1 Introduction

Much of human life occurs in contexts where people must coordinate their actions with those of others. From party planning to space exploration to grant-proposal evaluation, groups of people have accomplished great things by reasoning as a team and engaging in jointly intentional behavior [77]. Indeed, some have argued that, because other animals lack the capacity to work adaptively as a cohesive unit across many domains, team reasoning may be the hallmark of human sociality [83].

Artificial agents are becoming ubiquitious in everyday life. But, they are not yet capable of collaborating with us effectively. For this "partnership of our future"[1] to become a reality, artificial agents will need to possess human-like behavioral flexibility. Unfortunately for agent designers, there is no simple behavioral rule (or set of rules) that could be prescribed to cover all contexts, communities, and cultures. People perpetually learn new rules and alter their presumed roles,[2] and so must artificial agents.

Most attempts to build collaborative, artificial agents begin with a model (however flawed) of human behavior. One possibility is to assume we are *Homo economicus*—a term dating back to the 1800s that portrays individuals as being guided by self interest, and taking actions without regard to their impact on the collective. Alternatively, one might call us *Homo sociologicus*—a more recent term depicting individuals as pursuing goals imprinted on them by society.

The homo-sociologicus model was not intended as a complete description of human behavior, which is clearly self interested at times. But, psychologists and behavioral economists have also more or less rejected the homo-economicus model (e.g., [16, 42]), proposing countless homo-sociologicus-leaning models in its stead (e.g., [8, 10, 18, 28, 38, 41, 70]). Consistent with these findings, this proposal is founded on the assumption that **social preferences** hold the key to successful collaborations. Social preferences are in part **distributive**, meaning they evaluate how resources are allocated, e.g., efficiently or equitably; and they are in part **reciprocal**, meaning they capture the social or selfish intentions of other players.

We plan to model interactions among such social agents, both human and artificial, in the framework of game theory. Following Bacharach [5], not only do we assume that preferences are social rather than individual, we go one step further and assume that decision making can take the form of "What should *we* do?" rather than "What should *I* do?" This form of reasoning is called **team reasoning** [4], and we consider it a viable alternative to the usual notion of best-reply reasoning [22] in games.

We call optimal decision making, when agents can hold social preferences and employ team reasoning, **socially rational** behavior. Social rationality is a blend of the *Homo economicus* and *Homo sociologicus* concepts—agents optimize a **social utility function** that represents social preferences, which incorporate perceived (possibly inferred, possibly imagined) societal benefits.

In reality, when an agent arrives on the playing field, it does not know whether the other agents it faces err on the side of being social (team reasoners) or selfish (best-repliers). While evaluating other agents' behavior, it behooves a socially rational agent to decide upon a strategy for itself—social or selfish?—because there is no point in an individual agent adopting a joint plan, unilaterally.

We contend that the computational process by which collaborative behavior emerges strongly depends on social rationality, coupled with learning via repeated interaction. Specifically, the computational process is as follows: initialize agents with an appropriate representation of social preferences using a social utility function tailored to the environment and the actors;[3] 1. agents decide whether selfish or social behavior is appropriate; 2. agents plan accordingly, using best-reply or team reasoning; 3. agents act, simultaneously or sequentially, as dictated by their environment; 4. agents update their estimates of their social utility functions. Steps 1 through 4 then repeat.

When agents use team reasoning to make a plan that optimizes a social utility function, they choose actions that jointly optimize the behavior of the collective. When agents update their estimate of the social utility function, they do so using the observed history of joint actions. If this process stabilizes, the social utility function represents the preferences of the community as a whole, and the equilibrium implements a joint plan of action that optimizes that social utility function. Indeed, in our computational model, emergent collaborative behavior and social preferences are expected to reinforce one another.

[1] These were the words of Satya Nadella, CEO of Microsoft Corporation, on June 28, 2016.

[2] As Shakespeare said, "one man in his time plays many parts."

[3] This utility function is described in terms of features of the environment and the actors; as such, it is applicable to other environments and actors with similar features.

Classical game theory assumes that utility functions are exogeneously determined—agents come to the table with fixed extrinsically-set utilities. However, there is broad agreement in the judgment and decision-making community that preferences are *constructed*, not elicited [86], in response to available choices. Likewise, we do not assume that social preferences exist extrinsically. In our conception of social rationality, agents *learn* social preferences through repeated play of the game.

A social utility function is an immediate generalization of the usual notion of an individual's utility function, but it ascribes value to the outcome as it relates to all actors in the environment. As such, it can be used to capture important elements of an artificial agent's decision making, including (1) how it values the utility of others [52], (2) how it values the utility of intermediate goals [58], and (3) how it values sanctions against others who exhibit anti-social behavior [57].

We further assume that social utilities can be broken down into two components: one objective, which is a direct function of the rules of the game; and the other, subjective, which can capture, among other things, notions of distributivity and reciprocity. Often, the rules of a game alone give rise to multiple (objective) equilibria, and agents face a difficult equilibrium-selection problem (e.g., [76]) in the underlying game. Whereas others have proposed learning [30] and evolution [43] in games as a potential solution to this challenging problem, we contend that more is needed—specifically: socially rational behavior, whereby agents optimize a *learned* social utility function that reflects (constructed) social preferences.

Under the assumption that humans are, perhaps boundedly but nonetheless ideally, socially rational creatures, we propose to design and build socially rational artificial agents that learn via repeated interactions, with the aim being for such agents to collaborate effectively with humans.

In this proposal, we set out to demonstrate our potential for success by showing that our computational model is viable. To this end, we present a preliminary set of results showing how reinforcement-learning agents, interacting repeatedly in a grid-game environment, can represent social preferences using a social utility function so that eventually their emergent collaborative behavior and their discovered social preferences reinforce one another. We then go one step further and show that our approach can also learn (in an off-line, batch fashion) from human–human trace data so that the collaborative behavior that is learned mimics that of the trace data. While these two experiments (on machine–machine interactive, and human–human batch data) are indeed promising, it remains to extend these techniques to human–machine interactions, ultimately in experimental worlds with embodied robots.

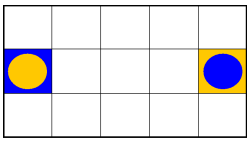
The ultimate goal in developing artificial agents that act intelligently in multi-agent scenarios is to apply them to real-world problems. To a limited extent, this goal has already been achieved: artificial agents trade stocks and bid in online ad auctions. In these two market environments, rationality may well be an appropriate model of behavior for the participating agents. However, an artificial agent that plans its collaboration with humans assuming their behavior is selfish, and dictated by extrinsic utility functions, is unlikely to be successful [31]. Instead of individual rationality as the underlying principle guiding human behavior, this proposal assumes that humans are socially rational beings whose preferences and behaviors emerge and reinforce one another through repeated interactions. For artificial agents to successfully collaborate with humans, they should be thus, too!

2 Grid Games

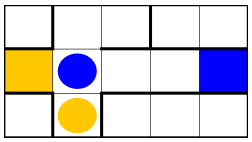
A grid game is a game played by two agents on a grid, in which each agent has a goal. See, for example, Figure 1a, which is a 3×5 grid in which the two agents' initial positions are one another's goals: Orange begins in position (1,2), Blue's goal; and Blue begins in position (5,2), Orange's goal. We refer to grid positions using x-y coordinates, with $(1, 1)$ as the bottom left position.

One grid game proceeds in rounds, each round consisting of multiple turns. On each turn, the agents choose one of five actions (N, S, E, W, or wait), which are then executed simultaneously. In the most basic setup, agents transition deterministically, and there is no tie-breaking when two agents collide.[4] Instead, if their chosen actions would result in a collision with one another, neither agent moves. A round ends when either (or both) players move into their goal, or when a maximum number of turns has been taken.

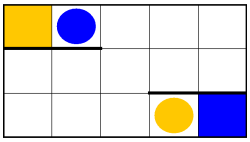
[4]It is a simple matter to vary these rules within our infrastructure, as future experimental design might dictate.



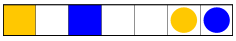
(a) Hallway



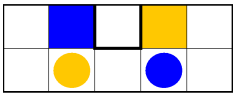
(b) Intersection



(c) Door



(d) Long Hall



(e) No Compromise

Figure 1: Five example grid games.

We present five two-agent grid games, specifically designed to vary the level of coordination required. The games also vary in whether it is possible to adopt a strategy that allows agents to defend against uncooperative partners—this is a property that we found to significantly impact the behavior of reinforcement-learning algorithms. In these games, one approach is for an agent to cooperate blindly with its opponent by simply moving out of the opponent's way, and hoping the opponent then waits for the agent to catch up. However, such strategies can be exploited by uncooperative ones that proceed directly to the goal as soon as their path is unobstructed.

To distinguish "unsafe" from "safe" cooperation, we devised a new categorization for strategies in our grid games. We call strategies that allow for cooperation, while at the same time maintain a defensive position in the event that the other agent is uncooperative, **cooperative defensive** strategies. More formally, an agent's strategy is **cooperative** (C) if it is one that allows both it and its opponent to reach their goals, while an agent's strategy is **defensive** (D) if its opponent does not have a strategy that allows it to reach its goal strictly first. A cooperative defensive (CD) strategy is both cooperative and defensive. We next apply these concepts to our games to illustrate the kinds of interactions we plan to study.

Our first example, Hallway, is depicted in Figure 1a. This game is one in which the agents can choose to coordinate, for example, if both agents agree upon a joint strategy where one agent moves along the top row and the other along the bottom, without interfering with one another. But, an agent could choose to "defect" from this joint strategy by proceeding straight to its goal. There are CD strategies, however, which defend against such non-cooperative behavior. For example, if Orange moves south initially to $(1, 3)$ and Blue moves west to $(4, 2)$, Orange might choose to return and remain on its goal until Blue retreats to $(4, 3)$ or $(4, 1)$, at which point the players are equidistant from their goals, and both can reach them safely. This joint strategy is an equilibrium comprised of CD strategies, since Orange and Blue both remain in positions where they have the ability to block their opponents until they both have unobstructed equidistant paths to their respective goals.

The grid in Figure 1b (Intersection) requires Blue to defend against the possibility of Orange behaving uncooperatively, which it can achieve by squatting on the orange goal until the agents are equidistant from their goals. Therefore, like Hallway, this game has an equilibrium comprised of CD strategies for both players. However, this equilibrium is not the shortest path and purely cooperative agents in this game could adopt a more efficient joint strategy.

Figure 1c (Door) is a grid that requires coordination to navigate through the narrow center space at $(3, 2)$. Several joint CD strategies exist, but they must be asymmetric because only one agent can be in the center cell at a time. Similarly, in the grid in Figure 1d (Long hall), Blue begins one step closer to its goal than Orange does. However, Orange can squat on the blue goal until Blue chooses to cooperate by taking one step back creating a joint CD strategy. In contrast, our last grid, shown in Figure 1e (No compromise), does not support joint CD strategies. Therefore, a trust spanning multiple rounds is required for the agents to effectively cooperate in this game.

Taking these five grid games as an initial test bed, we performed two pilot studies: the first involved simulations of artificial agents playing against one another; the second pitted humans against other humans on Mechanical Turk [2]. We summarize the results of these preliminary studies next.

Simulation Experiments We carried out a set of simulation experiments with Q-learning [87] in the grid games presented. As two Q-learning algorithms are not guaranteed to converge in self play, we arbitrarily stopped the learning after 30,000 rounds, and checked the strategies learned. In spite of Q-learning not explicitly seeking outcomes with high social welfare, it very reliably identified collaborative strategies in which both agents reached their goals.

In a second set of Q-learning experiments, we examined the impact of endowing agents with social preferences. That is, their utilities no longer depended solely on their own successes and failures but on those of other agents in the environment as well.

To make this idea more precise, we draw a distinction between **objective** and **subjective** utilities. Objective utility is the usual utility signal provided to the agent from the environment. Standard reinforcement-learning agents, such as Q-learners, seek to optimize their own objective utility signal—we call this preference the **selfish** preference because these agents are only concerned with their own outcomes.

In some environments, however, it useful to distinguish this objective utility from subjective utility—the perceived quantity the agent is optimizing. Previous work has shown that optimizing subjective utility can sometimes lead agents to be more effective in their acquisition of objective utility [80]. (Indeed, we reach this same conclusion in our experiments.)

We evaluate the result of agents maximizing utilities defined by a **fair** preference, specifically the objective utility of the agent minus 25% of the difference between its own and the opponent's objective utilities: $r_s = r_a - 0.25\,|r_a - r_o|$, where r_s is the agent's subjective utility, r_a is the agent's objective utility, and r_o is the opponent's objective utility.[5] With this social utility function, the agent prefers making it to its goal as opposed to not, but it additionally prefers that the opponent only get to its goal if the agent itself does as well. To say it another way, a fair agent wants its opponent to win with it or lose with it.

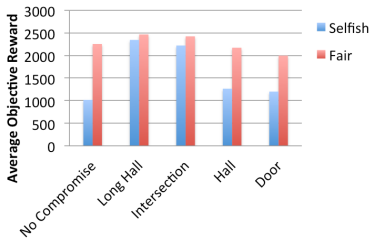


Figure 2: Average score of Q-learners in self play

Figure 2 shows the result of the selfish and fair agents playing against others of the same type in each of our test grid games. We find that fair agents obtain higher total (objective) utility than others. We also analyzed the types of strategies learned by fair and selfish Q-learners after multiple simulations of various configurations. Interestingly, we found that Q-learners with fair preferences tend to find CD strategies more often, especially when paired with selfish agents as these agents must be treated with caution.

In summary, Q-learners naturally learn to cooperate in the grid games studied, discovering equilibria comprised of CD strategies when they exist. Moreover, cooperation is induced by endowing the Q-learners with social utility functions.

Turk Experiments We conducted analogous behavioral experiments among humans playing grid games on Mechanical Turk. The results are consistent with our simulations of Q-learners playing grid games. In both cases, more collaboration was achieved when the treatment incorporated social preferences.

While the subjective utility functions of artificial agents are within our control, so that we can perhaps lead artificial agents towards collaborative behavior, the subjective utility functions of humans are not. Nonetheless, behavioral economists often infer approximations of utility functions from experimental data by assuming some underlying structural form, and then estimating the relevant parameters (e.g., [74]). Likewise, one of the primary intended uses of our Turk experimental framework is to generate trace data of humans playing grid games and learning collaborative behavior (such as trust, CD, etc.), so that we can then proceed to infer utility functions and relate them back to the behaviors they produce.

3 Computational Framework

Recall, from the introduction, our computational model: 0. agents represent social preferences using a social utility function; 1. agents decide whether selfish or social behavior is appropriate; 2. agents plan

[5] Other percentages would achieve the same result.

accordingly, using best-reply or team reasoning; 3. agents act, simultaneously or sequentially, as dictated by their environment; 4. agents update their estimates of their social utility functions.

While certain aspects of social preferences (e.g., fairness) are broadly applicable, Step 0 requires a representation that is tailored to the specific environment and actors within. Likewise, Step 3 involves a straightforward simulation in grid games, but is otherwise specific to the robots and their environment.

Only Steps 1, 2 and 4 are sufficiently generic to apply across environments. Step 2 is a **planning** step; as planning is a well-defined optimization problem, it can be accomplished using any off-the-shelf planner for stochastic sequential decision making environments [7, 9, 14, 20, 44, 47]. Steps 1 and 4, on the other hand, are not solved problems; indeed, in this proposal, we describe innovative approaches to these problems.

Step 4 can be thought of as a **learning from demonstration** problem [1], in which an agent learns how to behave in an environment by observing an expert. That is, an agent is presented with a data set consisting of multiple examples of behaviors (e.g., state–action trajectories through a Markov decision process), and is tasked with the objective of learning a policy capable of (closely) reproducing the data.

Two widely studied approaches to this problem include **imitation** and **intention** learning. The former is a supervised learning approach in which an agent learns a mapping from states to actions, with the expert's behavior serving as training data [68]. After training, a learner follows its learned policy directly.

Learning intentions [58], in contrast, is often framed as an **inverse reinforcement learning** (IRL) problem [3, 65]. Here, the agent's goal is to learn utilities that motivate the expert's behavior, after which it applies a planning algorithm to derive a policy that is consistent with those inferred utilities. A strength of intention learning is that even simple utility functions can capture complex behavior, leading to greater generalization capabilities (i.e., appropriate behavior in as-yet-unforeseen states) than the more direct approach taken by imitation learning.

We are proposing to use IRL in Step 4. Preliminary experiments, described herein, with one off-the-shelf IRL algorithm, demonstrate the potential of our computational model to facilitate collaborative behavior among artificial agents. But, to better support our goal of human–machine collaboration, we also propose a new approach to IRL in which demonstrations are not, by default, assumed to have been generated by an expert. With this additional power, our IRL approach can learn from all past interactions among agents, given a way of noting the successful and unsuccessful interactions. Indeed, in the computational process we put forth, learning is online, not off, so not all demonstrations should be interpreted as successful. On the other hand, as past interactions are shared context across agents, ideally they would *all* provide crucial guidance for identifying collaborative behavior, and that is what we set out to achieve.

Mathematical Models

As grid games are examples of stochastic games [29, 78], we use this game-theoretic framework to model agent interactions. Like econometricians [74], we assume a structural form for utility functions; and because we are interested in building socially rational artificial agents, we endow our agents with a (potentially) social utility function. We then develop a batch learning algorithm by which an artificial agent can, by learning from demonstrations, recover a social utility function that supports exactly one of many equilibria (assuming the demonstrations exhibit exactly one of many equilibria). We also develop an interactive learning algorithm by which two agents can simultaneously learn a social utility function, and converge to one among a plethora of equilibria, thereby endogenously solving the equilibrium-selection process.

Markov decision processes (MDPs) An MDP is a model of a single-agent decision-making problem defined by the tuple (S, A, T, R), where S is the set of states in the world; A is the set of actions that the agent can take; $T(s' \mid s, a)$ defines the transition dynamics: the probability the environment transitions to state $s' \in S$ after the agent takes action $a \in A$ in state $s \in S$; and $R(s, a, s')$ is the utility (or reward) function, which returns the reward the agent receives when environment transitions to state s' after the agent takes action a in state s.

The goal of planning in an MDP is to find a policy $\pi : S \to A$ (a mapping from states to actions) that maximizes the expected future discounted reward under that policy: $E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right]$, where $\gamma \in [0, 1]$ is a discount factor specifying how much immediate rewards are favored over distant rewards. To find an optimal policy, many algorithms compute the optimal state ($V^*(s)$) and state–action ($Q^*(s, a)$) value functions that specify the expected future discount reward under the optimal policy from each state,

and from taking an action in a state and then following the optimal policy, respectively. Given these functions, the optimal policy is derived by taking an action in each state with the maximum Q-value: $\pi(s) \in \arg\max_{a \in A} Q(s, a)$ [11].

Inverse reinforcement learning Although there are multiple IRL formalizations and approaches, they all take as input an MDP together with a **family** of utility functions R_Θ that is defined by some parameter space Θ, and a data set D of trajectories (where a trajectory is a finite sequence of state-action pairs: $\langle (s_1, a_1), ..., (s_n, a_n) \rangle$). The algorithms then output a specific parameterized utility function $R_\theta \in R_\Theta$, which induces a policy that is consistent with the input trajectories. Different IRL algorithms frame the objective function for policy consistency differently. One common approach is to treat the search problem as a probabilistic inference problem [3, 54, 72, 90].

For example, in the maximum-likelihood setting [3], the goal is to find a reward function parameterization that maximizes the likelihood of the data, $\theta \in \arg\max_\theta L(D \mid R_\theta) = \prod_{t \in D} \prod_i^{|t|} \pi_\theta(s_i, a_i)$, where $\pi_\theta(s, a)$ is a stochastic policy defining the probability of taking action a in state s when the parameterized reward function to be maximized is R_θ. Typically, the Boltzmann (softmax) stochastic policy over the Q-values is used. Popular methods for carrying out maximum-likelihood IRL (MLIRL) include gradient methods [75] and expectation maximization [71].

In Bayesian IRL, the goal is to compute a posterior distribution over reward function parameterizations: $\Pr(R_\theta \mid D) \propto \Pr(D \mid R_\theta) \Pr(R_\theta)$. Bayesian IRL algorithms typically rely on Markov chain Monte Carlo methods [69, 72]. Other approaches include maximum-entropy [90] and relative entropy IRL [12]. More recently, work has also commenced on nonparametric Bayesian IRL methods [19, 62].

IRL algorithms are typically computationally demanding because they require planning in their inner loops. Compatible with all of the above algorithms, **receding horizon IRL** (RHIRL) [58] addresses this limitation by replacing the usual infinite-horizon policy with a receding horizon controller (RHC) that only plans out to some finite horizon from any given state, thereby bounding computation time. An RHC tends to steer an IRL algorithm towards a **shaping reward function** [64], short-term rewards that guide the agent without it having to plan too far ahead. Consequently, RHIRL makes it possible to plan with a short horizon, thereby saving on computation time.

Our computational framework, and the generalized model of IRL we introduce in Section 3, are all agnostic about the particular approach to IRL taken. We plan to experiment with multiple variations.

Stochastic games The stochastic games formalism can be viewed as an extension of MDPs to the multi-agent case [51]. A **stochastic game** is defined by the tuple (I, S, A^I, T, R^I), where I is an index set of agents in the environment; S is the set of states of the environment; A^I is set of actions for each of the agents with A^i denoting the action set for agent $i \in I$; $T(s' \mid s, j)$ is the transition dynamics specifying the probability of the environment transitioning to state $s' \in S$ when the *joint action* $j \in \times_i A^i$ of all agents is taken in state $s \in S$; and R^I is a set of reward functions for each agent with $R^i(s, j, s')$ denoting the reward received by agent $i \in I$ when the environment transitions to state $s' \in S$ after the agents take joint action $j \in \times_i A^i$ in state $s \in S$.

The goal in a stochastic game is to find a joint strategy that satisfies some solution concept. Different solution concepts for stochastic games have been explored in the past including minimax, Nash, correlated, and CoCo equilibria [34, 39, 50, 91]. There are problems with these approaches, however. First, the resulting planners must solve for game-theoretic equilibria in an inner loop, a problem that, in the case of Nash equilibrium, for example, is notorious for its computational intractability [24]. Second, in the general case of non-constant-sum games, none of the planners that make reasonable assumptions about agent behavior yield unique joint plans, and none has solved the ensuing equilibrium-selection problem suitably.

Immediately generalizing from the case of MDPs, the goal of inverse reinforcement learning in stochastic games is to learn a set of reward functions for the stochastic game that describe an environment that would motivate the agents to behave in a way that is consistent with the observed behavior under some solution concept such as a Nash equilibrium [73]. This problem, however, is exceedingly difficult, because the planning that is necessary in the inner loop of an IRL algorithm is subject to the challenges identified by the equilibrium planners mentioned above. In this project, we plan to develop IRL algorithms that facilitate equilibrium selection in a stochastic game by leveraging shared experience.

Algorithm 1 Batch_Learning$((I, S, A^I, T, R^I), \gamma, D, \mathcal{T}, B_\Theta)$

Require: stochastic game (I, S, A^I, T, R^I); discount factor γ; multi-agent demonstrations D; team function $\mathcal{T}$; and parameterized bias function family B_Θ.

$A^M := \times_i A^I$ ▷ Joint action set

$R^M(s, a, s') := \mathcal{T}(R^I, s, a, s')$ ▷ Team function

$R^S_\Theta(s, a, s') := R^M(s, a, s') + B_\Theta(s, a, s')$ ▷ Social utility function family

$R^S_\theta := \text{IRL}((S, A^M, T, R^S_\Theta), \gamma, D)$

return R^S_θ ▷ Learned social utility function

Social Utility Functions In our preliminary studies, we assume players' individual (game) utilities are known, and our goal is to recover a **social utility function** that combines these known utilities in such a way as to capture social preferences expressed in the joint play of the agents (when one exists).

Like the players' reward functions, a social utility function operates on states, joint actions, and next states: $R^S : S \times_i A^i \times S$. In our initial model, we assume the social utility function can be written as a linear combination of a **team function**, which represents team goals, and a family of what we call **bias functions** (B_Θ) defined by some parameter space Θ. The team function takes as input a multi-agent utility function (R^I), and returns a single numeric "team" value for any state, joint action, next state triple. One example of such a function is total welfare (i.e., the sum of all agent utilities): $\mathcal{T}(R^I, s, j, s') = \sum_i R^i(s, j, s')$. The bias function family is similar in nature to a utility function family that would be input to a classic IRL algorithm, but operates on state, joint action, next state triples, thereby encoding a bias that motivates specific collaborative behavior in games. In addition to the parameters of the bias function, the social utility function may also include a parameter to trade-off the team utilities against the biases.

Learning Social Utility Functions

Given our representation of social preferences by a social utility function that is parameterized by a family of bias functions, our approach to learning these functions is to optimize those parameters to best incentivize observed behavior. We consider two learning frameworks. In **batch** learning, an agent learns offline from demonstrations. In **interactive** learning, agents learn the social preferences of the collective in the context of repeated interactions. At the core of both algorithms is team reasoning.

Team Reasoning In our model, the demonstrations agents learn from are trajectories of state, joint action pairs, specifying the behavior of all the agents in the environment. Likewise, their objective, as team reasoners, is to produce a plan for what all the agents in the game should do. To produce such a plan, the stochastic game must be transformed into an MDP. For the most part, this transformation is straightforward: the states are the same, the MDP action set is the space of joint actions, and the MDP transition dynamics still operate on joint actions (which is the action set in the MDP). A family of social utility functions is then defined as a linear combination of the input team function (which depends on the stochastic game's utility functions), and the family of bias functions.[6]

In addition to the human-behavioral motivations for building agents that employ team reasoning, team reasoning also makes the problem tractable, because IRL in MDPs is becoming increasingly easier, while IRL in stochastic games remains enormously challenging.

Batch Learning The batch-learning setting is similar to the standard IRL problem in that the algorithm is given a batch of demonstrations and a family of utility functions over which to search. In our case, however, the batch data consists of trajectories of states and joint actions in a stochastic game, and the utility function family is a social utility function family. To learn the parameterization of the social utility function family that best captures the behavior exhibited in the demonstrations, we convert the stochastic game into an MDP, and then use IRL to learn a parameterization of the utility function family in this MDP.

Pseudocode for the batch-learning algorithm is shown in Algorithm 1. The algorithm takes as input a stochastic game, a discount factor, a set of multi-agent demonstrations (which perhaps adheres to some

[6] A parameter controlling the linear combination of the team and bias function may be optimized as well.

Algorithm 2 Interactive_Learning($(I, S, A^I, T, R^I), k, \gamma, \mathcal{T}, B_\Theta$)

Require: stochastic game (I, S, A^I, T, R^I); agent index k; discount factor γ; team function $\mathcal{T}$; and parameterized bias function family B_Θ.
initialize policy π^k arbitrarily
$A^M := \times_i A^I$ ▷ Joint action set
$D := \{\}$
for each round of play **do**
play round by following policy π^k
append new trajectory of play to D
$R^S_\theta :=$ Batch_Learning($(I, S, A^I, T, R^I), \gamma, D, \mathcal{T}, B_\Theta$)
$\forall s \in S, \pi^k(s) := a^k \in A^k$ where $a \in \arg\max_a Q_{R^S_\theta}(s, a)$
end for

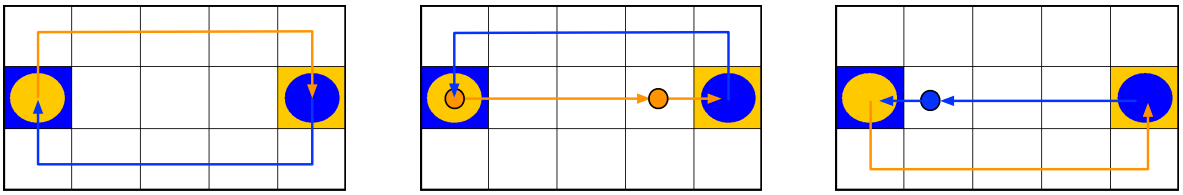
equilibrium: i.e., collaborative behavior), a team function, and a family of bias functions defined by some parameter space Θ. The first step of the batch-learning algorithm is to transform the stochastic game into an MDP, and the second to apply IRL to that MDP. After learning a social utility function, any single agent can then behave by selecting a joint action from the policy that results from it, and then selecting their individual action from the joint. That is, agent i takes action $j^i \in A^i$, where $j \in \arg\max_j Q_{R^S_\theta}(s, j)$.

Interactive Learning In the interactive learning setting, an agent plays with a set of unknown agents, with whom it can potentially establish new collaborations. Pseudocode for our interactive learning algorithm is shown in Algorithm 2. The algorithm takes as input the same arguments as the batch-learning algorithm, except it includes an index specifying which agent in the stochastic game the learner is, and it does not include the batch of demonstrations. The agent begins by initializing its policy arbitrarily.[7] The first round of interaction produces a trajectory of joint behavior that the learner can now learn from. It does so by running our batch-learning algorithm on the single trajectory of experience acquired thus far. It then proceeds to follow the learned policy in the next round. After the second round completes, the agent now has two trajectories on which it can run batch learning to update its social utility function and its behavior. This process then repeats for all rounds of interaction. Note that updating the policy after each round requires computing the Q-values under the newly learned social utility function. However, in practice, this computation can be performed lazily for each state the agent observes in the next interaction.

Preliminary Results Preliminary experiments suggest that both our batch-learning and interactive collaborative learning algorithms can produce reasonable results in grid games. For batch learning, we hard code examples of agents following different equilibrium strategies, and show that our algorithm is able to recover the appropriate equilibrium when it is trained on demonstrations of behavior following that equilibrium. We also tested the agents on similar grid games after learning to show that the learned behavior can generalize to other similar situations. For interactive learning, we show that agents can very quickly find an equilibrium, and that restarting learning from scratch can result in them finding a different one.

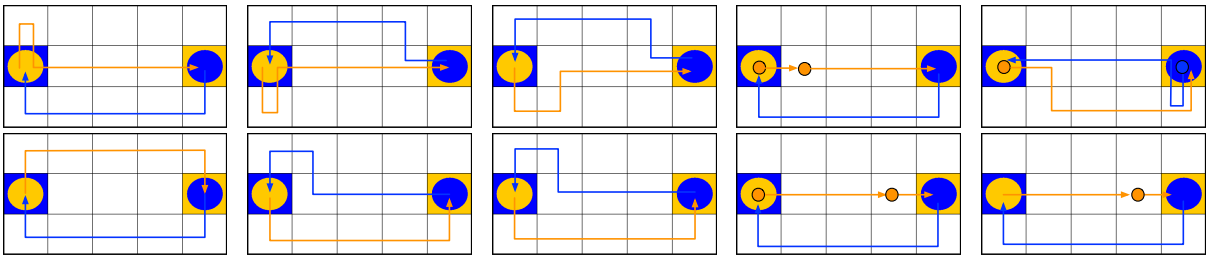
For both sets of experiments, we examine behavior in Hallway. The social utility function consists of a total welfare team function and a linear family of bias functions. The bias functions operate on 50 state, joint action binary features. The first 25 features are an indicator variable for selecting one of the 25 joint actions when the agents are in *conflict* and the latter 25 features are indicator functions for selecting one of the 25 joint actions when the agents are not in conflict. We define the agents to be in conflict when they are in the same row of the grid and each is closer to their opponent's goal than they are to their own. Additionally, when one of the agents is one step away from their goal, none of the features activate, thereby preventing premature termination on the joint task. Since these features directly encode social utilities for joint actions, rather than higher-level goals, when training we use RHIRL with a horizon of 1. We do not expect this set of features to be complete or capable of capturing all the possible behaviors that might emerge in a grid game. So, an important area in this project is to identify features that are both expressive and generalize well. Nevertheless, the features used here provide a proof of concept for our approach.

[7] In practice, the policy can be initialized to the agent's role in the joint policy that follows from the team function alone.



(a) (b) (c)

Figure 3: The three different equilibria the batch-learning algorithm trained on in Hallway. Colored arrows indicate the path of the player of the same color. A small circle indicates a wait action.



(a) (b) (c) (d) (e)

Figure 4: The learning results from each of the 5 interactive matches (a-e). The top image for a match is the first round of interaction of the agents. The bottom image for a match is the learned behavior. For matches b–e, the last move of an agent that reaches their goal sooner may be to wait for two steps, or to move north and/or south and back again; only one example is illustrated.

We tested our batch-learning algorithm on datasets in which agents followed three different equilibrium strategies. The first (Figure 3a) shows Orange going north and then traveling along the top row of the grid, while Blue goes south and then travels along the bottom row. The second (Figure 3b) shows Blue going north and then traveling along the top row while Orange waits, then goes east, and then waits again for Blue to catch up so they can enter their goals together. The third (Figure 3c) has Orange go south and then traveling along the bottow row, while Blue immediately goes west toward its goal and then waits two steps.

For each of these three equilibria, we generated five demonstrations, two of which described the behavior exactly, and the other three of which contained slight deviations (for example, moving back to the center row before reaching the end). We then separately trained our batch-learning algorithm on each set of demonstrations. In all cases, the learned social utility function motivated behavior that consistently replicated the data on which it was trained. Additionally, we planned using the learned social utility function in a larger 7×3 hallway grid game, and, in all cases, obtained the corresponding equilibrium behavior.

In our interactive learning experiment, we played two interactive learning agents against one another. The agents played five matches, each one consisting of five rounds. Each match began fresh, without any knowledge of previous matches, to see whether different equilibria could emerge. The social utility function was initialized to the total welfare joint policy. In each match, the agents very quickly (typically after the first round) converged to an equilibrium that persisted for the rest of the match. However, in each match, a different equilibrium was learned, showing that agents employing our algorithm quickly adapt to the particular shared experience they have with other agents.

Figure 4 shows the results of learning in each of the five matches (with the top image showing the behavior in the first round and the bottom image showing the learned behavior). While different equilibria emerged in different matches, once learned, that equilibrium was consistently employed in all subsequent rounds. Of particular interest are the results in the fifth match (Figure 4e). In the first round of this match, both players wait (a suboptimal joint action); Blue then moves in an odd looping motion. Nevertheless, the agents learn from the experience, ultimately finding collaborative behavior that works well for them both.

Generalized Inverse Reinforcement Learning

To complement the new computational model we put forth, the technology that we plan to develop as part of this proposal is a framework for IRL and accompanying algorithms that allows for both positive and negative (not necessarily binary) labels, and, moreover, that also allows for partial feedback, meaning subtrajectories (i.e., options [82]) can be ascribed such labels.

We call our framework **Generalized IRL** (GIRL). Whereas the input to standard IRL is a data set of trajectories and the output a utility function, the input to a GIRL problem is a data set of *labelled* trajectories. In its most basic formulation,[8] each label is either positive ($+1$) or negative (-1), indicating whether the associated trajectory is a good example of some desired behavior or a bad example of some desired behavior. The goal is to recover a utility function that motivates the desired behavior.

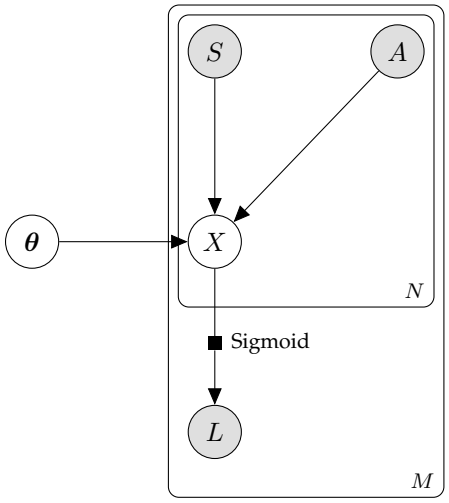


Figure 5: Generalized IRL

Others are also attempting to solve similar problems [15, 79], but our problem formulation, which is quite general, is also novel. The plate diagram that describes this formulation is shown in Figure 5. In this model, we interpret the label (L) associated with a trajectory of size N to be some random function of what the evaluator thought about each of the action selections (A) exhibited at each of the states (S) in the trajectory. However, these step-wise evaluations (X) are unobserved in the data.

For illustrative purposes, we adopt the probabilistic maximum likelihood problem formulation, in which we seek a utility function that maximizes the likelihood of the data, but Bayesian reasoning also applies. To find the utility function that maximizes the likelihood of the data, we first define a generative probability model for the system that is represented in Figure 5. We model the probability that an action is evaluated as good or not as proportional to its selection probability according a softmax policy computed for the utility function with parameters $\boldsymbol{\theta}$, specifically, $\Pr(x_i = +1 \mid s, a, \theta) = \pi(s, a \mid \boldsymbol{\theta})$ and $\Pr(x_i = -1 \mid s, a, \theta) = 1 - \pi(s, a \mid \boldsymbol{\theta})$. Here, $\pi(s, a \mid \boldsymbol{\theta})$ is the softmax policy over Q-values, assuming a utility function parameterized by $\boldsymbol{\theta}$.

For the probability distribution of L, given the sequence of N step-wise labels, we would like a distribution that has the property that as more step-wise labels are positive, the probability of a positive trajectory label increases (and vice versa). Although there are many possible distributions that satisfy this property, for concreteness, we choose the sigmoid function. That is, $\Pr(L = +1 \mid X_1, ..., X_n) = (1 + e^{-\phi \sum_i^N X_i})^{-1}$ and $\Pr(L = -1 \mid X_1, ..., X_n) = 1 - \Pr(L = +1 \mid X_1, ..., X_n)$. Here, ϕ is a parameter that controls how readily step-wise labels influence the trajectory labels. For example, when $\phi = 0$, trajectory labels are assigned uniformly at random, without any regard for of step-wise labels. As $\phi \to \infty$, the sigmoid converges to a step function in which a trajectory containing even one more positive step-wise label than negative will deterministically lead to a positive trajectory label (and vice versa).

EM Algorithm The problem with estimating the parameters ($\boldsymbol{\theta}$) of a utility function in this setting is that there is a latent variable vector X (or rather, some of the elements of the X vector are latent, and some may be observed), which makes it difficult to compute the likelihood of the model and maximize its parameters. The EM approach to solving this problem is to first choose values for $\boldsymbol{\theta}$; then choose a new $\boldsymbol{\theta}$ that maximizes the expected value of the log likelihood function, where the distribution in the expectation is the probability of the latent variables (the Xs in our case), given the observed data and previous choice of $\boldsymbol{\theta}$. The maximization can be performed using gradient ascent, and this process repeats until convergence.

To simplify the exposition of the EM algorithm, we introduce the notation $\boldsymbol{x}_k$ to indicate the values of the subset of observed (i.e., known) elements in an $\boldsymbol{x}$ vector, and $\boldsymbol{x}_u$ to represent a possible assignment to the subset of unobserved values. Using this notation, the EM algorithm, at a high level, is summarized in Algorithm 3. We have derived a dynamic programming algorithm that, coupled with importance sampling, yields a tractable version of EM for the generalized IRL problem.

[8] Labels need not be binary; they can be continuous valued.

Algorithm 3 EM Algorithm for GIRL

Require: initial $\boldsymbol{\theta}^0$, and data $\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{x}_k, l$
  for $t = 0$ to T **do**
    $\boldsymbol{\theta}_{t+1} \leftarrow \arg\max_{\boldsymbol{\theta}'} \mathbb{E}_{\boldsymbol{x}_u \sim \Pr(\boldsymbol{x}_u | l, \boldsymbol{x}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})} [\log \mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}' \mid l, \boldsymbol{x})]$
  end for

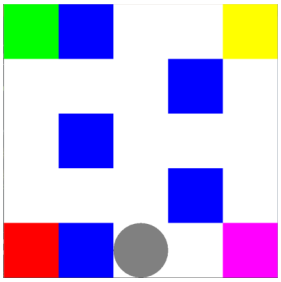


Figure 6: Puddle grid. Initial position.

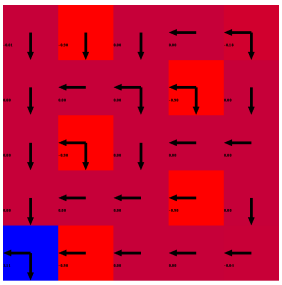


Figure 7: Two positive trajectories.

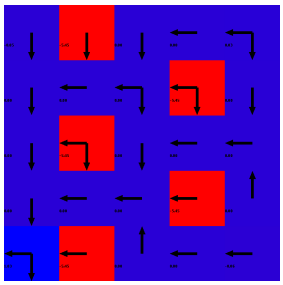


Figure 8: Positive & negative trajectories.

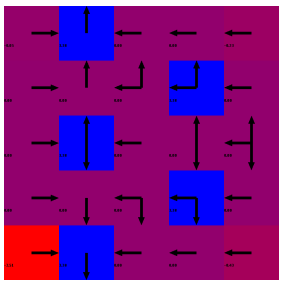


Figure 9: Negative & positive trajectories.

Preliminary Experiments We ran some (very) preliminary experiments to demonstrate the potential of our EM approach in the GIRL framework.[9] The agent is operating in a grid world (see Figure 6), with puddles of multiple colors: red, blue, green, yellow and pink. The agent (gray circle) can take north, south, east and west actions. In these experiments, we gave the agent two very short trajectories, and a corresponding pair of positive and negative labels. We show that varying these labels leads the agent to infer different utility functions, and correspondingly different behaviors.

In both trajectories, the agent begins in the middle cell of the bottom row of the grid (as shown). The first trajectory has the agent move two steps to the west, through the blue puddle, and terminating at the red puddle. The second trajectory has the agent move a single step west, terminating at the blue puddle. We compare three different labelings for these trajectories (all three labelings that include at least one positive label): two positive ($+1$ for each), positive & negative ($+1$ for first, -1 for second), and negative & positive (-1 for first, $+1$ for second). The learned policies, along with shading that reflects inferred utility values (blue is high; red is low; and shades of purple, in between) are shown in Figures 7, 8, and 9, respectively.

The two positive labels signal to the agent that both observed behaviors are good. Consequently, the agent infers that the red puddle is worth more than any other cell in the grid and seeks shortest paths for reaching it. Note that the agent does not learn to linger on the blue puddle, or direct its attention toward any other blue puddles in the grid—even though one trajectory ended there—because both trajectories have positive labels, and ultimately lead to the red puddle.

Given positive & negative labels, the agent again infers that reaching the red puddle is good and the blue puddle is not good. In contrast to the previous case, however, it infers that all other cells are also good. This example illustrates the power of our model to learn from negative examples. By asserting that the path terminating at the blue puddle is bad, the agent infers that the correct path for reaching the red puddle must be to go around the blue puddle.

The agent sees negative & positive labels as a signal that the red puddle is to be avoided and getting to the blue puddle is good. This time, the agent prefers the blue puddles to the red, and it finds a policy by which it can reach, and remain in, blue puddles.

In a fourth experiment (not shown), we labelled both trajectories -1. Given this information, the agent learns to avoid all puddles, and directs all its movements towards the white spaces on the grid, instead.

These experiments demonstrate an approach to GIRL, in which agents learn utility functions from both positive and negative examples. We plan to further hone this and related approaches, so that we can eventually bring them to bear in the context of interactive learning of social utility functions, thereby enabling artificial agents to learn from rich human evaluative feedback.

[9] Because the problem domain in these experiments was small (a 5×5 grid), we did not need importance sampling.

Selecting a Selfish or Social Stance

We expect the artificial agents we build to be able to interact with one another, and with humans, and infer social utilities that will guide the collective towards mutually beneficial behavior. In contrast to existing machine-learning approaches that achieve cooperative behavior by making strong assumptions about the algorithms other agents in the environment adopt (e.g., [21]), our approach to coordinatation relies only on a shared history among agents, at a purely behavioral (i.e., observational) level.

Nevertheless, we have for some time now been making a key assumption about the intentions of other agents, namely that they, too, have the goal of adopting joint goals. In our initial experiments among humans (only) in the Hallway game, we observed that some pairs readily cooperated, while others adopted a competitive or selfish stance toward the interaction. In the most interesting cases, one player began with a selfish stance, trying to block the other agent by getting to the goal first, while the other adopted a social stance, creating opportunities for both players to succeed. There are several important computational problems that arise when the two agents adopt conflicting stances. We briefly list these problems here, and a proposed approach to each:

1. *How can an agent detect that its stance differs from another's?* In preliminary work [46], we adopted a Bayesian perspective to classify the other agents in one's environment as selfish or social (competitive or cooperative, in the terminology of that earlier project) from observed trajectories. We will evaluate a selfish/social classifier on the data we collect from human–human interactions to see if its judgments match those of human observers. We will then deploy the classifier in human–machine interactions to see whether it can be used to behave socially without taking on too much risk of exploitation.

2. *If an agent adopts a social stance while another agent in its environment adopts a selfish stance, how can the social agent convince the selfish agent to change its behavior?* Our analysis of CD strategies (see Section 2) provides a partial answer—an agent using such a strategy encourages another, even if it is inherently selfish, to behave in a mutually beneficial way. In particular, a CD strategy allows for cooperation but prevents exploitation, by making the other player an offer it would be foolish to refuse. If a CD strategy does not exist, an agent could use a strategy that, across repeated games, provides the same benefits as CD strategies within games [63]. It is challenging for an agent to capture the attention of another to trigger behavior change, so we expect interesting problems to be revealed.

3. *When should an agent stop trying to pursuade others to adopt a social stance?* There is a cost to remaining social while other agents are behaving selfishly. The decision of when to conclude that another agent is unlikely to be convinced to change its behavior can be formulated as a reinforcement-learning problem in its own right [13]. We will investigate this problem, bringing to bear techniques from the study of partially observable Markov decision process [40], so that agents can reason effectively about the unobservable stances of others, as well as whether it can influence them, and if so, how.

Based on preliminary experiments with human–machine interactions, we conjecture that stance selection is an essential capability for socially rational agents. Consequently, we are proposing to integrate this decision into our computational framework (Step 1). We will evaluate our agents in a controlled environment, with and without this capability, to assess how effective it is in fostering collaborative interactions.

4 Plan, Deliverables, and Evaluation

Plan The three-year timeline for our proposed project is as follows:

- **Year 1**: Demonstrate that a socially rational agent can successfully learn collaborative behavior from batches of demonstrations (offline), and can generalizate across related games. Carry out simulation experiments on machine–machine pairings, varying both the algorithms and the environments. Complete the development of an experimental test bed that pairs humans with artificial agents. Evaluate algorithms that classify environments as selfish or social, and adopt the corresponding stance.

- **Year 2**: Establish that IRL algorithms within the GIRL framework can learn more accurately and more efficiently (i.e., from fewer data) than classic IRL algorithms, because of the availability of both

positive and negative feedback. Incorporate these algorithms into the Year 1 experimental test bed, and show collaborative behavior emerge in real-time during human–machine interactions. Develop the proposed "Social autonomous driving" course (see Broader Impacts).

- **Year 3**: Extend as necessary, and then evaluate our approach in real-world applications, specifically a "go fetch" scenario to be developed on top of a mobile-manipulation platform that is scheduled for deployment in the Brown Computer Science department in Spring 2017 (see Broader Impacts).

Deliverables The proposed project includes the following deliverables:

1. An *experimental platform* in which humans can interact with artificial agents, thereby providing a space for artificial agents in general, and our socially rational agents, specifically to demonstrate their capacity to represent, learn, and apply collaborative behaviors.
2. *Software infrastructure for socially rational algorithm development and experimentation* made available as a part of the Brown–UMBC Reinforcement Learning and Planning library.[10]
3. Data comprised of the behaviors produced by both humans and machines on a test bed of stochastic games, varying the degrees of complexity and uncertainty, as well evaluations of those behaviors, available as a *socially rational benchmark*.
4. *Syllabus for an undergraduate class* on "Social autonomous driving" in which students develop socially rational driving algorithms for robot cars.

In Year 2, we plan to organize a workshop (e.g., a AAAI symposium) on the machine learning of socially rational behavior. Such a gathering would be timely, as a number of subareas are grappling with related problems. We will include researchers in human–robot collaboration [33], self-driving cars [27], end-user programming of household devices [84], agent-based bidding [88], multi-agent reinforcement learning [81], computational social cognition [6], and others.

Evaluation We will evaluate how our socially rational agents fare in collaborative learning tasks using an experimental platform that simulates interactions between humans and artificial agents. Specifically, we will compare how often and how quickly humans are able to arrive at coordinated plans among themselves vs. what happens when socially rational agents are involved. We will also assess the quality of the plans that emerge under these varying conditions. Subjective evaluations will be used to assess whether humans find interacting with agents "natural."

For the batch-learning algorithm, we will develop new quantitative metrics to assess behavioral similarity so that trajectories produced by our batch learners can be quantitatively compared to trace data produced by human participants. Additionally, we plan to design a sort of "Turing test" for our problem domain, whereby we will have two humans interact with each other for a number of rounds and then, without notification, we will fork those interactions into two, where each human is now playing with a socially rational agent trained on the history of interactions between the two humans. At the end of all interactions, we will probe whether the participants noticed anything strange partway through the interactions.

We are particularly interested in designing flexible agents that generalize their behavior at least as well as humans do to new environments. An underlying goal, therefore, will be to engineer the underlying structure of social utility functions so that they generalize well across environments. We we will start by examining standard feature-selection methods used in existing reinforcement-learning and IRL research (e.g., linear models, neural networks, etc.) [26, 48, 49, 66], and build on them, as necessary. We will also explore available deep learning packages, as they are compatible with gradient-based methods to IRL, as they may enable us to scale to richer utility functions and environments.

[10]The open source Brown–UMBC Reinforcement Learning and Planning (BURLAP) library [56] is the software infrastructure that will be used for algorithm development in this project. Developed by James MacGlashan under the direction of co-PI Littman, the system provides a flexible and powerful implementation of a wide variety of reinforcement-learning algorithms and environments, and has been extensively used in both published research and education (including in a reinforcement-learning MOOC). The RHC-MLIRL algorithm used for the experiments in this proposal is available as part of the BURLAP distribution.

In year three, we will test our approach in a real-world robot–human collaboration domain. Even if (as we expect), our simulated worlds show that our agents collaborate well with people, the capabilities (or lack thereof) of robots may bring to the fore as-yet-unanticipated hurdles (e.g., robots might move slower than people expect). We will evaluate our approach in real-world scenarios by comparing a socially rational robot to a robot controlled by a human via teleoperation. We expect this investigation to raise novel research questions appropriate to real-world problems.

5 Broader Impacts

Recent trends in computer science and artificial intelligence are moving us toward increasing dependence on human–computer collaborative systems in which people and software make decisions that impact one another. Some fields, such as human–computer interaction (HCI), focus on systems in which a human being is primarily in control, and the computer's decisions are assessed in terms of the positive impact they have on the user [59]. An interesting recent example is Centaur chess [17] in which a human and machine team up to play on the same side in chess, both making decisions, but with the machine acting as an advisor and the human deciding which moves to actually make. Other fields, like crowd sourcing and human computation [85], combine human and machine expertise with the machine as the ultimate arbiter of behavior, and a group of humans acting as lower level computational components.

True collaboration, however, does not start with one participant being assigned to a leadership role. Instead, the various agents need to dynamically negotiate their roles and jockey for position, discovering when and how to trust each other to move forward. A concrete example is in the context of self-driving cars. Cars share the road with each other and must carefully choose when to be responsive to other vehicles and when to assert themselves to create a situation that benefits them. Doing so makes a significant difference in effectiveness [23]. A related problem arises when a self-driving wheelchair attempts to move through a group of pedestrians [45]. More generally, robots that interact with people in the physical world need to navigate the complex give-and-take of establishing mutually beneficial behaviors where possible.

Indeed, a wide variety of human–machine interaction problems would be positively impacted by the technology we are proposing. As such, the project fits well with Brown University's Humanity Centered Robotics Initiative (HCRI), of which co-PI Littman is co-director. Specifically, within HCRI, there are ongoing efforts to design robotic systems that interact with people and support independent living tasks (e.g., gerontechnological support for aging in place and robotic sous chefs) and our project will form an important foundation for this broader work.

The most direct application of our ideas would be in scenarios in which a robot is sharing a space with bystanders (human or robotic) who do not necessarily share its goals—a personal grocery-store assistant, a socially aware vacuum, an automated hospital linen cart, or a package delivery drone. To further develop our ideas in a physical environment like these, and moreover in embodied robots, we plan to build a "go fetch" application on top of a mobile-manipulation platform that is scheduled for deployment in the Brown Computer Science department in Spring 2017. In this application, a user will ask a mobile maipulator robot in the department to "go fetch object X from person Y" (for example, "borrow the HCI textbook from Prof. Huang" or "pick up the package that Genie has for me in the mail room"). To be successful in this capacity, the robot will need to be able to represent, learn, and apply collaborative behavior in (at least) three separate spheres: (1) navigating smoothly through the atrium and hallways, finding appropriate ways to avoid collisions, and adopting regularities that make it easy for people to coordinate their own movements with it; (2) riding in the elevator, which brings with it some related but distinct expectations for coordinating (moving in and out of the doors, standing inside, asking for help with the buttons, deciding what to do when obstructed, etc.); (3) understanding how to best signal a handoff of the object it is fetching, interrupting people without being too timid or too boorish. (Interest in robotics and machine learning is growing within the Brown CS department as elsewhere. Presently, our faculty is designing a new robotics introductory course sequence that includes classes on basic robotics algorithms, hardware design, and human-centered evaluation of robotics systems. Inspired by the ideas in this proposal, we plan to contribute a new intermediate robotics course for undergraduates called "Social autonomous driving." As a test platform, we will use MIT's Duckietown platform [67], which has already been installed at Brown. Students will develop robots that drive around this test environment. The main emphasis will be on making sure the robotic drivers

interact smoothly with other robotic and remote-controlled cars. The latter will be achieved by endowing the robots with socially rational capabilities, thereby enabling them to adapt to local "customs" (like the "Pittsburgh left" and the "Boston rotary").

The co-PIs have a strong record of mentoring students from underrepresented groups, and intend to continue pursuing these efforts in the context of this proposal. In particular, we plan to integrate our research on human–machine collaborations into Artemis, a free summer program directed by co-PI Greenwald that introduces rising 9th grade girls to computational thinking. For example, we can have the Artemis girls teach a robot to collaborate with them on routine tasks, such as navigation or object search.

Finally, we expect to publish the results of the proposed research in top-tier archival, conference proceedings and journals with high impact factors, and to present our work at innovative, non-archival workshops (e.g., the AAAI symposia).

6 Results from Prior NSF Support

Intellectual Merit Greenwald is currently funded by NSF (RI: Small–1217761, 2012–2016, $450k) to build artificial agents that assist humans with decision making in information-rich and time-critical environments like online markets [32, 35, 36, 37]. Previously, she was funded to develop "Methods of Empirical Mechanism Design (EMD)" (CCF: Medium–0905234, 2009–2012, $850k, with Mike Wellman). This project expanded the scope of mechanism design beyond the small-scale, stylized, or idealized domains most previous work was limited to [60, 61, 89]. Before that, she received two prior NSF awards, a PECASE CAREER grant, "Computational Social Choice Theory" (IIS: Career–0133689, 2002–2007, $375k), and "Efficient Link Analysis" (IIS: Small–0534586, 2005–2008, $363k).

Littman is a co-PI on "RI: Medium: Collaborative Research: Teaching Computers to Follow Verbal Instructions" (No. 1065195, 9/11–8/14, $704K) and "RI: Small: Collaborative Research: Speeding Up Learning through Modeling the Pragmatics of Training" (No. 1319305, 10/13–9/15, $440k). These proposals developed autonomous agents that extract preferences from people via verbal interaction and utility feedback [53, 55, 58].

Broader Impact Broader impacts of our past work include undergraduate and graduate student training, design and evaluation of learning modules for outreach programs Learning Exchange and Artemis, and organizing two major computer science conferences (Littman) [25]. We also published novel benchmarks for evaluating grounded language learning programs (Littman), and developed a simulation platform for empirical game-theoretic analysis (Greenwald).

Greenwald also had an additional NSF award (EAGER: 1059570, 2010–2012, $90k) which supported the expansion and evaluation of the Artemis Project. This program has the dual effect of empowering Brown women, while at the same time, exposing girls to computer science. With this money, she successfully expanded the project to Boston University, and ran pilot programs at Columbia and UMBC. Greenwald also has $5k in funding from the Tides Foundation, through the IgniteCS progam, for a project entitled Bringing Computer Science Education to Providence Public Schools.