# Teaching AI Agents Ethical Values Using Reinforcement Learning and Policy Orchestration (Extended Abstract) *

**Ritesh Noothigattu,**[3] **Djallel Bouneffouf,**[1] **Nicholas Mattei,**[4] **Rachita Chandra,**[2] **Piyush Madan,**[2] **Kush R. Varshney,**[1] **Murray Campbell,**[1] **Moninder Singh,**[1] and **Francesca Rossi**[1]
[1]IBM Research, Yorktown Heights, NY, USA
[2]IBM Research, Cambridge, MA, USA
[3]Carnegie Mellon University, Pittsburgh, PA, USA
[4]Tulane University, New Orleans, LA, USA

## Abstract

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.

## 1 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 [Rossi and Mattei, 2019]. 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 [Sen, 1974; Loreggia *et al.*, 2018a; 2018b], and we may want the systems we build to be restricted in their actions by similar principles [Arnold *et al.*, 2017]. The overriding concern is that the agents we construct may not obey these values while maximizing some objective function [Simonite, 2018; Rossi and Mattei, 2019].

The idea of teaching machines right from wrong has become an important research topic in both AI [Yu *et al.*, 2018] and related fields [Wallach and Allen, 2008]. 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 [Anderson and Anderson, 2011]. 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 aligned to human values [Russell *et al.*, 2015].

While giving a machine a code of ethics is important, the question of *how to provide the behavioral constraints to the agent* remains. A popular technique, the *bottom-up approach*, teaches a machine right and wrong by example [Allen *et al.*, 2005; Balakrishnan *et al.*, 2019; 2018]. Here we adopt this approach as we consider the case where only examples of the correct behavior are available.

We propose a framework which enables an agent to learn two policies: (1) $\pi_R$, a reward maximizing policy obtained through direct interaction with the world, and (2) $\pi_C$, 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 [Bouneffouf and Rish, 2019; Bouneffouf *et al.*, 2017] 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 [Abbeel and Ng, 2004] where the agent is learning how to be *safe*, rather than only maximizing reward [Leike *et al.*, 2017].

One may argue that we should employ $\pi_C$ for all decisions as it will be more 'safe' than employing $\pi_R$. Indeed, although one could use $\pi_C$ exclusively, there are a number of reasons to employ the orchestrator. First, while demonstrators may be good at demonstrating the constrained behavior, they may not provide good examples of maximizing reward. Second, the demonstrators may not be as creative as the agent when mixing the two policies [Ventura and Gates, 2018]. By allowing the orchestrator to learn when to apply which policy, the

*This paper is an extended abstract of an article in the the IBM Journal of Research & Development [Noothigattu *et al.*, 2019].

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 in all parts of the domain e.g., there may be dangerous parts of the domain in which human demonstrations are too costly to obtain. In this case, having the agent learn what to do in the non-demonstrated parts through RL is complementary.

**Contributions.** We propose and test a novel approach to teach machines to act in ways blend multiple objectives. One objective is the desired goal and the other is a set of behavioral constraints, learned 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. Our agent then blends these policies in novel and interpretable ways using an orchestrator. We demonstrate the effectiveness of these techniques on Pac-Man 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.

### 1.1 Problem Setting and Notation

Reinforcement learning defines a class of algorithms solving problems modeled as a Markov decision process (MDP). An MDP is 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; $\gamma$ is a discount factor that specifies how much long term reward is kept. The goal is to maximize the discounted long term reward. 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 [Bertsekas and Tsitsiklis, 1996]. 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.

Our problem is a *multi-objective* MDP. Instead of the scalar reward function $R(s, a, s')$, we have a reward vector $\vec{R}(s, a, s')$ consisting of $l$ dimensions representing the different objectives. However, not all components of the reward vector are observed. There is an objective $v \in [l]$ that is hidden, and the agent is only allowed to observe demonstrations to learn this objective. These demonstrations are given in the form of trajectories $\{Tr^{(1)}, Tr^{(2)}, \ldots, Tr^{(m)}\}$. To summarize, for some objectives, the agent has rewards observed from interaction with the environment, and for some objectives the agent has only demonstrations. The aim is the same as single objective reinforcement learning: to maximize $\sum_{t=0}^{\infty} \gamma^t R_i(s_t, a_t, s_{t+1})$ for each $i \in [l]$.

## 2 Proposed Approach

The overall approach we propose, aggregation at the policy phase, is depicted by Figure 1. It has three main components. The first is the IRL component to learn the desired constraints (depicted in green). We apply IRL to 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 $\pi_C$. We augment $\pi_C$ with a pure reinforcement learning component applied to the original environment rewards (depicted in red) to learn a domain reward maximizing policy $\pi_R$.

Now we have two policies: the constraint-obeying policy $\pi_C$ and the reward-maximizing policy $\pi_R$. To combine them, we use the third component, the orchestrator (depicted in blue). 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 1. 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, let the constraint policy $\pi_C$ be arm 0 and the reward policy $\pi_R$ be arm 1. 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 $\pi_{k_t}$ (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 1** 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 $r^C_{a_t}(t)$, $r^R_{a_t}(t)$, and 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

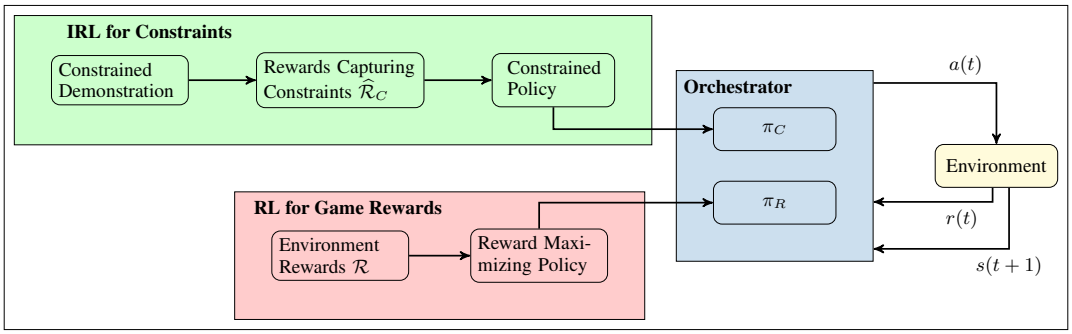


Figure 1: Overview of our system. At each time step the Orchestrator selects between two policies, $\pi_C$ and $\pi_R$ depending on the observations from the Environment. The two policies are learned before engaging with the environment. $\pi_C$ is obtained using IRL on the demonstrations to learn a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

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 $\pi_C$, and $V^R$ is the value-function according to the reward maximizing policy $\pi_R$.

In this equation, $\lambda$ is a hyperparameter chosen by a user to decide how much to trade off environment rewards for constraint satisfaction. For example, when $\lambda$ is set to 0, the orchestrator would always play the reward policy $\pi_R$, while for $\lambda = 1$, the orchestrator would always play the constraint policy $\pi_C$. For any value of $\lambda$ in-between, the orchestrator is expected to pick policies at each point of time that would perform well on both metrics (weighed according to $\lambda$). 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).

## 3 Demonstration on Pac-Man

We demonstrate the applicability of the proposed algorithm using the classic game of Pac-Man. The rules for the environment (adopted from Berkeley AI Pac-Man[1]) 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 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.

### 3.1 Details of the Pure RL

For the reinforcement learning component, we use Q-learning with linear function approximation. 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, 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 $\pi_R$) 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 4 can be seen in appropriate light.

### 3.2 Details of the IRL

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 $\pi_R$), we recover something very similar to the original reward function

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

$\mathcal{R}$. 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 $\pi^\star_C$ 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 $\pi^\star_C$ to play well in the game while avoiding eating ghosts as much as possible.[2] Now, to emulate erroneous human behavior, we use $\pi^\star_C$ with an error probability of 3%. That is, at every time step, with 3% probability we pick a completely random action, and otherwise follow $\pi^\star_C$. 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 $\pi_C$. On playing a total of 100 games, $\pi_C$ 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.

## 4 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 $\lambda$. For each value of $\lambda$, the orchestrator is trained for 100 games. The results are shown in Figure 2. Each point is averaged over 100 test games.

The graph shows a very interesting pattern. When $\lambda$ 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 $\lambda_o$ 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 $\lambda$ is smaller than $\lambda_o$, it learns the reverse and eats as many ghosts as possible.

**Policy-switching.** 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 $\pi_C$, but if no ghosts are scared, it plays $\pi_R$. In other words, it starts the game playing $\pi_R$ until a capsule is eaten. As soon as the first capsule is eaten, it switches to $\pi_C$ until the scared timer runs off. Then it switches back to $\pi_R$ 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 $\pi_R$). However, when there are scared ghosts, it is better to be safe (i.e., play $\pi_C$).

[2] 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 $\pi^\star_C$ cannot be learned; learning from human demonstrations would then be essential.

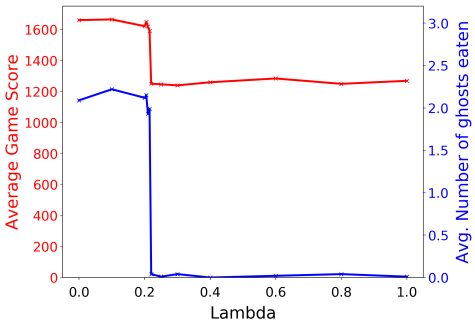


Figure 2: Both performance metrics as $\lambda$ is varied. The red curve depicts the average game score achieved, and the blue curve depicts the average number of ghosts eaten.

## 5 Discussion

In this paper, we considered the problem of autonomous agents learning policies constrained by implicitly-specified 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 Pac-Man and found it to perform interesting switching behavior among policies.

The contribution herein is only a starting point. We can pursue deep IRL to learn constraints without hand-crafted features 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 ($\lambda$) can be learned from demonstrations.

## Acknowledgments

This work was conducted under the auspices of the IBM Science for Social Good initiative.

## References


[Abbeel and Ng, 2004] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

[Allen *et al.*, 2005] 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.

[Anderson and Anderson, 2011] M. Anderson and S. L. Anderson. *Machine Ethics*. Cambridge University Press, 2011.

[Arnold *et al.*, 2017] 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.

[Balakrishnan *et al.*, 2018] 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.

[Balakrishnan *et al.*, 2019] 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.

[Bertsekas and Tsitsiklis, 1996] D. Bertsekas and J. Tsitsiklis. Neuro-dynamic programming. *Athena Scientific*, 1996.

[Bouneffouf and Rish, 2019] D. Bouneffouf and I. Rish. A survey on practical applications of multi-armed and contextual bandits. *CoRR*, abs/1904.10040, 2019.

[Bouneffouf *et al.*, 2017] 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.

[Leike *et al.*, 2017] 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.

[Loreggia *et al.*, 2018a] 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.

[Loreggia *et al.*, 2018b] 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.

[Noothigattu *et al.*, 2019] 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. *IBM Journal of Research & Development*, 2019. To Appear.

[Rossi and Mattei, 2019] F. Rossi and N. Mattei. Building ethically bounded AI. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[Russell *et al.*, 2015] S. Russell, D. Dewey, and M. Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.

[Sen, 1974] A. Sen. Choice, ordering and morality. In S. Körner, editor, *Practical Reason*. Blackwell, Oxford, 1974.

[Simonite, 2018] T. Simonite. When bots teach themselves to cheat. *Wired Magazine*, August 2018.

[Ventura and Gates, 2018] 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.

[Wallach and Allen, 2008] W. Wallach and C. Allen. *Moral Machines: Teaching Robots Right From Wrong*. Oxford University Press, 2008.

[Yu *et al.*, 2018] 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.