learning, Journal of Artificial Intelligence Research 2 (1995) 475–500.

[61] P. Stone, M. Veloso, Multiagent systems: A survey from a machine learning perspective, Autonomous Robots 8 (3) (2000) 345–383.

[62] L. Panait, S. Luke, Cooperative multi-agent learning: The state of the art, Autonomous Agents and Multi-Agent Systems 11 (2005) 387–434.

[63] E. Yang, D. Gu, Multi-robot systems with agent-based reinforcement learning: evolution, opportunities and challenges, International Journal of Modelling, Identification and Control 6 (4) (2009) 271–286.

[64] D. Bergemann, J. Valimaki, Bandit problems, Tech. rep., Cowles Foundation Discussion Paper (2006).

[65] P. Bolton, C. Harris, Strategic experimentation, Econometrica 67 (1999) 349–374.

[66] M. Cripps, G. Keller, S. Rady, Strategic experimentation with exponential bandits, ECONOMETRICA 73 (2005) 39–68.

[67] M. Aoyagi, Mutual observability and the convergence of actions in a multi-person two-armed bandit model, Journal of Economic Theory 82 (1998) 405–424.

[68] G. Keller, S. Rady, Strategic experimentation with poisson bandits, Tech. rep., Free University of Berlin, Humboldt University of Berlin, University of Bonn, University of Mannheim, University of Munich, discussion Papers 260 (2009).

[69] A. Kayay, When does it pay to get informed?, International Economic ReviewForthcoming.

[70] R. D. Kleinberg, Online decision problems, Ph.D. thesis, Department of Mathematics (2005).

[71] L. S. Shapley, A Value for n-person Games, Vol. 2, 1953, pp. 307–317.

[72] G. Chalkiadakis, E. Elkind, M. Wooldridge, Computational Aspects of Cooperative Game Theory, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2011.

[73] R. J. Aumann, Acceptable points in general cooperative n-person games, Contributions to the Theory of Games 4.

[74] Subjectivity and correlation in randomized strategies, Journal of Mathematical Economics 1 (1) (1974) 67–96.

[75] N. Agmon, P. Stone, Leading ad hoc agents in joint action settings with multiple teammates, in: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012), 2012.

[76] S. Barrett, P. Stone, Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards, in: Tenth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA), 2011.