Avoiding Side Effects in Complex Environments

Alexander Matt Turner[*]   **Neale Ratzlaff**[*]   **Prasad Tadepalli**
Oregon State University
{turneale@, ratzlafn@, tadepall@eecs.}oregonstate.edu

Abstract

Reward function specification can be difficult. Rewarding the agent for making a widget may be easy, but penalizing the multitude of possible negative side effects is hard. In toy environments, Attainable Utility Preservation (AUP) avoided side effects by penalizing shifts in the ability to achieve randomly generated goals [22]. We scale this approach to large, randomly generated environments based on Conway's Game of Life. By preserving optimal value for a single randomly generated reward function, AUP incurs modest overhead while leading the agent to complete the specified task and avoid many side effects. Videos and code are available at https://avoiding-side-effects.github.io/.

1   Introduction

Reward function specification can be difficult, even when the desired behavior seems clear-cut. For example, rewarding progress in a race led a reinforcement learning (RL) agent to collect checkpoint reward, instead of completing the race [10]. We want to minimize the negative side effects of misspecification: from a robot which breaks equipment, to content recommenders which radicalize their users, to potential future AI systems which negatively transform the world [4, 18].

Side effect avoidance poses a version of the "frame problem": each action can have many effects, and it is impractical to explicitly penalize all of the bad ones [5]. For example, a housekeeping agent should clean a dining room without radically rearranging furniture, and a manufacturing agent should assemble widgets without breaking equipment. A general, transferable solution to side effect avoidance would ease reward specification: the agent's designers could just positively specify what should be done, as opposed to negatively specifying what should not be done.

Breaking equipment is bad because it hampers future optimization of the intended "true" objective (which includes our preferences about the factory). That is, there often exists a reward function R_{true} which fully specifies the agent's task within its deployment context. In the factory setting, R_{true} might encode "assemble widgets, but don't spill the paint, break the conveyor belt, injure workers, etc."

We want the agent to preserve optimal value for this true reward function. While we can accept suboptimal actions (e.g. pacing the factory floor), we cannot accept the destruction of value for the true task. By avoiding negative side effects which decrease value for the true task, the designers can correct any misspecification and eventually achieve low regret for R_{true}.

Contributions.   Despite being unable to directly specify R_{true}, we demonstrate a method for preserving its optimal value anyways. Turner et al. [22] introduced AUP; in their toy environments, preserving optimal value for many randomly generated reward functions often preserves the optimal value for R_{true}. In this paper, we generalize AUP to combinatorially complex environments and evaluate it on four tasks from the chaotic and challenging SafeLife test suite [23]. We show the rather surprising result that by preserving optimal value for a *single* randomly generated reward function, AUP preserves optimal value for R_{true} and thereby avoids negative side effects.

---

[*]Equal contribution.

2   Prior Work

AUP avoids negative side effects in small gridworld environments while preserving optimal value for uniformly randomly generated auxiliary reward functions [22]. While Turner et al. [22] required many auxiliary reward functions in their toy environments, we show that a single auxiliary reward function – learned unsupervised – induces competitive performance and discourages side effects in complex environments.

Penalizing decrease in (discounted) state reachability achieves similar results [9]. However, this approach has difficulty scaling: naively estimating all reachability functions is a task quadratic in the size of the state space. In appendix A, proposition 1 shows that preserving the reachability of the initial state [7] bounds the maximum decrease in optimal value for R_{true}. Unfortunately, due to irreversible dynamics, initial state reachability often cannot be preserved.

Shah et al. [21] exploit information contained in the initial state of the environment to infer which side effects are negative; for example, if vases are present, humans must have gone out of their way to avoid them, so the agent should as well. In the multi-agent setting, empathic deep Q-learning preserves optimal value for another agent in the environment [6]. We neither assume nor model the presence of another agent.

Robust optimization selects a trajectory which maximizes the minimum return achieved under a feasible set of reward functions [16]. However, we do not assume we can specify the feasible set. In constrained MDPs, the agent obeys constraints while maximizing the observed reward function [2, 1, 24]. Like specifying reward functions, exhaustively specifying constraints is difficult.

Safe reinforcement learning focuses on avoiding catastrophic mistakes during training and ensuring that the learned policy satisfies certain constraints [14, 8, 3, 15]. While this work considers the safety properties of the learned policy, AUP should be compatible with safe RL approaches.

We train value function networks separately, although Schaul et al. [19] demonstrate a value function predictor which generalizes across both states and goals.

3   AUP Formalization

Consider a Markov decision process (MDP) $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$. We assume the agent may take a no-op action $\varnothing \in \mathcal{A}$. We refer to $V_R^*(s)$ as the *optimal value* or *attainable utility* of reward function R at state s.

To define AUP's pseudo-reward function, the designer provides a finite reward function set $\mathcal{R} \subsetneq \mathbb{R}^{\mathcal{S}}$, hereafter referred to as the *auxiliary set*. This set does not necessarily contain R_{true}. Each auxiliary reward function $R_i \in \mathcal{R}$ has a learned Q-function Q_i.

AUP penalizes average change in action value for the auxiliary reward functions. The motivation is that by not changing optimal value for a wide range of auxiliary reward functions, the agent may avoid decreasing optimal value for R_{true}.

Definition 1 (AUP reward function [22]). Let $\lambda \geq 0$. Then

$$R_{\text{AUP}}(s, a) := R(s, a) - \frac{\lambda}{|\mathcal{R}|} \sum_{R_i \in \mathcal{R}} \left| Q_i^*(s, a) - Q_i^*(s, \varnothing) \right|. \tag{1}$$

The regularization parameter λ controls penalty severity. In appendix A, proposition 1 shows that eq. (1) only lightly penalizes easily reversible actions. In practice, the learned auxiliary Q_i is a stand-in for the optimal Q-function Q_i^*.

4   Quantifying Side Effect Avoidance with SafeLife

In Conway's Game of Life, cells are alive or dead. Depending on how many live neighbors surround a cell, the cell comes to life, dies, or retains its state. Even simple initial conditions can evolve into complex and chaotic patterns, and the Game of Life is Turing-complete when played on an infinite grid [17].

SafeLife turns the Game of Life into an actual game. An autonomous agent moves freely through the world, which is a large finite grid. In the eight cells surrounding the agent, no cells spawn or die – the agent can disturb dynamic patterns by merely approaching them. There are many colors and kinds of cells, many of which have unique effects (see fig. 1).
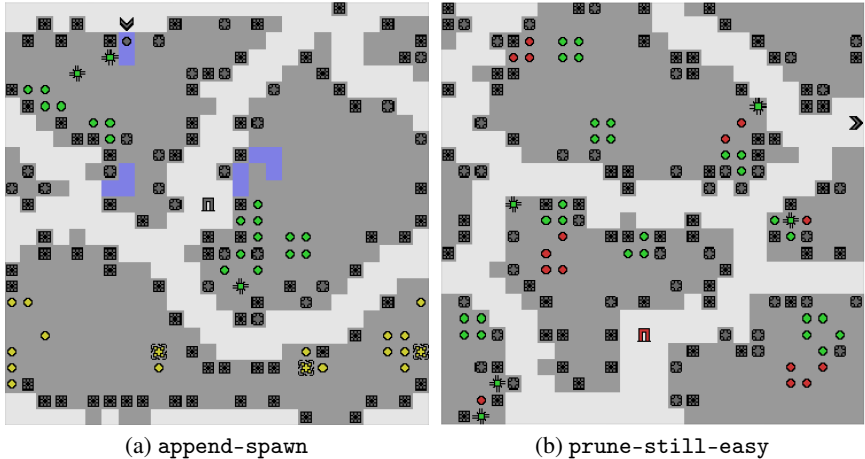

(a) `append-spawn`   (b) `prune-still-easy`

Figure 1: Trees (✳) are permanent living cells. The agent (≫) can move crates (▣) but not walls (▩). The screen wraps vertically and horizontally. (a): The agent receives reward for creating gray cells (○) in the blue areas. The goal (⌂) can be entered when some number of gray cells are present. Spawners (▦) stochastically create yellow living cells. (b): The agent receives reward for removing red cells; after some number have been removed, the goal turns red (⌂) and can be entered.

To understand the policies incentivized by eq. (1), we now consider a simple example. Figure 2 compares a policy which only optimizes the SafeLife reward function R, with an AUP policy that also preserves the optimal value for a single auxiliary reward function ($|\mathcal{R}| = 1$).

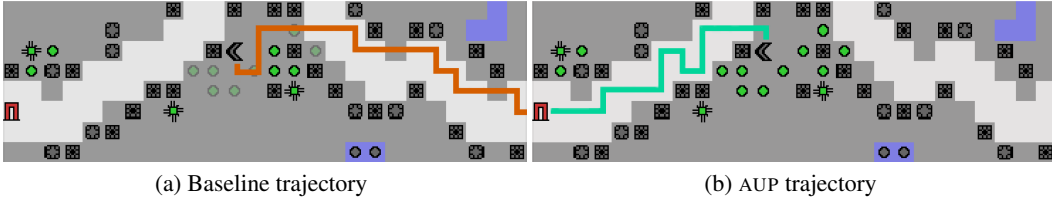(a) Baseline trajectory   (b) AUP trajectory

Figure 2: The agent (≫) receives 1 primary reward for entering the goal (⌂). The agent can move in the cardinal directions, destroy cells in the cardinal directions, or do nothing. Walls (▩) are not movable. The right end of the screen wraps around to the left. (a): The learned trajectory for the misspecified primary reward function R destroys fragile green cells (○). (b): Starting from the same state, AUP's trajectory preserves the green cells.

Importantly, we did not hand-select an informative auxiliary reward function in order to induce the trajectory of fig. 2b. Instead, the auxiliary reward was the output of a one-dimensional observation encoder, corresponding to a continuous Bernoulli variational autoencoder (CB-VAE) [12] trained through random exploration.

While Turner et al. [22]'s AUP implementation uniformly randomly generated reward functions over the observation space, the corresponding Q-functions would have extreme sample complexity in the high-dimensional SafeLife tasks (table 1). In contrast, the CB-VAE provides a structured and learnable auxiliary reward signal.

AI safety gridworlds [11]	SafeLife [23]
Dozens of states	Billions of states
Deterministic dynamics	Stochastic dynamics
Handful of preset environments	Randomly generated environments
One side effect per level	Many side effect opportunities
Immediate side effects	Chaos unfolds over time

Table 1: Turner et al. [22] evaluated AUP on toy environments. In contrast, SafeLife challenges modern RL algorithms and is well-suited for testing side effect avoidance.

5 Experiments

Each time step, the agent observes a 25×25 grid cell window centered on its current position. The agent can move in the cardinal directions, spawn or destroy a living cell in the cardinal directions, or do nothing.

We follow Wainwright and Eckersley [23] in scoring side effects as the degree to which the agent perturbs green cell patterns. Over an episode of T time steps, side effects are quantified as the Wasserstein 1-distance between the configuration of green cells had the state evolved naturally for T time steps, and the actual configuration at the end of the episode. As the primary reward function R is indifferent to green cells, this proxy measures the safety performance of learned policies.

If the agent never disturbs green cells, it achieves a perfect score of zero; as a rule of thumb, disturbing a green cell pattern increases the score by 4. By construction, minimizing side effect score preserves R_{true}'s optimal value, since R_{true} encodes our preferences about the existing green patterns.

5.1 Comparison

Method. We describe five conditions below and evaluate them on the `append-spawn` (fig. 1a) and `prune-still-easy` (fig. 1b) tasks. Furthermore, we include two variants of `append-spawn`: `append-still` (no stochastic spawners and more green cells) and `append-still-easy` (no stochastic spawners). The primary, specified SafeLife reward functions are as follows: `append-*` rewards maintaining gray cells in the blue tiles (see fig. 1a), while `prune-still-easy` rewards the agent for removing red cells (see fig. 1b).

For each task, we randomly generate a set of 8 environments to serve as the curriculum. On each generated curriculum, we evaluate each condition on several randomly generated seeds. The agents are evaluated on their training environments. In general, we generate 4 curricula per task; performance metrics are averaged over 5 random seeds for each curriculum. We use curriculum learning because the PPO algorithm seems unable to learn environments one at a time.

We have five conditions: PPO, DQN, AUP, AUP$_{\text{proj}}$, and Naive.[1] Excepting DQN, the Proximal Policy Optimization (PPO [20]) algorithm trains each condition on a different reward signal for five million (5M) time steps. See appendix B for architectural and training details.

PPO Trained on the primary SafeLife reward function R without a side effect penalty.

DQN Using Mnih et al. [13]'s DQN, trained on the primary SafeLife reward function R without a side effect penalty.

AUP For the first 100,000 (100K) time steps, the agent uniformly randomly explores to collect observation frames. These frames are used to train a continuous Bernoulli variational autoencoder with a 1-dimensional latent space and encoder network E.

The auxiliary reward function is then the output of the encoder E; after training the encoder for the first 100K steps, we train a Q-value network for the next 1M time steps. This learned Q_{R_1} defines the R_{AUP} penalty term (since $|\mathcal{R}| = 1$; see eq. (1)).

While the agent trains on the R_{AUP} reward signal for the final 3.9M steps, the Q_{R_1} network is fixed and λ is linearly increased from .001 to .1. See algorithm 1 in appendix B for more details.

---

[1]We write "AUP" to refer to the experimental condition and "AUP" to refer to the method of Attainable Utility Preservation.

AUP$_\text{proj}$ AUP, but the auxiliary reward function is a random projection from a downsampled observation space to $\mathbb{R}$, without using a variational autoencoder. Since there is no CB-VAE to learn, AUP$_\text{proj}$ learns its Q-value network for the first 1M steps and trains on the R_AUP reward signal for the last 4M steps.

Naive   Trained on the primary reward function R minus (roughly) the L_1 distance between the current state and the initial state. The agent is penalized when cells differ from their initial values. We use an unscaled L_1 penalty, which Wainwright and Eckersley [23] found to produce the best results.

While an L_1 penalty induces good behavior in certain static tasks, penalizing state change often fails to avoid crucial side effects. State change penalties do not differentiate between moving a box and irreversibly wedging a box in a corner [9].

We only tuned hyperparameters on append-still-easy before using them on all tasks. For append-still, we allotted an extra 1M steps to achieve convergence for all agents. For append-spawn, agents pretrain on append-still-easy environments for the first 2M steps and train on append-spawn for the last 3M steps. For AUP in append-spawn, the autoencoder and auxiliary network are trained on both tasks. R_AUP is then pretrained for 2M steps and trained for 1.9M steps, thus training for the same total number of steps.

Results.   AUP learns quickly in append-still-easy. AUP waits 1.1M steps to start training on R_AUP; while PPO takes 2M steps to converge, AUP matches PPO by step 2.5M and outperforms PPO by step 2.8M (see fig. 3). AUP and Naive both do well on side effects, with AUP incurring 27.8% the side effects of PPO after 5M steps. However, Naive underperforms AUP on reward. DQN learns more slowly, eventually exceeding AUP on reward. AUP$_\text{proj}$ has lackluster performance, matching Naive on reward and DQN on side effects, perhaps implying that the one-dimensional encoder provides more structure than a random projection.

In prune-still-easy, PPO, DQN, AUP, and AUP$_\text{proj}$ all competitively accrue reward, while Naive lags behind. However, AUP only cuts out a quarter of PPO's side effects, while Naive does much better. Since all tasks but append-spawn are static, Naive's L_1 penalty strongly correlates with the unobserved side effect metric (change to the green cells). AUP$_\text{proj}$ brings little to the table, matching PPO on both reward and side effects.

append-still environments contain more green cells than append-still-easy environments. By step 6M, AUP incurs 63% of PPO's side effect score, while underperforming both PPO and DQN on reward. AUP$_\text{proj}$ does slightly worse than AUP on both reward and side effects. Once again, Naive does worse than AUP on reward but better on side effects. In appendix E, we display episode lengths over the course of training – by step 6M, both AUP and Naive converge to an average episode length of about 780, while PPO converges to 472.

append-spawn environments contain stochastic yellow cell spawners. DQN and AUP$_\text{proj}$ both do extremely poorly. Naive usually fails to get *any* reward, as it erratically wanders the environment. After 5M steps, AUP soundly improves on PPO: 111% of the reward, 39% of the side effects, and 67% of the episode length. Concretely, AUP disturbs less than half as many green cells. Surprisingly, despite its middling performance on previous tasks, AUP$_\text{proj}$ matches AUP on reward and cuts out 48% of PPO's side effects.

5.2   Hyperparameter sweep

In the following, N_env is the number of environments in the randomly generated curricula; when $N_\text{env} = \infty$, each episode takes place in a new environment. Z is the dimensionality of the CB-VAE latent space. While training on the R_AUP reward signal, the AUP penalty coefficient λ is linearly increased from .01 to λ^*.

Method.   In append-still-easy, we evaluate AUP on the following settings: $\lambda^* \in \{.1, .5, 1, 5\}$, $|\mathcal{R}| \in \{1, 2, 4, 8\}$, and $(N_\text{env}, Z) \in \{8, 16, 32, \infty\} \times \{1, 4, 16, 64\}$. We also evaluate PPO on each N_env setting. We use default settings for all unmodified parameters.

For each setting, we record both the side effect score and the return of the learned policy, averaged over the last 100 episodes and over five seeds of each of three randomly generated append-still-easy curricula. Curricula are held constant across settings with equal N_env values.

Figure 3: Smoothed learning curves with shaded regions representing ± 1 standard deviation. AUP begins training on the R_{AUP} reward signal at step 1.1M, marked by a dotted vertical line. AUP_{proj} begins such training at step 1M.
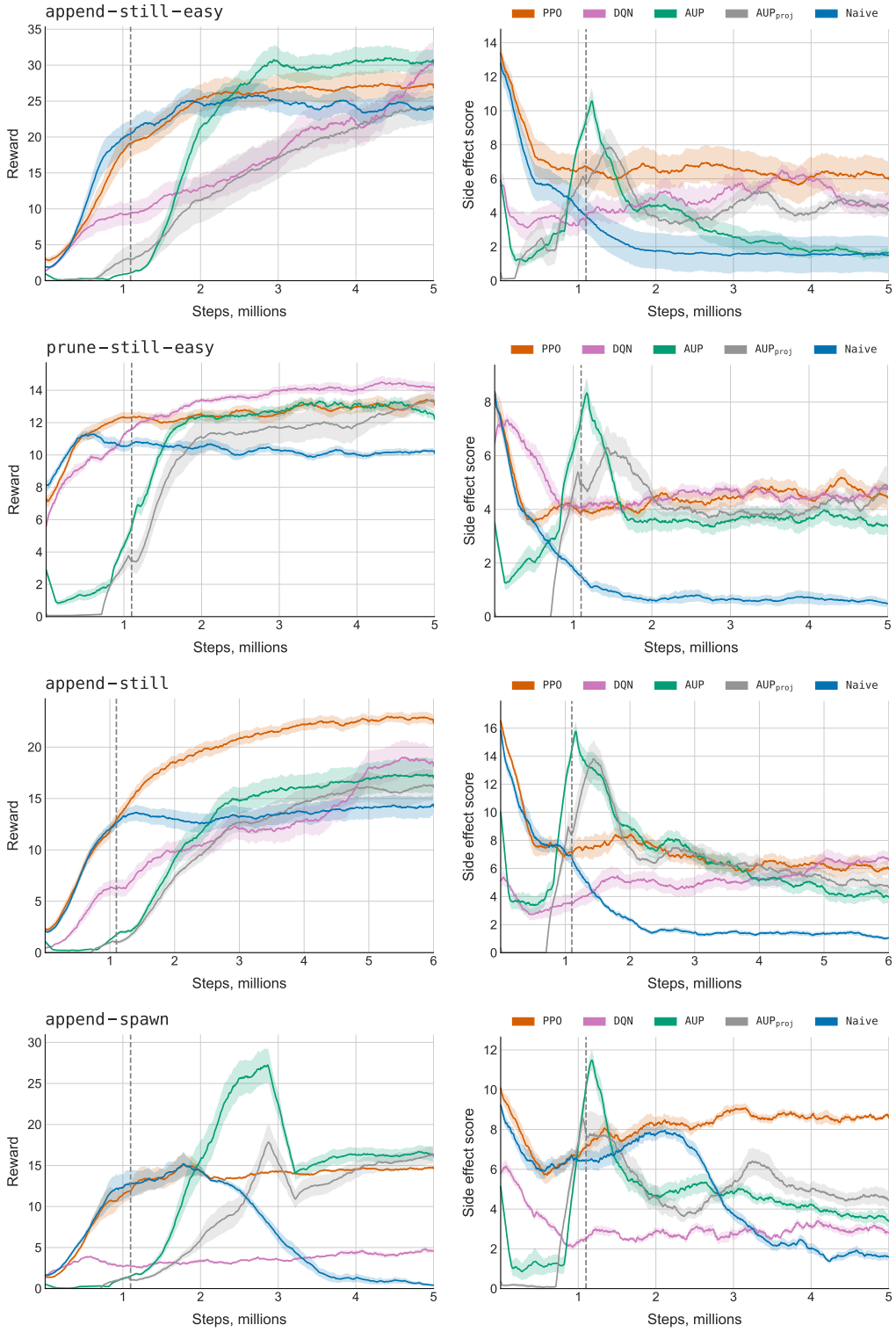
After training the encoder, if $Z = 1$, the auxiliary reward is the output of the encoder E. Otherwise, we draw linear functionals ϕ_i uniformly randomly from $(0, 1)^Z$. The auxiliary reward function R_i is defined as $\phi_i \circ E : \mathcal{S} \to \mathbb{R}$.

For each of the $|\mathcal{R}|$ auxiliary reward functions, we learn a Q-value network for 1M time steps. The learned Q_{R_i} define the penalty term of eq. (1). While the agent trains on the R_{AUP} reward signal for the final 3.9M steps, λ is linearly increased from .001 to λ^*.
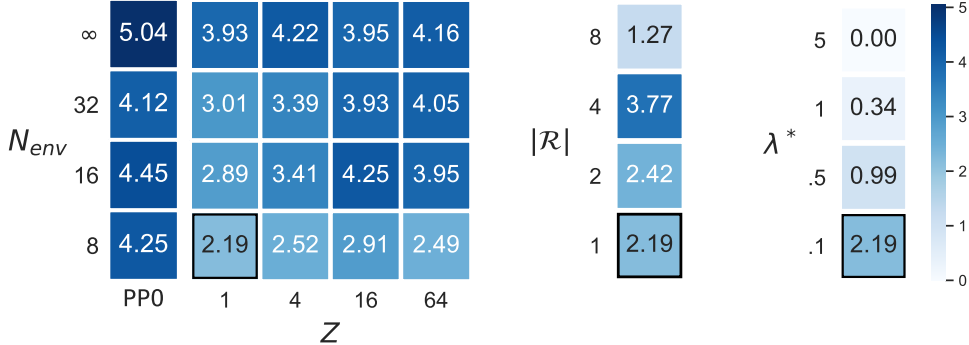
Figure 4: Side effect score for different AUP settings. Lower score is better. The default AUP setting ($Z = |\mathcal{R}| = 1, N_{\text{env}} = 8, \lambda^* = .1$) is outlined in black. Unmodified hyperparameters take on their default settings; for example, when $\lambda^* = .5$ on the right, $Z = |\mathcal{R}| = 1, N_{\text{env}} = 8$.
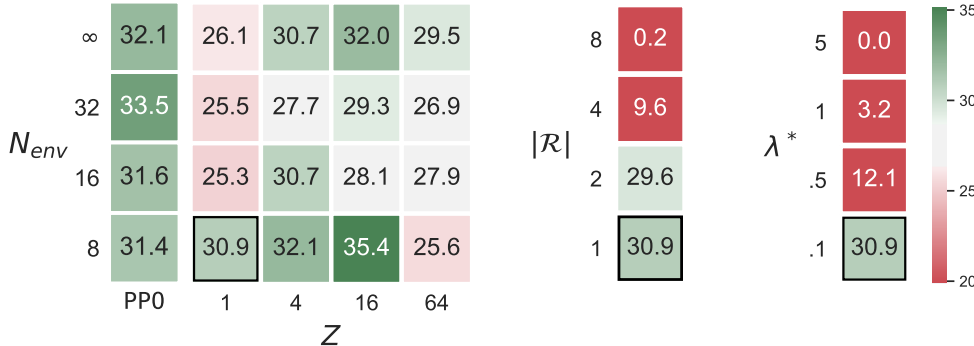
Figure 5: Episodic reward for different AUP settings. Higher reward is better.

Results. As N_{env} increases, side effect score tends to increase. AUP robustly beats PPO on side effect score: for each N_{env} setting, AUP's worst configuration has lower score than PPO. Even when $N_{\text{env}} = \infty$, AUP ($Z = 16$) shows the potential to significantly reduce side effects without reducing episodic return.

AUP does well with a single latent space dimension ($Z = 1$). For most N_{env} settings, Z is positively correlated with AUP's side effect score. In appendix E, data show that higher-dimensional auxiliary reward functions are harder to learn, presumably resulting in a poorly learned auxiliary Q-function.

When $Z = 1$, reward decreases as N_{env} increases. We speculate that the CB-VAE may be unable to properly encode large numbers of environments using only a 1-dimensional latent space. This would make the auxiliary reward function noisier and harder to learn, which could make the AUP penalty term less meaningful.

AUP's default configuration achieves 98% of PPO's episodic return, with just over half of the side effects. The fact that AUP is generally able to match PPO in episodic reward leads us to hypothesize that the AUP penalty term might be acting as a shaping reward. This would be intriguing – shaping

usually requires knowledge of the desired task, whereas the auxiliary reward function is randomly generated. Additionally, after AUP begins training on the R_{AUP} reward signal at step 1.1M, AUP learns more quickly than PPO did (fig. 3), which supports the shaping hypothesis. AUP imposed minimal overhead: due to apparently increased sample efficiency, AUP reaches PPO's asymptotic episodic return at the same time as PPO in `append-still-easy` and `append-spawn` (fig. 3).

Surprisingly, AUP does well with a single auxiliary reward function ($|\mathcal{R}| = 1$). We hypothesize that destroying patterns decreases optimal value for a wide range of reward functions. Furthermore, we suspect that decreasing optimal value in general often decreases optimal value for any given single auxiliary reward function. In other words, we suspect that optimal value at a state is heavily correlated across reward functions, which might explain Schaul et al. [19]'s success in learning regularities across value functions. This potential correlation might explain why AUP does well with one auxiliary reward function.

We were surprised by the results for $|\mathcal{R}| = 4$ and $|\mathcal{R}| = 8$. In Turner et al. [22], increasing $|\mathcal{R}|$ reduced the number of side effects without impacting performance on the primary objective. We believe that work on better interpretability of AUP's Q_{R_i} will increase understanding of these results.

When $\lambda^* = .5$, AUP becomes more conservative. As λ^* increases further, the learned AUP policy stops moving entirely.

6   Discussion

We successfully scaled AUP to complex environments without providing task-specific knowledge – the auxiliary reward function was a one-dimensional variational autoencoder trained through uniformly random exploration. To the best of our knowledge, AUP is the first task-agnostic approach which reduces side effects and competitively achieves reward in complex environments.

Wainwright and Eckersley [23] speculated that avoiding side effects must necessarily decrease performance on the primary task. This may be true for optimal policies, but not necessarily for learned policies. AUP improved performance on `append-still-easy` and `append-spawn`, matched performance on `prune-still-easy`, and underperformed on `append-still`. Note that since AUP only regularizes learned policies, AUP can still make expensive mistakes during training.

AUP$_{\text{proj}}$ worked well on `append-spawn`, while only slightly reducing side effects on the other tasks. This suggests that AUP works (to varying extents) for a wide range of uninformative reward functions.

While `Naive` penalizes every state perturbation equally, AUP theoretically applies penalty in proportion to irreversibility (proposition 1). For example, the agent could move crates around (and then put them back later). AUP incurred little penalty for doing so, while `Naive` was more constrained and consistently earned less reward than AUP. We believe that AUP will continue to scale to useful applications, in part because it naturally accounts for irreversibility.

Future work.   Off-policy learning could allow simultaneous training of the auxiliary R_i and of R_{AUP}. Instead of learning an auxiliary Q-function, the agent could just learn the auxiliary advantage function with respect to inaction.

Some environments do not have a no-op action, or the agent may have more spatially distant effects on the world which are not reflected in its auxiliary action values. In addition, separately training the auxiliary networks may be costly, which might necessitate off-policy learning. We look forward to future work investigating these challenges.

The SafeLife suite includes more challenging variants of `prune-still-easy`. SafeLife also includes difficult `navigation` tasks, in which the agent must reach the goal by wading either through fragile green patterns or through robust yellow patterns. Additionally, AUP has not yet been evaluated in partially observable domains.

AUP's strong performance when $|\mathcal{R}| = Z = 1$ raises interesting questions. Turner et al. [22]'s small "`Options`" environment required $|\mathcal{R}| \approx 5$ for good performance. SafeLife environments are much larger than `Options` (table 1), so why does $|\mathcal{R}| = 1$ perform well, and why does $|\mathcal{R}| > 2$ perform poorly? To what extent does the AUP penalty term provide reward shaping? Why do one-dimensional encodings provide a learnable reward signal over states?

Conclusion. To realize the full potential of RL, we need more than algorithms which train policies – we need to be able to train policies which actually do what we want. Fundamentally, we face a frame problem: we often know what we want the agent to do, but we cannot list everything we want the agent *not* to do. AUP scales to challenging domains, incurs modest overhead, and induces competitive performance on the original task while significantly reducing side effects – without explicit information about what side effects to avoid.

Broader Impact

A scalable side effect avoidance method would ease the challenge of reward specification and aid deployment of RL in situations where mistakes are costly, such as embodied robotics tasks for which sim2real techniques are available. Conversely, developers should carefully consider how RL algorithms might produce policies with catastrophic impact. Developers should not blindly rely on even a well-tested side effect penalty.

In some applications, AUP may decrease performance on the primary objective. In this case, developers may be incentivized to "cut corners" on safety in order to secure competitive advantage.

Acknowledgments and Disclosure of Funding

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31, 2017.

[2] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[3] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pages 908–918, 2017.

[4] Nick Bostrom. *Superintelligence*. Oxford University Press, 2014.

[5] Frank M Brown. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*. Morgan Kaufmann, 2014.

[6] Bart Bussmann, Jacqueline Heinerman, and Joel Lehman. Towards empathic deep Q-learning. *arXiv:1906.10918*, 2019.

[7] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*, 2018.

[8] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[9] Victoria Krakovna, Laurent Orseau, Miljan Martic, and Shane Legg. Measuring and avoiding side effects using relative reachability. *CoRR*, abs/1806.01186, 2018.

[10] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity, 2020. URL https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity.

[11] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *arXiv:1711.09883*, November 2017.

[12] Gabriel Loaiza-Ganem and John P Cunningham. The continuous Bernoulli: fixing a pervasive error in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 13266–13276, 2019.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] Martin Pecka and Tomas Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *International Workshop on Modelling and Simulation for Autonomous Systems*, pages 357–375. Springer, 2014.

[15] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 2019.

[16] Kevin Regan and Craig Boutilier. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *AAAI*, 2010.

[17] Paul Rendell. Turing universality of the game of life. In *Collision-based computing*, pages 513–539. Springer, 2002.

[18] Stuart Russell. *Human compatible: Artificial intelligence and the problem of control*. Viking, 2019.

[19] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[21] Rohin Shah, Dmitrii Krasheninnikov, Jordan Alexander, Pieter Abbeel, and Anca Dragan. The implicit preference information in an initial state. In *International Conference on Learning Representations*, 2019.

[22] Alexander Matt Turner, Dylan Hadfield-Menell, and Prasad Tadepalli. Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 385–391, 2020.

[23] Carroll L Wainwright and Peter Eckersley. Safelife 1.0: Exploring side effects in complex environments. *arXiv:1912.01217*, 2019.

[24] Shun Zhang, Edmund H Durfee, and Satinder P Singh. Minimax-regret querying on side effects for safe optimality in factored Markov decision processes. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4867–4873, 2018.