Teaching AI Agents Ethical Values Using Reinforcement Learning and Policy Orchestration

Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R. Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi

Autonomous cyber-physical agents play an increasingly large role in our lives. To ensure that they behave in ways aligned with the values of society, we must develop techniques that allow these agents to not only maximize their reward in an environment, but also to learn and follow the implicit constraints of society. We detail a novel approach that uses inverse reinforcement learning to learn a set of unspecified constraints from demonstrations and reinforcement learning to learn to maximize environmental rewards. A contextual bandit-based orchestrator then picks between the two policies: constraint-based and environment reward-based. The contextual bandit orchestrator allows the agent to mix policies in novel ways, taking the best actions from either a reward-maximizing or constrained policy. In addition, the orchestrator is transparent on which policy is being employed at each time step. We test our algorithms using Pac-Man and show that the agent is able to learn to act optimally, act within the demonstrated constraints, and mix these two functions in complex ways.

Introduction

Concerns about the ways in which autonomous decision making systems behave when deployed in the real world are growing. Stakeholders worry about systems achieving goals in ways that are not considered acceptable according to values and norms of the impacted community, also called "specification gaming" behaviors [23]. Thus, there is a growing need to understand how to constrain the actions of an AI system by providing boundaries within which the system must operate. To tackle this problem, we may take inspiration from humans, who often constrain the decisions and actions they take according to a number of exogenous priorities, be they moral, ethical, religious, or business values [17, 18, 25], and we may want the systems we build to be restricted in their actions by similar principles [6]. The overriding concern is that the agents we construct may not obey these values while maximizing some objective function [23, 26].

The idea of teaching machines right from wrong has become an important research topic in both AI [34] and related fields [32]. Much of the research at the intersection of artificial intelligence and ethics falls under the heading of *machine ethics*, i.e., adding ethics and/or constraints to a particular system's decision making process [5]. One popular technique to handle these issues is called *value alignment*, i.e., restrict the behavior of an agent so that it can only pursue goals which follow values that are aligned to human values [17, 18, 24].

Another important notion for these autonomous decision making systems is the idea of *transparency* or *interpretability*, i.e., being able to see why the system made the choices it did. Theodorou et al. [29] observe that the Engineering and Physical Science Research Council (EPSRC) Principles of Robotics dictates the implementation of transparency in robotic systems. The authors go on to define transparency in a robotic or autonomous decision making system as "a mechanism to expose the decision making of the robot".

While giving a machine a code of morals or ethics is important, there is still the question of *how to provide the behavioral constraints to the agent*. A popular technique is called the *bottom-up approach*, i.e., teaching a machine what is right and wrong by example [3, 7, 8]. In this paper, we adopt this approach as we consider the case where only examples of the correct behavior are available to the agent, and it must therefore learn from only these examples.

We propose a framework which enables an agent to learn two policies: (1) π_R, a reward maximizing policy obtained through direct interaction with the world, and (2) π_C, obtained via inverse reinforcement learning over demonstrations by humans or other agents of how to obey a set of behavioral constraints in the domain. Our agent then uses a contextual-bandit-based orchestrator [11, 12] to learn to blend the policies in a way that maximizes a convex combination of the rewards and constraints. Within the RL community this can be seen as a particular type of apprenticeship learning [1] where the agent is learning how to be *safe*, rather than only maximizing reward [15].

One may argue that we should employ π_C for all decisions as it will be more 'safe' than employing π_R. Indeed, although one could use π_C exclusively for the agent, there are a number of reasons to employ the orchestrator. First, while the humans or other demonstrators may be good at demonstrating the constrained behavior, they may not provide good examples of how best to maximize reward. Second, the demonstrators may not be as creative as the agent when mixing the two policies [30]. By allowing the orchestrator to learn when to apply which policy, the agent may be able to devise better ways to blend the policies, leading to behavior which both follows the constraints and achieves higher reward than any of the human demonstrations. Third, we may not want to obtain demonstrations of what to do in all parts of the domain e.g., there may be dangerous or hard-to-model regions, or there may be mundane parts of the domain in which human demonstrations are too costly or boring to obtain. In this case, having the agent learn what to do in the non-demonstrated parts through RL is complementary. Finally, as we have argued, interpretability is an important feature to have. Although the policies themselves may not be directly interpretable (though there is recent work in this area [16, 31]), our proposed explicit orchestrator captures the notion of transparency and interpretability as we can see which policy is being applied in real time.

Contributions. We propose and test a novel approach to teach machines to act in ways that achieve and compromise multiple objectives in a given environment. One objective is the desired goal and the other one is a set of behavioral constraints, learnt from examples. Our technique uses aspects of both traditional reinforcement learning and inverse reinforcement learning to identify policies that both maximize rewards and follow particular constraints within an environment. Our agent then blends these policies in novel and interpretable ways using an orchestrator based on the contextual bandits framework. We demonstrate the effectiveness of these techniques on the Pac-Man domain where the agent is able to learn both a reward-maximizing and a constrained policy, and select between these policies in a transparent way based on context, to employ a policy that achieves high reward *and* obeys the demonstrated constraints.

Related Work

Ensuring that autonomous systems act in line with our values while achieving their objectives is a major research topic in AI. These topics have gained popularity among a broad community including philosophers [32] and non-profits [24]. Yu et al. [34] provide an overview of much of the recent research at major AI conferences on ethics in AI.

Agents may need to balance objectives and feedback from multiple sources when making decisions. One prominent example is the case of autonomous cars. There is extensive research from multidisciplinary groups into the questions of when autonomous cars should make lethal decisions [10], how to aggregate societal preferences to make these decisions [22], and how to measure distances between these notions [17, 18]. In a recommender systems setting, a parent or guardian may want the agent to not recommend certain types of movies to children, even if this recommendation could lead to a high reward [7, 8]. Recently, as a compliment to their concrete problems in AI saftey which includes reward hacking and unintended side effects [4], a DeepMind study has compiled a list of specification gaming examples, where very different agents game the given specification by behaving in unexpected (and undesired) ways.[1]

Within the field of reinforcement learning there has been specific work on ethical and interpretable RL. Wu and Lin [33] detail a system that is able to augment an existing RL system to behave ethically. In their framework, the assumption is that, given a set of examples, most of the examples follow ethical guidelines. The system updates the overall policy to obey the ethical guidelines learned from demonstrations using IRL. However, in this system only one policy is maintained so it has no transparency. Laroche and Feraud [14] introduce a system that is capable of selecting among a set of RL policies depending on context. They demonstrate an orchestrator that, given a set of policies for a particular domain, is able to assign a policy to control the next episode. However, this approach use the classical multi-armed bandit, so the state context is not considered.

Interpretable RL has received significant attention in recent years. Luss and Petrik [19] introduce action constraints over states to enhance the interpretability of policies. Verma et al. [31] present a reinforcement learning framework, called Programmatically Interpretable Reinforcement Learning (PIRL), that is designed to generate interpretable and verifiable agent policies. PIRL represents policies using a high-level, domain-specific programming language. Such programmatic policies have the benefit of being more easily interpreted than neural networks, and being amenable to verification by symbolic methods. Additionally, Liu et al. [16] introduce Linear Model U-trees to approximate neural network predictions. An LMUT is learned using a novel on-line algorithm that is well-suited for an active play setting, where the mimic learner observes an ongoing interaction between the neural net and the environment. The transparent tree structure of an LMUT facilitates understanding the learned knowledge by analyzing feature influence, extracting rules, and highlighting the super-pixels in image inputs.

Background

Reinforcement Learning

Reinforcement learning defines a class of algorithms solving problems modeled as a Markov decision process (MDP) [28]. An MDP is usually denoted by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where: $\mathcal{S}$ is a set of possible states; $\mathcal{A}$ is a set of actions; $\mathcal{T}$ is a transition function defined by $\mathcal{T}(s, a, s') = \Pr(s'|s, a)$, where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a reward function; γ is a discount factor that specifies how much long term reward is kept. The goal in an MDP is to maximize the discounted long term reward received. Usually the infinite-horizon objective is considered: $\max \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1})$.

Solutions come in the form of policies $\pi : \mathcal{S} \mapsto \mathcal{A}$, which specify what action the agent should take in any given state deterministically or stochastically. One way to solve this problem is through Q-learning with function approximation [9]. The Q-value of a state-action pair, $\mathcal{Q}(s, a)$, is the expected future discounted reward for taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. A common method to handle very large state spaces is to approximate the $\mathcal{Q}$ function as a linear function of some features. Let $\boldsymbol{\psi}(s, a)$ denote relevant features of the state-action pair $\langle s, a \rangle$. Then, we assume $\mathcal{Q}(s, a) = \boldsymbol{\theta} \cdot \boldsymbol{\psi}(s, a)$, where $\boldsymbol{\theta}$ is an unknown vector to be learned by interacting with the environment. Every time the

[1] 38 AI "specification gaming" examples are available at: https://docs.google.com/spreadsheets/d/e/2PACX-1vRPiprOaC3HsCf5Tuum8bRfzYUiKLRqJmbOoC-32JorNdfyTiRRsR7Ea5eWtvsWzuxo8bjOxCG84dAg/pubhtml

RL agent takes action a from state s, obtains immediate reward r, and reaches new state s', the parameter $\boldsymbol{\theta}$ is updated using

$$\begin{aligned}\text{difference} &= \left[r + \gamma \max_{a'} \mathcal{Q}(s', a')\right] - \mathcal{Q}(s, a) \\ \theta_i &\leftarrow \theta_i + \alpha \cdot \text{difference} \cdot \psi_i(s, a),\end{aligned} \tag{1}$$

where α is the learning rate. A common strategy used for exploration is ϵ-greedy: during the training phase, a random action is played with a probability of ϵ and the action with maximum Q-value is played otherwise. The agent follows this strategy and updates the parameter $\boldsymbol{\theta}$ according to (1) until either the Q-values converge or a maximum number of time-steps is met.

Inverse Reinforcement Learning

IRL seeks to find the most likely reward function $\mathcal{R}_E$, which an expert E is executing [1, 20]. IRL methods assume the presence of an expert that solves an MDP, where the MDP is fully known and observable by the learner except for the reward function. Since the state and action of the expert is fully observable by the learner, it has access to trajectories executed by the expert. A trajectory consists of a sequence of state and action pairs, $Tr = (s_0, a_0, s_1, a_1, \ldots, s_{L-1}, a_{L-1}, s_L)$, where s_t is the state of the environment at time t, a_t is the action played by the expert at the corresponding time and L is the length of this trajectory. The learner is given access to m such trajectories $\{Tr^{(1)}, Tr^{(2)}, \ldots, Tr^{(m)}\}$ to learn the reward function. Since the space of all possible reward functions is extremely large, it is common to represent the reward function as a linear combination of $\ell > 0$ features. $\widehat{\mathcal{R}}_{\boldsymbol{w}}(s, a, s') = \sum_{i=1}^{\ell} w_i \phi_i(s, a, s')$, where w_i are weights to be learned, and $\phi_i(s, a, s') \rightarrow \mathbb{R}$ is a feature function that maps a state-action-state tuple to a real value, denoting the value of a specific feature of this tuple. Current state-of-the-art IRL algorithms utilize feature expectations as a way of evaluating the quality of the learned reward function [27]. For a policy π, the feature expectations starting from state s_o are defined as

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s_t, a_t, s_{t+1}) \middle| \pi\right],$$

where the expectation is taken with respect to the state sequence achieved on taking actions according to π starting from s_0. One can compute an empirical estimate of the feature expectations of the expert's policy with the help of the trajectories $\{Tr^{(1)}, Tr^{(2)}, \ldots, Tr^{(m)}\}$, using

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{L-1} \gamma^t \boldsymbol{\phi}\big(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}\big). \tag{2}$$

Given a weight vector $\boldsymbol{w}$, one can compute the optimal policy $\pi_{\boldsymbol{w}}$ for the corresponding reward function $\widehat{\mathcal{R}}_{\boldsymbol{w}}$, and estimate its feature expectations $\hat{\mu}(\pi_{\boldsymbol{w}})$ in a way similar to (2). IRL compares this $\hat{\mu}(\pi_{\boldsymbol{w}})$ with expert's feature expectations $\hat{\mu}_E$ to learn best fitting weight vectors $\boldsymbol{w}$.

Contextual Bandits

Following Langford and Zhang [13], the contextual bandit problem is defined as follows. At each time $t \in \{0, 1, \ldots, (T-1)\}$, the player is presented with a *context vector* $c(t) \in \mathbb{R}^d$ and must choose an arm $k \in [K] = \{1, 2, \ldots, K\}$. Let $\mathbf{r} = (r_1(t), \ldots, r_K(t))$ denote a reward vector, where $r_k(t)$ is the reward at time t associated with the arm $k \in [K]$. We assume that the expected reward is a linear function of the context, i.e. $\mathbb{E}[r_k(t)|c(t)] = \mu_k^T c(t)$, where μ_k is an unknown weight vector (to be learned from the data) associated with the arm k.

The purpose of a contextual bandit algorithm A is to minimize the cumulative regret. Let $H : C \rightarrow [K]$ where C is the set of possible contexts and $c(t)$ is the context at time t, $h_t \in H$ a hypothesis computed by the algorithm A at time t and $h_t^* = \underset{h_t \in H}{\text{argmax}}\ r_{h_t(c(t))}(t)$ the optimal hypothesis at the same round. The cumulative regret is: $R(T) = \sum_{t=1}^{T} r_{h_t^*(c(t))}(t) - r_{h_t(c(t))}(t)$.

One widely used way to solve the contextual bandit problem is the Contextual Thompson Sampling algorithm (CTS) [2] given as Algorithm 1. In CTS, the reward $r_k(t)$

Algorithm 1 Contextual Thompson Sampling Algorithm

```
1: Initialize: $B_k = I_d$, $\hat{\mu}_k = 0_d$, $f_k = 0_d$ for $k \in [K]$.
2: Foreach $t = 0, 1, 2, \ldots, (T-1)$ do
3:     Sample $\tilde{\mu_k}(t)$ from $N(\hat{\mu}_k, v^2 B_k^{-1})$.
4:     Play arm $k_t = \underset{k \in [K]}{argmax}\ c(t)^\top \tilde{\mu_k}(t)$
5:     Observe $r_{k_t}(t)$
6:     $B_{k_t} = B_{k_t} + c(t)c(t)^T$, $f_{k_t} = f_{k_t} + c(t) r_{k_t}(t)$,
       $\hat{\mu}_{k_t} = B_{k_t}^{-1} f_{k_t}$
7: End
```

for choosing arm k at time t follows a parametric likelihood function $Pr(r(t)|\tilde{\mu})$. Following Agrawal and Goyal [2], the posterior distribution at time $t+1$, $Pr(\tilde{\mu}|r(t)) \propto Pr(r(t)|\tilde{\mu}) Pr(\tilde{\mu})$ is given by a multivariate Gaussian distribution $\mathcal{N}(\hat{\mu_k}(t+1),\ v^2 B_k(t+1)^{-1})$, where $B_k(t) = I_d + \sum_{\tau=1}^{t-1} c(\tau) c(\tau)^\top$, d is the size of the context vectors c, $v = R\sqrt{\frac{24}{z} d \cdot ln(\frac{1}{\gamma})}$ and we have $R > 0$, $z \in [0, 1]$, $\gamma \in [0, 1]$ constants, and $\hat{\mu}(t) = B_k(t)^{-1} (\sum_{\tau=1}^{t-1} c(\tau) r_k(\tau))$.

Every step t consists of generating a d-dimensional sample $\tilde{\mu_k}(t)$ from $\mathcal{N}(\hat{\mu_k}(t),\ v^2 B_k(t)^{-1})$ for each arm. We then decide which arm k to pull by solving for $\text{argmax}_{k \in [K]}\ c(t)^\top \tilde{\mu_k}(t)$. This means that at each time step we are selecting the arm that we expect to maximize the observed reward given a sample of our current beliefs over the distribution of rewards, $c(t)^\top \tilde{\mu_k}(t)$. We then observe the actual reward of pulling arm k, $r_k(t)$ and update our beliefs.

Problem Setting

In our setting, the agent is in multi-objective Markov decision processes (MOMDPs). Instead of the usual scalar reward function $R(s, a, s')$, a reward vector $\vec{R}(s, a, s')$ is present. The vector $\vec{R}(s,a,s')$ consists of l dimensions or components representing the different objectives, i.e., $\vec{R}(s,a,s') = (R_1(s,a,s'),\dots,R_l(s,a,s'))$. However, not all components of the reward vector are observed in our setting. There is an objective $v \in [l]$ that is hidden, and the agent is only allowed to observe expert demonstrations to learn this objective. These demonstrations are given in the form of trajectories $\{Tr^{(1)}, Tr^{(2)}, \dots, Tr^{(m)}\}$. To summarize, for some objectives, the agent has rewards observed from interaction with the environment, and for some objectives the agent has only expert demonstrations. The aim is still the same as single objective reinforcement learning, which is trying to maximize $\sum_{t=0}^{\infty} \gamma^t R_i(s_t,a_t,s_{t+1})$ for each $i \in [l]$.

Proposed Approach

The overall approach we propose, aggregation at the policy phase, is depicted by Figure 1.[2] It has three main components. The first is the IRL component to learn the desired constraints (depicted in green in Figure 1). We apply IRL to the demonstrations depicting desirable behavior, to learn the underlying constraint rewards being optimized by the demonstrations. We then apply RL on these learned rewards to learn a strongly constraint-satisfying policy π_C. Next, we augment this with a pure reinforcement learning component (depicted in red in Figure 1). For this, we directly apply reinforcement learning to the original environment rewards to learn a domain reward maximizing policy π_R.

Now we have two policies: the constraint-obeying policy π_C and the reward-maximizing policy π_R. To combine these two, we use the third component, the orchestrator (depicted in blue in Figure 1). This is a contextual bandit algorithm that orchestrates the two policies, picking one of them to play at each point of time. The context is the state of the environment; the bandit decides which arm (policy) to play at each step. We use a modified CTS algorithm to train the bandit. The context of the bandit is given by features of the current state (for which we want to decide which policy to choose), i.e., $c(t) = \Upsilon(s_t) \in \mathbb{R}^d$.

The exact algorithm used to train the orchestrator is given in Algorithm 2. Apart from the fact that arms are policies (instead of atomic actions), the main difference from the CTS algorithm is the way rewards are fed into the bandit. For simplicity, we call the constraint policy π_C as arm 0 and the reward policy π_R as arm 1. We now go over Algorithm 2. First, all the parameters are initialized as in the CTS algorithm (Line 1). For each time-step in the training phase (Line 3), we do the following. Pick an arm k_t according to the Thompson Sampling algorithm and the context $\Upsilon(s_t)$ (Lines 4 and 5). Play the action according to the chosen policy π_{k_t} (Line 6). This takes us to the next state s_{t+1}. We also observe two rewards (Line 7): (i) the original reward in environment, $r^R_{a_t}(t) = \mathcal{R}(s_t,a_t,s_{t+1})$ and (ii) the constraint rewards according to the rewards learnt by inverse reinforcement learning, i.e., $r^C_{a_t}(t) = \widehat{\mathcal{R}}_C(s_t,a_t,s_{t+1})$. $r^C_{a_t}(t)$ can intuitively be seen as the predicted reward (or penalty) for any constraint satisfaction (or violation) in this step.

Algorithm 2 Orchestrator Based Algorithm

```
1:  **Initialize:** $B_k = I_d$, $\hat{\mu}_k = 0_d$, $f_k = 0_d$ for $k \in \{0,1\}$.
2:  **Observe** start state $s_0$.
3:  **Foreach** $t = 0,1,2,...,(T-1)$ **do**
4:      Sample $\tilde{\mu_k}(t)$ from $N(\hat{\mu}_k, v^2 B_k^{-1})$.
5:      Pick arm $k_t = \arg\max_{k \in \{0,1\}} \Upsilon(s_t)^\top \tilde{\mu_k}(t)$.
6:      Play corresponding action $a_t = \pi_{k_t}(s_t)$.
7:      Observe rewards $r^C_{a_t}(t)$ and $r^R_{a_t}(t)$, and the next state $s_{t+1}$.
8:      Define $r_{k_t}(t) = \lambda\left(r^C_{a_t}(t) + \gamma V^C(s_{t+1})\right)$
                 $+(1-\lambda)\left(r^R_{a_t}(t) + \gamma V^R(s_{t+1})\right)$
9:      Update $B_{k_t} = B_{k_t} + \Upsilon(s_t)\Upsilon(s_t)^\top$, $f_{k_t} = f_{k_t} + \Upsilon(s_t) r_{k_t}(t)$, $\hat{\mu}_{k_t} = B_{k_t}^{-1} f_{k_t}$
10: **End**
```

To train the contextual bandit to choose arms that perform well on both metrics (environment rewards and constraints), we feed it a reward that is a linear combination of $r^R_{a_t}(t)$ and $r^C_{a_t}(t)$ (Line 8). Another important point to note is that $r^R_{a_t}(t)$ and $r^C_{a_t}(t)$ are immediate rewards achieved on taking action a_t from s_t, they do not capture long term effects of this action. In particular, it is important to also look at the "value" of the next state s_{t+1} reached, since we are in the sequential decision making setting. Precisely for this reason, we also incorporate the value-function of the next state s_{t+1} according to both the reward maximizing component and constraint component (which encapsulate the long-term rewards and constraint satisfaction possible from s_{t+1}). This gives exactly Line 8, where V^C is the value-function according the constraint policy π_C, and V^R is the value-function according to the reward maximizing policy π_R.

In this equation, λ is a hyperparameter chosen by a user to decide how much to trade off environment rewards for constraint satisfaction. For example, when λ is set to 0, the orchestrator would always play the reward policy π_R, while for $\lambda = 1$, the orchestrator would always play the constraint policy π_C. For any value of λ in-between, the orchestrator is expected to pick policies at each point of time that would perform well on both metrics (weighed according to λ). Finally, for the desired reward $r_{k_t}(t)$ and the context $\Upsilon(s_t)$, the parameters of the bandit are updated according to the CTS algorithm (Line 9).

Alternative Approaches

Observe that in the proposed approach, we combine or "aggregate" the two objectives at the highest level, i.e., at the policy stage. Alternative approaches could be to combine the two objectives at lower levels, i.e., the reward stage or the demonstrations stage itself.

- **Aggregation at reward phase.** As before, we can perform inverse reinforcement learning to learn the underlying rewards capturing the desired constraints. Now, instead of learning a policy for each of the two

[2] Alternative formulations of aggregation at the reward phase and aggregation at the data phase are discussed in the appendix.

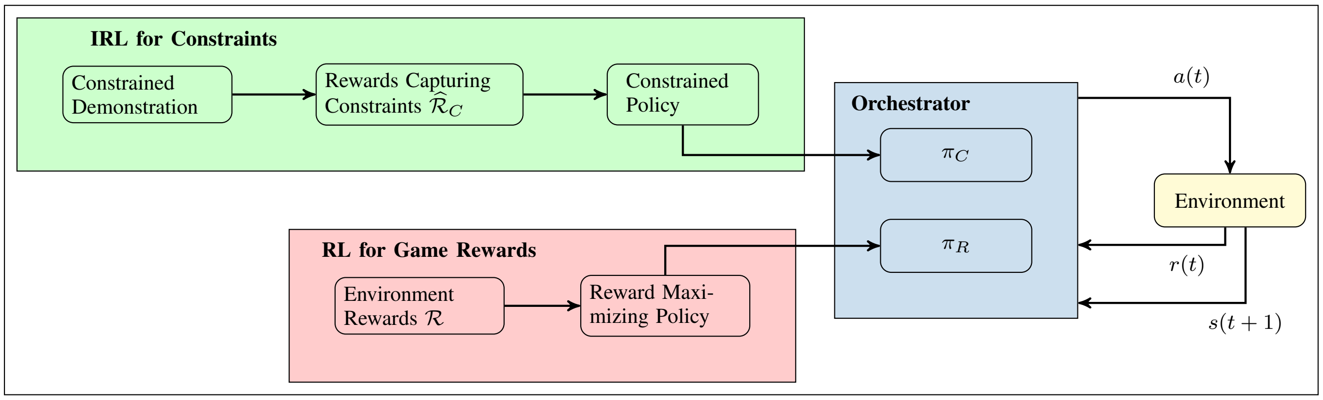


Figure 1 Overview of our system. At each time step the Orchestrator selects between two policies, π_C and π_R depending on the observations from the Environment. The two policies are learned before engaging with the environment. π_C is obtained using IRL on the demonstrations to learn a reward function that captures demonstrated constraints. The second, π_R is obtained by the agent through RL on the environment.

reward functions (environment rewards and constraint rewards) followed by aggregating them, we could just combine the reward functions themselves. And then, we could learn a policy on these "aggregated" rewards that performs well on both the objectives, environment reward, and constraints. This process captures the intuitive idea of "incorporating the constraints into the environment rewards." Hence, if we were explicitly given the penalty of violating constraints this would be ideal. However, note that this is a *top-down* approach and in this study we want to focus on the example driven, or *bottoms-up* approach.

- **Aggregation at data phase.** Moving another step backward, we could aggregate the two objectives of play at the data phase. This could be performed as follows. We perform pure reinforcement learning as in the proposed approach given in Figure 1 (depicted in red). Once we have our reward maximizing policy π_R, we use it to generate numerous reward-maximizing demonstrations. Then, we combine these environment reward trajectories with the original constrained demonstrations, aggregating the two objectives in the process. And once we have the combined data, we can perform inverse reinforcement learning to learn the appropriate rewards, followed by reinforcement learning to learn the corresponding policy.

Aggregation at the policy phase is the proposed approach in the main paper, where we go all the way to the end of the pipeline, learning a policy for each of the objectives followed by aggregation. A similar parameter to λ there can be used by the reward aggregation and data aggregation approaches as well, to decide how to weigh the two objectives while performing the corresponding aggregation.

The question now is, "which of these aggregation procedures is the most useful?". The reason we use aggregation at the policy stage is to gain *interpretability*. Using an orchestrator to pick a policy at each point of time helps us identify which policy is being played at each point of time and also the reason for which it is being chosen (in the case of an interpretable orchestrator, which it is in our case).

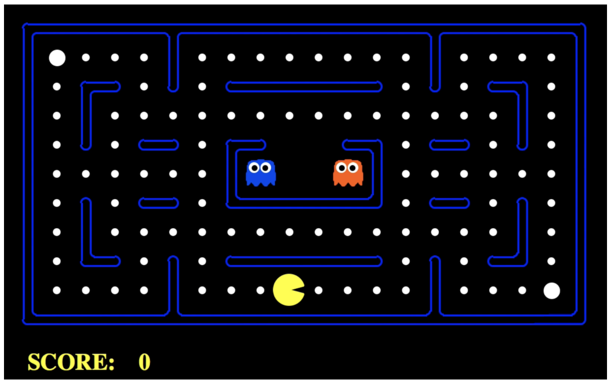


Figure 2 Layout of Pac-Man

Demonstration on Pac-Man

We demonstrate the applicability of the proposed algorithm using the classic game of Pac-Man.

Details of the Domain

The layout of Pac-Man we use is given in Figure 2. The rules for the environment (adopted from Berkeley AI Pac-Man[3]) are as follows. The goal of the agent is to eat all the dots in the maze, known as Pac-Dots, as soon as possible while simultaneously avoiding collision with ghosts. On eating a Pac-Dot, the agent obtains a reward of $+10$. On successfully eating all the Pac-Dots, the agent obtains a reward of $+500$. In the meantime, the ghosts roam the maze trying to kill Pac-Man. On collision with a ghost, Pac-Man loses the game and gets a reward of -500. The game also has two special dots called Power Pellets in the corners of the maze, which on consumption, give Pac-Man the temporary ability of "eating" ghosts. During this phase, the ghosts are in a "scared" state for 40 frames and move at half their speed. On eating a ghost, the agent gets a reward of $+200$, the ghost returns to the center box and returns to its normal "unscared" state. Finally, there is a constant

[3] http://ai.berkeley.edu/project_overview.html

time-penalty of -1 for every step taken.

For the sake of demonstration of our approach, we define *not eating ghosts* as the desirable constraint in the game of Pac-Man. However, recall that this constraint is not given explicitly to the agent, but only through examples. To play optimally in the original game one should eat ghosts to earn bonus points, but doing so is being demonstrated as undesirable. Hence, the agent has to combine the goal of collecting the most points while not eating ghosts.

Details of the Pure RL

For the reinforcement learning component, we use Q-learning with linear function approximation as described in Section . Some of the features we use for an $\langle s, a\rangle$ pair (for the $\boldsymbol{\psi}(s, a)$ function) are: "whether food will be eaten", "distance of the next closest food", "whether a scared (unscared) ghost collision is possible" and "distance of the closest scared (unscared) ghost".

For the layout of Pac-Man we use (shown in Figure 2), an upper bound on the maximum score achievable in the game is 2170. This is because there are 97 Pac-Dots, each ghost can be eaten at most twice (because of two capsules in the layout), Pac-Man can win the game only once and it would require more than 100 steps in the environment. On playing a total of 100 games, our reinforcement learning algorithm (the reward maximizing policy π_R) achieves an average game score of 1675.86, and the maximum score achieved is 2144. We mention this here, so that the results in Section can be seen in appropriate light.

Details of the IRL

For inverse reinforcement learning, we use the linear IRL algorithm as described in Section . For Pac-Man, observe that the original reward function $\mathcal{R}(s, a, s')$ depends only on the following factors: "number of Pac-Dots eating in this step (s, a, s')", "whether Pac-Man has won in this step", "number of ghosts eaten in this step" and "whether Pac-Man has lost in this step". For our IRL algorithm, we use exactly these as the features $\boldsymbol{\phi}(s, a, s')$. As a sanity check, when IRL is run on environment reward optimal trajectories (generated from our policy π_R), we recover something very similar to the original reward function $\mathcal{R}$. In particular, the weights of the reward features learned is given by $^1/_{1000}[+2.44, +138.80, +282.49, -949.17]$, which when scaled is almost equivalent to the true weights $[+10, +500, +200, -500]$ in terms of their optimal policies. The number of trajectories used for this is 100.

Ideally, we would prefer to have the constrained demonstrations given to us by humans, but for the sake of simplicity we generate them synthetically as follows. We learn a policy $\pi_C^\star$ by training it on the game with the original reward function $\mathcal{R}$ augmented with a very high negative reward (-1000) for eating ghosts. This causes $\pi_C^\star$ to play well in the game while avoiding eating ghosts as much as possible.[4] Now, to emulate erroneous human behavior, we use $\pi_C^\star$ with an error probability of 3%. That is, at every time step, with 3% probability we pick a completely random action, and otherwise follow $\pi_C^\star$. This gives us our constrained demonstrations, on which we perform inverse reinforcement learning to learn the rewards capturing the constraints. The weights of the reward function learned is given by $^1/_{1000}[+2.84, +55.07, -970.59, -234.34]$, and it is evident that it has learned that eating ghosts strongly violates the favorable constraints. The number of demonstrations used for this is 100. We scale these weights to have a similar L_1 norm as the original reward weights $[+10, +500, +200, -500]$, and denote the corresponding reward function by $\widehat{\mathcal{R}}_C$.

Finally, running reinforcement learning on these rewards $\widehat{\mathcal{R}}_C$, gives us our constraint policy π_C. On playing a total of 100 games, π_C achieves an average game score of 1268.52 and eats just 0.03 ghosts on an average. Note that, when eating ghosts is prohibited in the domain, an upper bound on the maximum score achievable is 1370.

Details of the Contextual Bandit

The features of the state we use for context $c(t)$ are: (i) A constant 1 to represent the bias term, and (ii) The distance of Pac-Man from the closest scared ghost in s_t. One could use a more sophistical context with many more features, but we use this restricted context to demonstrate a very interesting behavior (shown in Section).

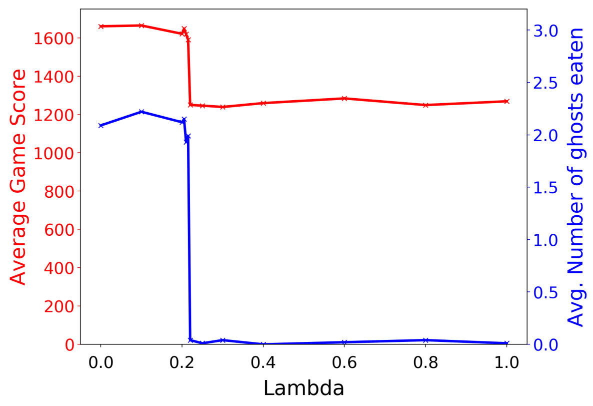
Evaluation

We measure performance on two metrics, (i) the total score achieved in the game (the environment rewards) and (ii) the number of ghosts eaten (the constraint violation). We also observe how these metrics vary with λ. For each value of λ, the orchestrator is trained for 100 games. The results are shown in Figure 3. Each point in the graph is averaged over 100 test games.

The graph shows a very interesting pattern. When λ is at most than 0.215, the agent eats a lot of ghosts, but when it is above 0.22, it eats almost no ghosts. In other words, there is a value λ_o which behaves as a tipping point, across which there is drastic change in behavior. Beyond the threshold, the agent learns that eating ghosts is not worth the score it is getting and so it avoids eating as much as possible. On the other hand, when λ is smaller than λ_o, it learns the reverse and eats as many ghosts as possible.

Policy-switching. As mentioned before, one important property of our approach is interpretability, we know exactly which policy is being played at each time. For moderate values of $\lambda > \lambda_o$, the orchestrator learns a very interesting policy-switching technique: whenever at least one of the ghosts in the domain is scared, it plays π_C, but if no ghosts are scared, it plays π_R. In other words, it starts the game

[4]We do this only for generating demonstrations. In real domains, we would not have access to the exact constraints that we want to be satisfied, and hence a policy like $\pi_C^\star$ cannot be learned; learning from human demonstrations would then be essential.



[t]

Figure 3 Both performance metrics as λ is varied. The red curve depicts the average game score achieved, and the blue curve depicts the average number of ghosts eaten.

playing π_R until a capsule is eaten. As soon as the first capsule is eaten, it switches to π_C until the scared timer runs off. Then it switches back to π_R until another capsule is eaten, and so on. It has learned a very intuitive behavior: when there is no scared ghost, there is no possibility of violating constraints. Hence, the agent is as greedy as possible (i.e., play π_R). However, when there are scared ghosts, it is better to be safe (i.e., play π_C).

Discussion and Extensions

In this paper, we have considered the problem of autonomous agents learning policies that are constrained by implicitly-specified norms and values while still optimizing their policies with respect to environmental rewards. We have taken an approach that combines IRL to determine constraint-satisfying policies from demonstrations, RL to determine reward-maximizing policies, and a contextual bandit to orchestrate between these policies in a transparent way. This proposed architecture and approach for the problem is novel. It also requires a novel technical contribution in the contextual bandit algorithm because the arms are policies rather than atomic actions, thereby requiring rewards to account for sequential decision making. We have demonstrated the algorithm on the Pac-Man video game and found it to perform interesting switching behavior among policies.

We feel that the contribution herein is only the starting point for research in this direction. We have identified several avenues for future research, especially with regards to IRL. We can pursue deep IRL to learn constraints without hand-crafted features, develop an IRL that is robust to noise in the demonstrations, and research IRL algorithms to learn from just one or two demonstrations (perhaps in concert with knowledge and reasoning). In real-world settings, demonstrations will likely be given by different users with different versions of abiding behavior; we would like to exploit the partition of the set of traces by user to improve the policy or policies learned via IRL. Additionally, the current orchestrator selects a single policy at each time, but more sophisticated policy aggregation techniques for combining or mixing policies is possible. Lastly, it would be interesting to investigate whether the policy aggregation rule (λ in the current proposal) can be learned from demonstrations.

Acknowledgments

This work was conducted under the auspices of the IBM Science for Social Good initiative. An abbreviated version of this paper appeared at the 2019 International Joint Conference on Artificial Intelligence (IJCAI 2019) [21].

References

1. P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.
2. Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *ICML (3)*, pages 127–135, 2013.
3. C. Allen, I. Smit, and W. Wallach. Artificial morality: Top-down, bottom-up, and hybrid approaches. *Ethics and Information Technology*, 7(3):149–155, 2005.
4. Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.
5. M. Anderson and S. L. Anderson. *Machine Ethics*. Cambridge University Press, 2011.
6. T. Arnold, D. Kasenberg, and M. Scheutzs. Value alignment or misalignment - what will keep systems accountable? In *AI, Ethics, and Society, Papers from the 2017 AAAI Workshops*, 2017.
7. A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi. Using contextual bandits with behavioral constraints for constrained online movie recommendation. In *Proc. of the 27th Intl. Joint Conference on AI (IJCAI)*, 2018.
8. A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi. Incorporating behavioral constraints in online ai systems. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.
9. D. Bertsekas and J. Tsitsiklis. Neuro-dynamic programming. *Athena Scientific*, 1996.
10. Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. The social dilemma of autonomous vehicles. *Science*, 352(6293):1573–1576, 2016.
11. D. Bouneffouf and I. Rish. A survey on practical applications of multi-armed and contextual bandits. *CoRR*, abs/1904.10040, 2019. URL http://arxiv.org/abs/1904.10040.
12. D. Bouneffouf, I. Rish, G. A. Cecchi, and R. Féraud. Context attentive bandits: Contextual bandit with restricted context. In *Proc. of the 26th Intl. Joint Conference on AI (IJCAI)*, pages 1468–1475, 2017. doi: 10.24963/ijcai.2017/203. URL https://doi.org/10.24963/ijcai.2017/203.
13. John Langford and Tong Zhang. The Epoch-Greedy

Algorithm for Contextual Multi-armed Bandits. In *Proc. 21st NIPS*, 2008.

14. Romain Laroche and Raphael Feraud. Reinforcement learning algorithm selection. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2017.
15. J. Leike, M. Martic, V. Krakovna, P.A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.
16. Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. *CoRR*, abs/1807.05887, 2018. URL http://arxiv.org/abs/1807.05887.
17. A. Loreggia, N. Mattei, F. Rossi, and K. B. Venable. Preferences and ethical principles in decision making. In *Proc. of the 1st AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, 2018.
18. A. Loreggia, N. Mattei, F. Rossi, and K. B. Venable. Value alignment via tractable preference distance. In R. V. Yampolskiy, editor, *Artificial Intelligence Safety and Security*, chapter 18. CRC Press, 2018.
19. Ronny Luss and Marek Petrik. Interpretable policies for dynamic product recommendations. In *Proc. Conf. Uncertainty Artif. Intell.*, page 74, New York, USA, June 2016.
20. Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL http://dl.acm.org/citation.cfm?id=645529.657801.
21. R. Noothigattu, D. Bouneffouf, N. Mattei, R. Chandra, P. Madan, K. Varshney, M. Campbell, M. Singh, and F. Rossi. Teaching AI agents ethical values using reinforcement learning and policy orchestration (extended abstract). In *Proc. of the 28th Intl. Joint Conference on AI (IJCAI)*, 2019.
22. Ritesh Noothigattu, Snehalkumar'Neil'S Gaikwad, Edmond Awad, Sohan Dsouza, Iyad Rahwan, Pradeep Ravikumar, and Ariel D Procaccia. A voting-based system for ethical decision making. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2017.
23. F. Rossi and N. Mattei. Building ethically bounded AI. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.
24. S. Russell, D. Dewey, and M. Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.
25. A. Sen. Choice, ordering and morality. In S. Körner, editor, *Practical Reason*. Blackwell, Oxford, 1974.
26. T. Simonite. When bots teach themselves to cheat. *Wired Magazine*, August 2018.
27. Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2017.
28. Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.
29. Andreas Theodorou, Robert H Wortham, and Joanna J Bryson. Why is my robot behaving like that? designing transparency for real time inspection of autonomous robots. In *AISB Workshop on Principles of Robotics*. University of Bath, 2016.
30. D. Ventura and D. Gates. Ethics as aesthetic: A computational creativity approach to ethical behavior. In *Proc. Int. Conference on Computational Creativity (ICCC)*, pages 185–191, 2018.
31. Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 5052–5061, 2018. URL http://proceedings.mlr.press/v80/verma18a.html.
32. W. Wallach and C. Allen. *Moral Machines: Teaching Robots Right From Wrong*. Oxford University Press, 2008.
33. Yueh-Hua Wu and Shou-De Lin. A low-cost ethics shaping approach for designing reinforcement learning agents. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2017.
34. H. Yu, Z. Shen, C. Miao, C. Leung, V. R. Lesser, and Q. Yang. Building ethics into artificial intelligence. In *Proc. of the 27th Intl. Joint Conference on AI (IJCAI)*, pages 5527–5533, 2018.

Ritesh Noothigattu *Carnegie Mellon University, Pittsburgh, PA, USA (riteshn@cmu.edu).* Ritesh Noothigattu is a PhD candidate in the Machine Learning Department at Carnegie Mellon University advised by Prof. Ariel Procaccia. He received a Bachelor of Technology in Computer Science & Engineering from Indian Institute of Technology Delhi in 2016 and a Master of Science in Machine Learning from Carnegie Mellon University in 2019. He is currently interested in problems at the intersection of machine learning and algorithmic game theory, computational social choice and fairness.

Djallel Bouneffouf *IBM Research, Yorktown Heights, NY, USA (djallel.bouneffouf@ibm.com).* Djallel Bouneffouf is a Research Staff Member in the IBM Research AI organization at the IBM T. J. Watson Research Center. He received his Ph.D. in Computer Science from the University of Paris-Saclay in 2013. He is primarily interested in the old goal of Artificial Intelligence (AI) which is building autonomous system that can learn to be competent in uncertain environments. His main research areas are online learning, Bandit, active learning, clustering, and reinforcement learning.

Nicholas Mattei *Tulane University, New Orleans, LA, USA (nsmattei@tulane.edu).* Nicholas Mattei received a B.S., M.S., and Ph.D. in Computer Science from the University of Kentucky. Dr. Mattei was an Aerospace Research Engineer for NASA Ames Research Center in Mountain View, CA, USA from 2006 to 2009; he was a Researcher for National ICT Australia (NICTA), Sydney, Australia, from 2012 to 2015; he was a Senior Researcher at the Commonwealth Science Research Organization (CSIRO), Sydney, Australia, from 2015 to 2016; he was a Research Staff Member at IBM T.J. Watson Research Center, Yorktown Heights, NY USA; from 2016 to 2018, and from 2019 is an Assistant Professor of Computer Science at Tulane University, New Orleans, LA, USA. He serves on the board of the Journal of Artificial Intelligence Research (JAIR) and is the vice president of the ACM Special Interest Group on Artificial Intelligence (ACM:SIGAI).

Rachita Chandra *IBM Research, Cambridge, MA, USA* *(rachitac@us.ibm.com)*. Rachita Chandra is a Solutions Architect and Developer at IBM Research. She received her B.Sc. and M.Sc. in Electrical and Computer Engineering from Carnegie Mellon University. Prior to this, she worked at Watson Health, Yahoo and Silver Spring Networks building scalable backend systems and machine learning solutions.

Piyush Madan *IBM Research, Cambridge, MA, USA (piyush.madan1@ibm.com)*. Piyush has been involved in Real World Evidence (RWE) platform and observation studies with multiple clients at IBM. Prior to joining IBM, he was a graduate student studying Health Informatics at Johns Hopkins' School of Medicine. He has worked in different healthcare settings around the world. Before joining graduate school, Piyush was based out of Haiti to support USAID and PEPFAR program for prevention of mother to child transmission of HIV. His other experiences include contribution to an opensource EMR systems as a Google Summer of Code developer and full stack developement for a Health IT startup based in New Delhi, his home town.

Kush R. Varshney *IBM Research, Yorktown Heights, NY, USA (krvarshn@us.ibm.com)*. Kush R. Varshney received the B.S. degree from Cornell University in 2004. He received the S.M. degree in 2006 and the Ph.D. degree in 2010 from the Massachusetts Institute of Technology. Dr. Varshney is a principal research staff member and manager with IBM Research at the Thomas J. Watson Research Center, Yorktown Heights, NY, where he leads the machine learning group in the Foundations of Trusted AI department. He is the founding co-director of the IBM Science for Social Good initiative. He is a senior member of the IEEE and a member of the Partnership on AI's Safety-Critical AI expert group.

Murray Campbell *IBM Research, Yorktown Heights, NY, USA (mcam@us.ibm.com)*. Murray Campbell is a Distinguished Research Staff Member at the IBM T. J. Watson Research Center, where he is a manager in the IBM Research AI organization. He received his B.Sc. and M.Sc. in Computing Science from the University of Alberta, and his Ph.D. in Computer Science from Carnegie Mellon University. He was a member of the IBM team that developed Deep Blue, which was the first computer to defeat the human world chess champion in a match. He received numerous awards for Deep Blue, including the Allen Newell Medal for Research Excellence and the Fredkin Prize. He is an ACM Distinguished Scientist and a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI).

Moninder Singh *IBM Research, Yorktown Heights, NY, USA (moninder@us.ibm.com)*. Moninder Singh is a Research Staff Member in the IBM Research AI organization at the IBM T. J. Watson Research Center. He received his Ph.D. in Computer and Information Science from the University of Pennsylvania in 1998. He is primarily interested in developing and deploying solutions for interesting analytics and decision support problems. His main research areas are machine learning and data mining, artificial intelligence, data privacy, information retrieval, probabilistic modeling and reasoning, and text mining.

Francesca Rossi *IBM Research, Yorktown Heights, NY, USA (francesca.rossi2@ibm.com)*. Francesca Rossi is a Distinguished Research Staff Member at the IBM T. J. Watson Research Center, and the IBM AI Ethics Global Leader. She received a PhD in Computer Science in 1993. She has been an assistant professor of computer science at the University of Pisa in 1992-1998 and a professor of computer science at the University of Padova 1998-2018. Her main research areas are decision making, constraint reasoning, preferences, and AI ethics. She published about 200 papers and edited about 20 volumes. She has been the program chair of IJCAI 2013 and the general chair of AAAI 2020. She is a AAAI and an EurAI fellow, and she has been awarded the 2019 IJCAI Distinguished Service award.