Strong Mitigation: Nesting Search for Good Policies Within Search for Good Reward

Jeshua Bratman
Computer Science & Eng.
University of Michigan
jeshua@umich.edu

Satinder Singh
Computer Science & Eng.
University of Michigan
baveja@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Jonathan Sorg
Facebook
jdsorg@umich.edu

ABSTRACT

Recent work has defined an optimal reward problem (ORP) in which an agent designer, with an objective reward function that *evaluates* an agent's behavior, has a choice of what reward function to build into a learning or planning agent to *guide* its behavior. Existing results on ORP show *weak mitigation* of limited computational resources, i.e., the existence of reward functions so that agents when guided by them do better than when guided by the objective reward function. These existing results ignore the cost of finding such good reward functions. We define a nested optimal reward and control architecture that achieves *strong mitigation* of limited computational resources. We show empirically that the designer is better off using the new architecture that spends some of its limited resources learning a good reward function instead of using all of its resources to optimize its behavior with respect to the objective reward function.

Categories and Subject Descriptors

H.4 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Reinforcement Learning, Planning

1. INTRODUCTION

Reinforcement learning (RL) and decision-theoretic planning approaches for solving sequential decision making problems typically start with an agent's reward function and focus on designing learning/planning architectures for the agent that allow it to efficiently optimize some cumulative measure of the given reward. Recently Singh et al. [1] have argued that for many applications a more accurate and potentially useful way to describe the applied RL setting is as an agent designer who has an *objective* reward function that prescribes *preferences* over agent behavior, so that when building an agent, the designer is free to choose any reward function as *parameters* for the agent. In this view the designer's objective reward function (as preferences) *evaluates* agent behavior while the agent's reward function (as parameters) *guides* the agent's behavior via its planning/learning algorithm. The conventional RL approach confounds these two roles of rewards, evaluation and guidance, by using the objective reward for both. This raises the question of what benefits, if any, might there be to breaking the conventional equality between preferences and parameters? Singh et al. formalize this question via a new *optimal reward problem* (ORP) that defines the optimal reward function as one that—if used to guide agent behavior—ends up maximizing the utility as evaluated by the agent designer's objective reward function. By solving the ORP for classes of agents and environments, Sorg et al. [2] developed a key insight: good reward functions mitigate agent boundedness, i.e., choosing a reward function to guide the agent different from the evaluatory objective reward function can improve agent performance by mitigating inevitable agent limitations (e.g., bounds on architecture or on computational resources).

In this paper we draw a distinction between two ways in which solving the optimal reward problem might be beneficial to the agent designer. *Weak mitigation* is about the existence of good reward functions — ones that improve agent performance compared to using objective reward—however they are found. Past work on solving the ORP has largely focused on demonstrating and understanding weak mitigation. *Strong mitigation* takes into account the cost of finding good reward functions. Intuitively, strong mitigation is demonstrated when it benefits the agent designer to spend a portion of limited computational resources on a search for good reward functions, rather than spending all on a search for policies that optimize objective reward. Achieving strong mitigation is much more practically consequential than weak mitigation[1] and is the focus of this paper.

The contributions of this paper are threefold. 1) We formalize a new distinction between *weak mitigation* and *strong mitigation* of agent limitations. 2) We define a family of abstract architectures we call **N**ested **O**ptimal **R**eward and **C**ontrol or NORC that follows transparently from our def-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

[1]Achieving weak mitigation may also have practical benefits if the cost of the reward function search is amortized through extended use of the discovered reward function

inition of strong mitigation. 3) We provide concrete examples of NORC architectures and demonstrate empirically that they achieve strong mitigation in some simple domains crafted to illustrate the connection between forms of good reward and the types of agent limitations, as well as on Othello to show scaling and applicability of the NORC architecture.

Other Approaches to Designing Reward Functions. Inverse-RL [3] and preference elicitation [4] consider the problem of determining a reward function, in the former by observing behavior of a human and in the latter from answers to queries asked of a human. Both are different from ORP which assumes that the human's objective reward function is known to begin with. Reward shaping [5], the idea of designing reward functions that accelerate learning, was shown to be a special case of ORP [6] (they also showed the generality of ORP helps). In summary, while these and other approaches to designing rewards exist, prior work focuses on designing rewards from an understanding of the environment and designer's goals without taking into account limitations of the agent (see Definition 1 below).

A key distinctive feature of the ORP is that the agent's limitations are integral to defining rewards. This is a potential strength because it can lead to greater benefit from a good choice of rewards, but it is also a potential weakness because it makes the problem of finding good rewards harder. Previous work on ORP has demonstrated the strength by showing weak mitigation. In this paper we demonstrate that the weakness can be overcome by showing strong mitigation.

2. FROM WEAK TO STRONG MITIGATION

ORP and weak mitigation. The following notation allows us to define the ORP and weak mitigation. At time step t, the agent ($\mathcal{A}$) gets an observation o_t from the environment ($\mathbf{M}$) and takes an action u_t which causes a stochastic transition in the state of the environment. The agent's history at time k is $h_k = o_1 u_1 \ldots o_{k-1} u_{k-1} o_k$. Reward functions are mappings from agent histories to scalar rewards, and are used both to evaluate and guide behavior. Given a history h (of length $|h|$), the designer's or objective utility is $\mathcal{U}^{\mathcal{O}}(h) \stackrel{\text{def}}{=} \frac{1}{|h|}\sum_{t=1}^{|h|} R^{\mathcal{O}}(h_t)$ while the agent's or guidance utility is $\mathcal{U}^{A}(h) \stackrel{\text{def}}{=} \frac{1}{|h|}\sum_{t=1}^{|h|} R(h_t)$, where $R^{\mathcal{O}}$ is the objective reward function and R is the agent's reward function. The designer's objective is to build an agent whose behavior maximizes expected objective utility. The agent's objective is to behave so as to maximize expected guidance utility. Agent $\mathcal{A}$, when guided by reward function R is denoted $\mathcal{A}^R$, and when interacting with the environment produces history $h \sim \langle \mathcal{A}^R, \mathbf{M}\rangle$, where $\langle \mathcal{A}^R, \mathbf{M}\rangle$ is the distribution over histories. The expected objective utility obtained by the designer from the agent is $\mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R) \stackrel{\text{def}}{=} \mathbb{E}_{h\sim\langle \mathcal{A}^R,\mathbf{M}\rangle}[\mathcal{U}^{\mathcal{O}}(h)]$. The conventional RL agent is $\mathcal{A}^{R^{\mathcal{O}}}$ and achieves expected objective utility $\mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^{\mathcal{O}})$. Separating the agent's reward function from the designer's objective reward function leads to the following definition.

DEFINITION 1. ***(ORP [1])***
Given environment $\mathbf{M}$*, agent* $\mathcal{A}$*, and a set of reward functions* $\mathbf{R}$*, choose an optimal reward function*

$$R^* \stackrel{\text{def}}{=} \arg\max_{R\in\mathbf{R}} \mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R) \stackrel{\text{def}}{=} \arg\max_{R\in\mathbf{R}} \mathbb{E}_{h\sim\langle \mathcal{A}^R,\mathbf{M}\rangle}[\mathcal{U}^{\mathcal{O}}(h)].$$

Note that the set of optimal reward functions is a function of the triplet $\mathcal{A}, \mathbf{M}, R^{\mathcal{O}}$. In other words the limitations of the agent with respect to its environment play a role in defining the optimal reward; this is the formal departure from other (non-ORP) work on designing reward functions.

A number of results are available on ORP including: 1) If $R^{\mathcal{O}} \in \mathbf{R}$, then $\forall(\mathcal{A},\mathbf{M}), \mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^*) \geq \mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^{\mathcal{O}})$, i.e., using the optimal reward function to guide agent behavior cannot be detrimental to the designer's goals [1], 2) If $R^{\mathcal{O}} \in \mathbf{R}$ and agent $\mathcal{A}$ is unlimited in capability with respect to environment $\mathbf{M}$, then $\mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^*) = \mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^{\mathcal{O}})$, i.e., solving the ORP only helps limited agents [2], 3) a number of empirical and theoretical investigations that match agent limitations (e.g., depth-limits on planning) with classes of mitigating reward functions (e.g., exploration based rewards) point to possible prescriptions for good reward function choice [2], and 4) a gradient based algorithm PGRD for learning reward functions online for some planning agents [6]. These results demonstrate what we define as weak mitigation.

DEFINITION 2. ***(Weak Mitigation)***
If $\mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^*) > \mathcal{U}^{\mathcal{O}}_{(\mathcal{A},\mathbf{M})}(R^{\mathcal{O}})$*, then we say* R^* *weakly mitigates* $\mathcal{A}$*'s limitations in environment* $\mathbf{M}$ *for objective reward function* $R^{\mathcal{O}}$*.*

Crucially, weak mitigation demands only the existence of a good reward function for the $\mathcal{A}, \mathbf{M}, R^{\mathcal{O}}$ of interest. It demands nothing of how easy or hard it is to find good reward functions for it ignores that cost completely. We consider this cost next.

NORC and Strong Mitigation. Solving the ORP involves computation that may be better spent in the agent elsewhere (e.g., in doing deeper planning). Accounting for this cost of finding good rewards is this paper's main contribution, and is effectively a commitment to a new RL agent architecture in which there is both a conventional search for good value-functions/control-policies as well as a search for good reward functions. Because the goodness of a policy is only defined with respect to a reward function the search for good policies has to be nested inside the search for good reward functions. This leads to a two-level nested optimal reward and control (NORC) architecture that is depicted in the right panel of Figure 1. It splits the computational resources available to the designer between a critic-agent (denoted $\mathcal{C}$) that learns/plans good reward-function-policies to guide the actor-agent $\mathcal{A}$, and the conventional actor-agent that learns/plans good control-policies in the actor environment to achieve high reward from the critic-agent.

The departure in NORC is best understood by comparing it to the other two architectures in Figure 1. The conventional RL architecture (left panel) is an artificial agent that simply receives the objective reward from the environment and engages in a search for the best policy with respect to it. For the weak mitigation architecture (middle panel), the designer precomputes a good reward function outside the architecture and builds it into a critic inside the actor-agent (in effect "misleading" the artificial agent concerning the true nature of the objective reward in order to help overcome its limitations). Finally the new NORC agent (right panel), like the conventional agent, simply receives the objective reward from the environment, and then internally partitions the computation between search for a better reward function (the responsibility of the critic-agent) and the search for a good policy (the responsibility of the actor-agent).

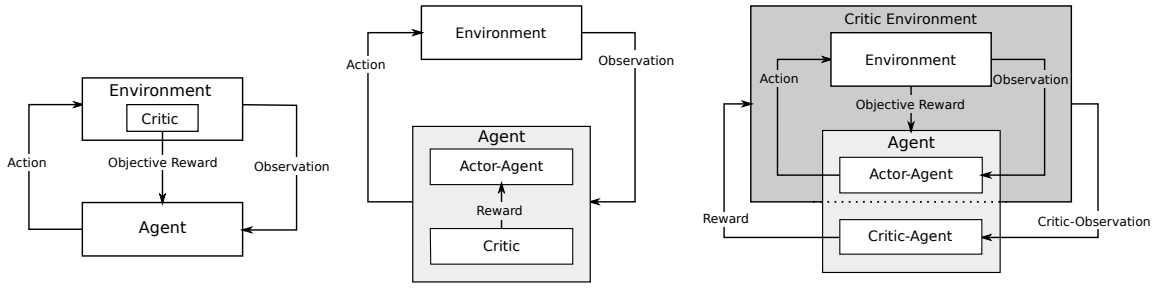


Figure 1: (Left) The conventional RL architecture's agent environment diagram. (Middle) The weak mitigation architecture (from Singh et al. [1]). (Right) The new strong mitigation NORC architecture introduced in this paper. See Section 2 for details.

Intuitively stated, strong mitigation holds when it is better for the resource-bound designer to use a NORC agent rather than a conventional agent, i.e., when learning rewards in addition to learning behavior is a good decomposition of limited resources within an agent. In other words, strong mitigation suggests that reward functions are another important locus of learning or adaptivity within a bounded agent. To formalize this consider a parameterized family of conventional agents, $\mathcal{A}(\Theta)$, where Θ is some set of parameters that impact resource consumption (e.g., one family of agents used in the experiments reported here are finite depth planning agents with depth as resource parameter; larger depth requires more CPU resource per decision). Given a critic-agent $\mathcal{C}$ (we present concrete instances in the next Section), we get a family of NORC agents $\{\mathcal{CA}(\Theta)\}$. Let $Res(\mathcal{A}(\theta \in \Theta))$ be the resource consumption of conventional agent $\mathcal{A}(\theta)$, and $Res(\mathcal{CA}(\theta \in \Theta))$ be the resource consumption of NORC agent $\mathcal{CA}(\theta)$. The expected objective utility obtained by the designer with resource bound τ via the NORC architecture is

$$\mathcal{U}^{\mathcal{O}}_{\mathbf{M}}(\mathcal{CA}(\Theta);\tau) = \underset{\theta\in\Theta \text{ s.t. } Res(\mathcal{CA}(\theta))\leq\tau}{\arg\max} \mathbb{E}_{h\sim\langle\mathcal{CA}(\theta),\mathbf{M}\rangle}[\mathcal{U}^{\mathcal{O}}(h)], \tag{1}$$

and similarly, the expected objective utility obtained via the conventional architecture is

$$\mathcal{U}^{\mathcal{O}}_{\mathbf{M}}(\mathcal{A}(\Theta);\tau) = \underset{\theta\in\Theta \text{ s.t. } Res(\mathcal{A}(\theta))\leq\tau}{\arg\max} \mathbb{E}_{h\sim\langle\mathcal{A}(\theta),\mathbf{M}\rangle}[\mathcal{U}^{\mathcal{O}}(h)]. \tag{2}$$

Definition 3. ***(Strong Mitigation)*** *If $\mathcal{U}^{\mathcal{O}}_{\mathbf{M}}(\mathcal{CA}(\Theta);\tau) > \mathcal{U}^{\mathcal{O}}_{\mathbf{M}}(\mathcal{A}(\Theta);\tau)$, then we say that the critic-agent $\mathcal{C}$ strongly mitigates the limitations of the family of agents $\mathcal{A}(\Theta)$ in environment $\mathbf{M}$ with objective reward function $R^{\mathcal{O}}$ for a resource bound τ.*

We emphasize that weak mitigation (cf. Definition 2) has expected objective utility as a function of reward functions, because that is what the designer chooses in the weak mitigation setting, while strong mitigation (cf. Definition 3) has expected utility as a function of agent architectures (via Equations 1 and 2) because that is what the designer chooses in the strong mitigation setting.

3. CRITIC-AGENTS FOR NORC

Here we turn to providing concrete NORC agents by developing critic-agents; for actor-agents we simply use existing families of popular resource parameterized learning and planning algorithms. The critic-agent's environment is composed of the actor's environment and the actor-agent and thus the state of the critic's environment is (s, s^a) where s is the environment's state and s^a is the actor-agent's state (this is the state of the actor-agent program). The critic-agent's observation consists of both the environment observation o, and the actor-agent's state s^a. The critic-agent's action-space is the set of reward functions $\mathbf{R}$. Recall that the critic-agent is guided by $R^{\mathcal{O}}$ in the NORC architecture while the actor-agent is guided by whatever reward function is provided to it by the critic-agent's action choice

Algorithm 1 General NORC Psuedocode

```
for (t = 1, 2, ...)
  Observe o_t from the environment
  C observes u_{t-1}, R_{t-1}, s^a_{t-1}, o_t, R^O(·_t)
  C chooses reward-function-action R_t ∈ R
    Set A's reward function to R_t
    A observes u_{t-1}, o_t
    A chooses action u_t
  Take action u_t in the environment
```

The NORC architecture involves two interacting exploration/exploitation problems. The first (and inner) is the conventional one faced by the RL actor-agent. The second (and outer) is the new one faced by the critic-agent of whether to stay with the policy for selecting reward functions for the actor-agent that is best based on current knowledge (exploit), or whether to choose reward functions for the actor-agent to improve current knowledge to allow greater objective reward in the future (explore). Unsurprisingly then, the critic-agent is *also* usefully thought of as an RL agent, only with reward functions ($\in \mathbf{R}$) as actions and the joint actor-agent and actor environment as critic environment. This makes it possible to leverage the considerable existing body of algorithms and analysis available for designing RL agents. We consider two different classes of exploration/exploitation problem formulations, the simpler bandits and the more complex MDPs/POMDPs, as the basis for critic-agent algorithms.

3.1 Bandit-based Critic-Agents

First we consider critic-agents that treat their exploration-exploitation problem as a multi-arm bandit problem, i.e., they ignore critic-observations and treat each reward-function-action ($R \in \mathbf{R}$) as an arm and the objective reward as the reward function. Pulling an arm, i.e., selecting a reward-function-action, yields an objective reward through the resulting action choice made by the actor-agent in the environment. In a standard bandit formulation, when an arm

is pulled the reward obtained is an unbiased estimate of the expected utility of that arm, and the choice of arm has no impact on the rewards sampled from other arms in the future. The critic-agent's bandit problem violates these assumptions and thus faces the following challenges: (1) choices of reward-function-action can affect the achievable objective utility for other reward-function-actions by transitioning the critic environment's state (2) samples of immediate objective reward obtained on selecting a reward-function-action are biased estimators for objective-utility for that reward-function-action, and (3) the bias in the objective reward samples can change over time due to dynamics of the actor-agent. We consider two broad classes of bandit critic-agents, corresponding to finite and infinite sets of reward functions.

Finite set of reward functions. When the set of reward functions $|\mathbf{R}|$ is finite (and for practical purposes small) there exist bandit (or experts) algorithms capable of dealing with the exploration-exploitation challenges of the critic-agent via one essential trick, namely that of holding the arm (reward-function-action) fixed for a period of time to alleviate bias and non-stationarity. Of course, how long to hold a reward-function-action fixed is unknown for it depends on the details of the actor-agent and the actor environment. There are several algorithms in the literature that adopt different schemes for incrementally searching for the right length while exploring and exploiting. The algorithm AtEase (*alternating trusted exploration and suspicious exploration*)[7] assumes finite but unknown ϵ-mixing-times, i.e, assumes that if any reward-function-action were held fixed for some unknown, but finite, amount of time the actual average objective reward obtained will be ϵ-close to the expected objective-utility of that reward-function-action choice. A second algorithm, EEE [8], makes no such mixing-time assumptions, and as a result offers weaker guarantees. We focus only on AtEase here.

We do not present the AtEase algorithm in detail here (see [7])), but we sketch a result for an AtEase-critic-based NORC architecture (denoted AtEase-$\mathcal{CA}(\theta)$) that follows directly from Theorem 1 in Talvitie & Singh [7].

Theorem 1. *For environment* $\mathbf{M}$ *and critic-agent reward-function-actions* $\mathbf{R}$*, let* $\mathcal{A}(\theta)$ *be such that NORC architecture* AtEase-$\mathcal{CA}(\theta)$ *satisfies any applicable resource-bounds and for which the finite* ϵ*-mixing-time assumption holds for every* $\mathcal{A}^{R\in\mathbf{R}}(\theta)$*. Then, with high probability* $(1-\delta)$ *for a number of actions* t *polynomial in:* ϵ*-mixing-time of* $\mathcal{A}^{R^*}(\theta)$ *acting in* $\mathbf{M}$*,* $1/\epsilon$*,* $1/\delta$ *and other parameters related to the size of* $\mathbf{M}$*, a history* $h \sim \langle$AtEase-$\mathcal{CA}(\theta), \mathbf{M}\rangle$ *of length* t*, will have objective-utility* $\mathcal{U}^{\mathcal{O}}(h)$ *that is* ϵ*-close to* $\mathcal{U}^{\mathcal{O}}_{\langle\mathcal{A}(\theta),\mathbf{M}\rangle}(R^*)$*.*

In words, if the finite-mixing-time assumption holds, the NORC architecture with an AtEase-critic-agent will, with high probability, and after a number of steps polynomial in the parameters defined above, achieve nearly the same objective-utility as agent $\mathcal{A}^{R^*}(\theta)$ (the weak-mitigation result with apriori-provided optimal reward function R^*). This also means, of course, that the NORC agent will compare favorably with the conventional architecture's agent $\mathcal{A}^{R^{\mathcal{O}}}(\theta)$ (since the conventional agent performs no better than $\mathcal{A}^{R^*}(\theta)$).

Infinite set of reward functions. When the space of reward-function-actions is infinite, the above bandits algorithms do not apply. However, it turns out that PGRD [6] (*policy gradient for reward design*), a recently introduced algorithm for finding good rewards by approximately solving the ORP in differentiably parameterized infinite reward spaces, can be interpreted as a critic-agent algorithm in the NORC architecture. It requires computing the gradient of the objective utility with respect to the reward-function-action parameters. This is only feasible if the actor-agent's procedural mapping from actor-histories to behavior policies (which in turn determine the objective utility) is differentiable. When this condition is satisfied PGRD can treat the actor-agent as a policy parameterized by the reward-function-action parameters and ascend the objective-utility gradient using a policy gradient algorithm. Sorg et al. developed approximate gradient computations for depth-limited planning and UCT[6]; we will use resulting PGRD-critic-agent algorithms for these two families of actor-agents in our experiments.

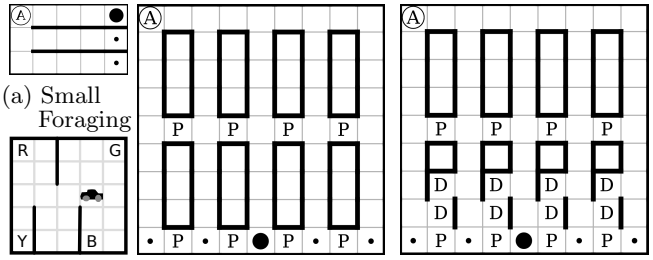
3.2 MDP-based Critic-Agents

The more general case concerns a critic-agent that learns a policy mapping critic-histories to reward-function-actions ($\in \mathbf{R}$). The strongest results in the RL literature for learning policies are in the context of MDPs. When is the critic-agent's exploration-exploitation problem an MDP? The state of the critic environment is given by (s, s^a), and since the state of the actor-agent is always observable, when the actor environment is fully observable, that is $o_t = s_t$, the critic environment is also fully observable (and therefore the critic-agent's problem is an MDP with reward function $R^{\mathcal{O}}$). However, if the actor-agent's dynamics are non-stationary, then the critic's problem will be a non-stationary MDP. For depth-limited planning and UCT with fixed and given models, the actor-agent's dynamics are stationary. In such settings, any algorithm for solving MDPs can be used as a critic-agent algorithm. In particular, simple RL algorithms such as ϵ-greedy Q-learning become applicable (and for finite-state critic environment and a finite set of reward-function-actions, are guaranteed to converge to the optimal reward-function-action policy). In cases where either the actor environment is not fully observable or the actor-agent's dynamics are non-stationary, more sophisticated RL algorithms can be used, but we do not explore those here.

4. EXPERIMENTS

The following set of experiments together are intended to show the robustness of the strong mitigation phenomenon across different families of actor-agents (depth-limited planning, the recent TDPlanning algorithm [9], and state-of-the-art UCT), across different kinds of actor-agent limitations (limited-depth, incorrect modeling assumptions, limited sampling and search control) as well as across different spaces of reward functions (those based on domain independent features such as inverse-recency and inverse-frequency, as well as those based on domain-dependent sub-goal features).

Common Structure in Experiments. For each experiment we will describe 1) the environment and objective reward function, 2) the family of resource-parameterized planning algorithms used both as conventional agents and as actor-agents for NORC, 3) the critic-agent algorithms used, and 4) the space of reward functions that determine the actions available to the critic-agents. In all experiments, the resource bounds we consider are limits on the *CPU-time per decision* and we compare the conventional agent with mul-



(a) Small Foraging (d) Taxi (b) Foraging (c) Darkroom Foraging

Figure 2: Illustration of environments. Thick black lines are impassable walls. In the foraging domains (a,b,d), small circles represent inexhaustible food, the large circle is one possible location for the consumable food, 'P' are pits, and 'D' are dark rooms.

tiple NORC agents for various limits on time per decision. Intuitively, in each experiment we expect a threshold on time per decision below which the NORC agent will outperform the conventional agent, demonstrating strong mitigation.

4.1 Learning in Bandit-Based Critic-Agents

Our first set of results are for critic-agents using the bandit algorithms described above (for finite sets of reward functions) and PGRD (for continuous reward functions).

4.1.1 Bounded planning depth

Here we contrast weak and strong mitigation for a family of actor-agents directly parameterized by planning depth (which indirectly parameterizes time per decision).

Environment and objective reward: An agent navigates a 5×3 grid (shown in figure 2a) choosing among actions: *North, South, East, West,* and *Eat*. Movements fail with probability 0.1 resulting in a random movement in any other direction. At the end of each corridor is an inexhaustible food source which when eaten gives objective reward $+0.01$. A second, consumable, food is at the end of one of the three corridors which when eaten provides $+1$ and is then replaced by another at the end of a different corridor.

Family of actor-agents: Agents employing full width, limited depth planning, with parameters $\Theta = \{d\}$ for planning depths $d \in \{3, \ldots, 11\}$ (which determine CPU-time per decision).

Critic-agents: Algorithms described in section 3: ATEASE, and PGRD[2]. We also included other bandit algorithms for comparison: EEE [8], EXP3 [10], and ϵ-greedy.

Reward functions: The reward functions were linear combinations of three domain-independent features: (1) inverse-recency (used e.g., by [2]) given by $\boldsymbol{\phi}_{\text{inv-rec}}(h) = 1 - 1/c(o_{|h|})$ where $c(o_{|h|})$ is the number of steps since observing the current observation $o_{|h|}$ (recall this domain is fully observable so this is the same as the number of steps since visiting the current state $s_{|h|}$). A reward function with a positive coefficient on this feature encourages the actor-agent to visit less-recently-visited states, resulting in persistent exploration. (2) inverse-frequency given by $\boldsymbol{\phi}_{\text{inv-freq}}(h) = 1/n(o_{|h|-1}, u_{|h|-1})$ where $n(o_{|h|}, u_{|h|})$ is the total number of times the agent has taken action $u_{|h|}$ after observing $o_{|h|}$. A positive coefficient on this feature encourages the actor-agent to visit less-frequently-visited transitions. (3) distance-to-goal heuristics, specifically we used Manhattan distance to the consumable food ($\boldsymbol{\phi}_{\text{dist}}(h)$). The reward function space is composed of linear functions:

$$R(h) = R^{\mathcal{O}}(h) + \theta_1 \boldsymbol{\phi}_{\text{inv-rec}}(h) + \theta_2 \boldsymbol{\phi}_{\text{inv-freq}}(h) + \theta_3 \boldsymbol{\phi}_{\text{dist}}(h)$$

The bandit algorithms used a coarse discretization of the parameter space with $(\theta_1, \theta_2, \theta_3) \in \{-1, -0.1, 0, 0.1, 1\}^3$ yielding a total of $|\mathbf{R}| = 5^3$ reward functions, while PGRD optimized over the continuous space $(\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3$.

[2]ATEASE used $\epsilon = \infty$ (so it is not suspicious when exploiting) and $l = 90$, PGRD used discount $\gamma = 0.99$, temperature 100, and learning rate $\alpha = 5^{-5}$ (see references for details).

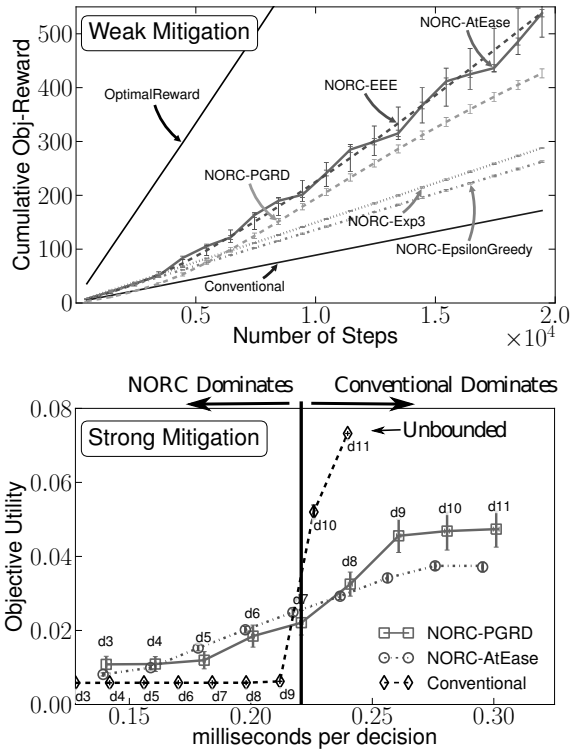


Figure 3: Experiments on small foraging domain; (top) compares single conventional agent — a depth 5 planner — with corresponding NORC agents. All NORC agents outperform the conventional agent showing weak mitigation, but not strong mitigation. (bottom) shows both objective utility and computational resources required for a family of actor-agents and corresponding NORC agents. For a resource bound less than that indicated by the vertical line, NORC agents outperform conventional agents, thus showing strong mitigation.

Results Figure 3 (top) shows performance as a function of number of steps for an actor-agent with planning depth 5. Unsurprisingly Agent $\mathcal{A}^{R^*}$ (denoted OptimalReward) did the best since it was given the optimal reward function for guidance from the start. In particular it did far better than the conventional agent that used the objective reward; this is a weak mitigation result because it ignores bounds on CPU-time. Interestingly, all NORC agents obtained greater cumulative objective reward than the conventional agent. This sets up the possibility of strong mitigation which we explore in Figure 3 (bottom) that shows objective-utility (over 20,000 steps) plotted against CPU-time per decision (τ) for three agents. For τ less than about $0.22ms$ the agent designer is better off choosing an agent with smaller plan-

ning depth using the NORC architecture than choosing a conventional agent with larger planning depth. This is the key strong mitigation result. As the resource bound gets looser ($\tau > 0.22ms$), choosing the conventional agent becomes better (in fact, at depth 11, the conventional agent acts optimally).

4.1.2 Bounded sample-based planning

This experiment shows strong mitigation for a second family of actor-agents using the popular and efficient Monte Carlo tree-search algorithm UCT [11].

Environment and objective reward Similar to Experiment 1, but larger, and with the addition of pits that transport the agent to a random location in the top row (see Figure 2b), eating the inexhaustible food yields objective reward +0.001.

Family of actor-agents: UCT with parameters $\Theta = \{(d, t)\}$ for planning depths $d \in \{5, 15, 35\}$ and trajectory counts $t \in \{50, 100, 500, 1000\}$; together these parameters determine CPU time per decision over 20,000 steps. UCT builds a search tree by simulating (from the current state) t trajectories of depth d where decisions in the trajectory generation are treated as a bandit problem solved with the UCB1 algorithm. We performed two experiments, one in which the actor-agent was given a perfect environment model and a second in which the actor-agent learned a model using the empirical probabilities observed in the data. To provide exploration, the model-learning actor-agents had a 0.1 probability of taking a random action.

Critic-agents: AtEase, and PGRD with the same parameters as in the previous experiment.

Reward functions: The form of the reward function was similar to the previous experiment, but the Manhattan distance heuristic feature was replaced by a model error feature $\boldsymbol{\phi}_{\text{model-error}}$ measuring inaccuracy of the agent's model for each transition (as described by Sorg et al. [2]). Reward functions with positive coefficients on the model-error feature encourage an agent to explore less-well-modeled transitions. AtEase used the same discretization of the parameter space as before.

Results: Results are in Figure 4a and 4d. When the model was given, for CPU-time per decision τ less than about $8ms$, choosing a NORC agent is better than a conventional agent. Again, showing strong mitigation, in this case for UCT-based actor-agents. When the model was learned, for the range of t, d parameters explored, the NORC agents dominate conventional agents in part because the reward functions provided by the critic-agent improved model-learning speed through rewarding persistent exploration (this is evident in that for the largest CPU-time per decision, the NORC agents achieved objective-utility much closer to their given-model counterparts than did the conventional-agent).

4.1.3 Incorrect model representation

In this experiment we show strong mitigation for UCT-based actor-agents with an incorrect modeling assumption in the model (both learned and given).

Environment and objective reward Like previous experiment except the environment is partially observable: between each corridor is a 2-square dark room through which the agent can move as normal, but cannot distinguish its location. See Figure 2c.

Actor-agents, critic-agents, rewards Like previous experiment; both the given and learned models made the first-order Markov assumption which is incorrect due to the dark-room. Also, notice negative coefficients on the model error feature encourage agent to avoid the dark room transitions.

Results: Results are in Figure 4b and 4e. When the model was given, the PGRD-NORC agents performed significantly better than the conventional agent for $\tau < 2ms$ CPU-time per decision. Evidence of strong mitigation again, in this case with the additional limitation of incorrect modeling assumptions in the face of partial-observability. For the second experiment when the actor-agent learned a first-order Markov model, the NORC agents dominated conventional agents for all parameters tested (as in the learned model instance of the previous Experiment for similar reasons).

4.2 Learning in MDP-based Critic-Agents

Here we depart from previous experiments with bandit-based critic-agents to explore MDP-based critic-agents that learn reward-function-action policies conditioned on abstractions of critic-histories.

Environment and objective reward: The taxi domain (seen in Figure 2d) was introduced by Dietterich [12]. The agent controls a taxi tasked with delivering a passenger from one of four pickup locations (labeled R,Y,G,B; random on each episode) to one of four (also random) destinations. There are six actions: *North, South, East, West, Pick Up Passenger,* and *Drop Off Passenger*. Objective reward gives +1 if passenger is dropped off at the correct destination (in which case the episode ends), -0.5 if dropped off incorrectly, and -0.05 on other transitions. State is fully observable and factored into three features $s = (\boldsymbol{\phi}_{taxi}, \boldsymbol{\phi}_{pssngr}, \boldsymbol{\phi}_{dest})$.

Family of actor-agents: Planning agents using TDPlanning [9] which is similar to UCT, but rather than building a search tree, it learns a value function[3] through t simulated trajectories of maximum length d. Depths and transition count parameters Θ were identical to the UCT experiments above. We also performed the same experiment using UCT actor-agents with similar results (not shown).

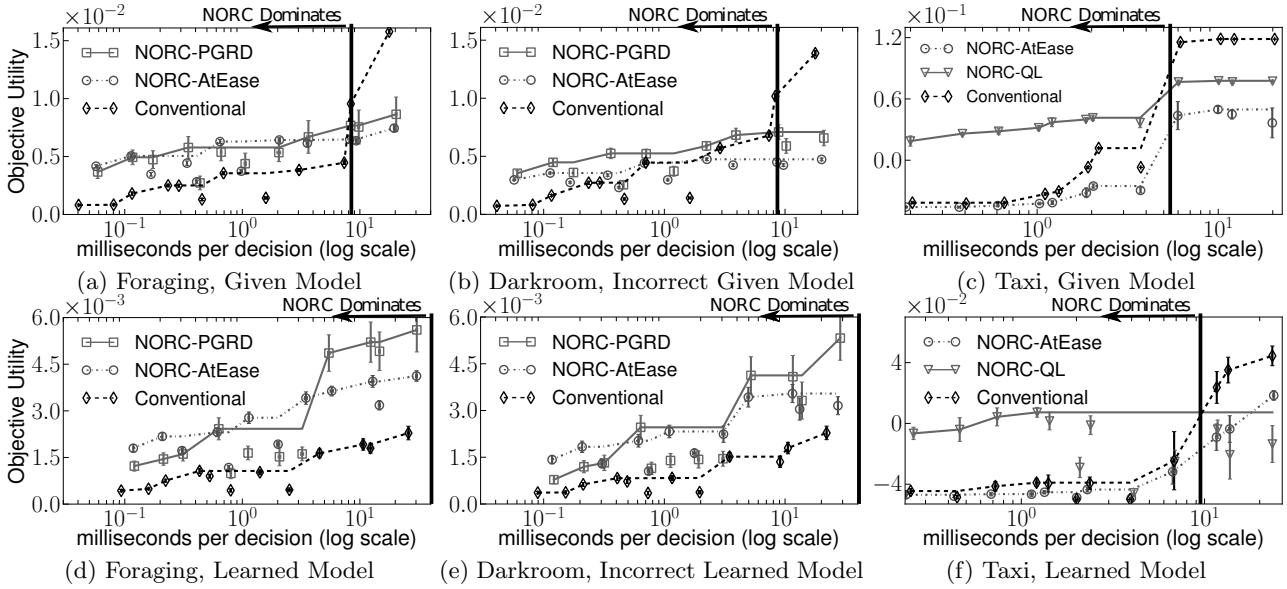
Critic-agents: ϵ-greedy Q-Learning[4] (actor-agent had no persistent state so critic environment state was actor environment state), and AtEase with parameters before.

Reward functions: Each was a single reward feature multiplied by a scalar for the inverse-recency feature $\boldsymbol{\phi}_{\text{inv-rec}}$ and state features $\boldsymbol{\phi}_{taxi}$ and $\boldsymbol{\phi}_{pssngr}$. Let δ be the Kronecker delta function, then we have one reward functions for each taxi location: $\delta(\boldsymbol{\phi}_{taxi}, l) \forall l \in \{1, \cdots, 25\}$, each passenger location $\delta(\boldsymbol{\phi}_{pssngr}, p) \forall p \in \{1, 2, 3, 4, 5\}$, for a discrete set of weights on the inverse-recency $\theta \boldsymbol{\phi}_{\text{inv-rec}} \forall \theta \in \{-1, -.1, 0, .1, 1\}$, and the objective reward function $R^{\mathcal{O}}$ totaling 36 reward functions (actions for the critic-agent).

Results: Results are in Figure 4c and 4f. The critic-agent using Q-Learning successfully learned a policy over reward functions allowing the NORC-QL agent to perform much better than the conventional agents for CPU-time limitations τ less than about $4ms$. However, NORC-AtEase where the critic-agent learned an unconditional reward policy, did not perform well because no single reward function in the set is particularly useful unless chosen as a function of state. When the actor-agent learns a model, the NORC-QL

[3] In our TDPlanning algorithm, the value function was learned with ϵ-greedy Q-Learning, $\epsilon = 0.1$, learning rate $\alpha = 0.4$

[4] $\epsilon = 0.1$ and learning rate of $\alpha = 0.9$



(a) Foraging, Given Model (b) Darkroom, Incorrect Given Model (c) Taxi, Given Model

(d) Foraging, Learned Model (e) Darkroom, Incorrect Learned Model (f) Taxi, Learned Model

In the graphs above, the points for each agent read from left to right correspond to actor-agent parameters:
(depth, trajectories): (5,50),(5,100),(15,50),(15,100), (35,50),(5,500),(35,100),(5,1000),(15,500),(15,1000),(35,500),(35,1000)

Figure 4: (left column) strong mitigation with bounded sample-based planning actor-agents (UCT) and critic-agents learned unconditional selection of reward functions; (center column) same as experiment in left column, but actor-agents had incorrect modeling assumption; (rightmost) strong mitigation in Taxi where the NORC-QL critic learned a policy over reward functions (NORC-AtEase did not, and as a result performed poorly in this experiment). The line drawn for each agent shows the highest score achieved up to a given resource level (this is a visual aid approximating the best performance per cpu time across parameters).

agent does not perform well, this may be because the non-stationarity of the actor-agent misleads the QL critic-agent.

These results show strong mitigation when the critic-agent learns a policy over reward-function-actions. In this experiment, the critic-agent essentially chose subgoals that could be achieved by an actor-agent using far less computational resources than necessary to act well given only the objective reward.

5. STRONG MITIGATION IN OTHELLO

Our final results show the practical import of NORC and strong mitigation on the board game of Othello, a large domain with a long history in AI.

Environment and objective reward: We experiment using both a 6×6 board and the standard 8×8 board. The 8×8 game has roughly 10^{28} states and a large branching factor determined by the number of legal moves at each step. For our experiments, the agent played against a fixed opponent agent planning with conventional UCT (depth 20 and trajectory count 100). The objective reward was $+1$ for a win, 0.5 for a draw, and 0 for a loss. The agents were evaluated and learned while playing against an opponent rather than learning with self play (a common paradigm in computer game playing).

Family of actor-agents: UCT with parameters given by the 20 combinations of depths $\{5, 10, 15, 20, 25\}$ and trajectory limits $\{50, 100, 250, 500\}$. We did not provide the agent with an opponent model, instead UCT generated a minimax planning tree by choosing actions for both black and white players as described by Gelly and Silver [13] for their master level computer GO player.

Critic-agent: PGRD-UCT with learning rate 10^{-7} and temperature parameter 100.

Reward functions: We used the weighted piece counter (wpc) [14] feature representation where $\boldsymbol{\phi}_{wpc}(o)$ consists of 64 features, one for each location on the board, with value $+1$ if that location contains a black piece 0 if empty and -1 if white. Reward functions were linear combinations of these features evaluated on the state reached after taking an action: $R(h) = R(o, a, o') = \theta^T \boldsymbol{\phi}_{wpc}(o')$.

Results: Each NORC and conventional agent was evaluated over 12 trials of 30,000 games; for half of these trials our agent plays white and for the other half black. Results are shown in Figure 5. It is immediately clear that the NORC agents required much more computation than the corresponding conventional agents (with same (d, t) parameters). For example, with depth 25 and trajectory count 500 the NORC agent required nearly twice the time per decision as the corresponding conventional agent. Nevertheless, we see strong mitigation: the reward functions learned by the critic-agent allow the UCT actor-agent to achieve higher quality planning with less computation, so given equivalent CPU time, a designer is better off choosing a NORC agent. Learned reward functions can improve UCT in a variety of ways such as providing better terminal evaluation and by improving search control via encouraging exploration of more

[5]For each agent, reading points from left to right on the graph, the parameter pairs are $(depth, trajectories)$:(5,50), (10,50), (5,100), (15,50), (20,50), (25,50), (10,100), (15,100), (20,100), (5,250), (25,100), (10,250), (15,250), (5,500),(20,250), (25,250), (10,500), (15,500), (20,500), (25,500)

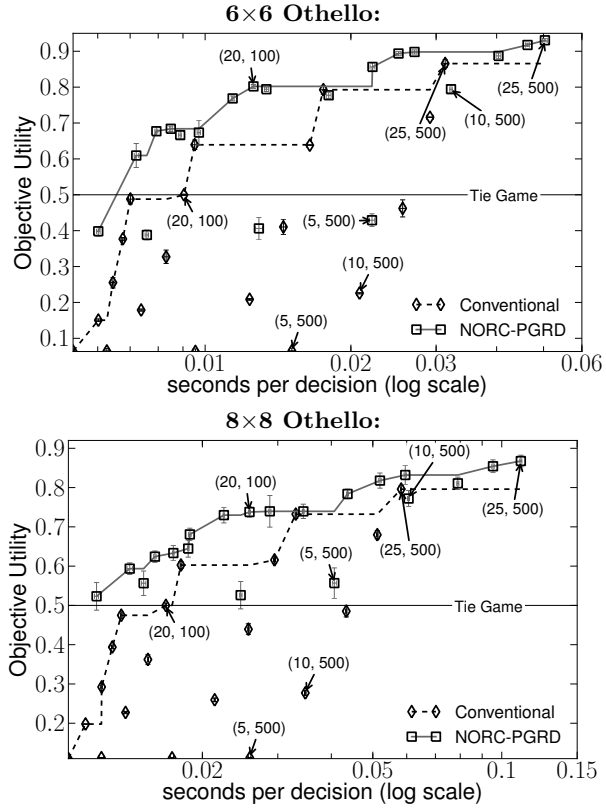


Figure 5: Othello results. NORC agents required significantly more computation than the corresponding conventional agents, but allowed the bounded UCT planning actor-agents to choose better actions even with less computational time (fewer and shallower sample trajectories). Several parameter pairs $(depth, trajectories)$ **are labeled[5].**

fruitful trajectories in planning (as discussed by Sorg et. al. [6]). Furthermore, in these experiments the NORC agents are more robust across the space of parameters.

6. CONCLUSION

The previously defined optimal reward problem (ORP) takes the agent's limitations expressed in its environment into account when designing rewards; this was in contrast to other approaches for designing rewards. Building on ORP this paper distinguishes between weak mitigation, the subject of previous ORP work, and the more practical strong mitigation developed here. We showed how strong mitigation implies a nested optimal reward and control (NORC) architecture and developed concrete NORC instantiations that allowed us to demonstrate over a variety of actor-agent planning algorithms that it can be better to nest the search for good policies inside a search for good reward functions. Our strong mitigation results on Othello demonstrate the scalability of the NORC architecture. Finally, our results with NORC suggest it might be beneficial to treat reward functions as a locus of learning and adaptivity within an autonomous agent – just as it might be beneficial to learn value functions or policy functions.

Acknowledgments.

This work was supported by NSF grants IIS-0905146 and IIS-1148668. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

7. REFERENCES

[1] Satinder Singh, Richard L. Lewis, Andrew G. Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2010.

[2] Jonathan Sorg, Satinder Singh, and Richard Lewis. Internal rewards mitigate agent boundedness. In *Proceedings of the International Conference on Machine Learning*, 2010.

[3] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2000.

[4] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *In Proceedings of the National Conference on Artificial Intelligence*, 2000.

[5] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, 1999.

[6] Jonathan Sorg, Satinder Singh, and Richard Lewis. Optimal rewards versus leaf-evaluation heuristics in planning agents. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*. 2011.

[7] Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceeding of the International Joint Conference on Artificial Intelligence*, 2007.

[8] Daniela Pucci De Farias and Nimrod Megiddo. Combining expert advice in reactive environments. *Journal of the ACM*, 2006.

[9] David Silver, Richard S. Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[10] Peter Auer, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of the Symposium on Foundations of Computer Science*, 1995.

[11] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, 2006.

[12] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 2000.

[13] Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the Conference on Artificial Intelligence*, 2008.

[14] P. Hingston and M. Masek. Experiments with Monte Carlo Othello. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007.