Composing Efficient, Robust Tests for Policy Selection

Dustin Morrill[1] **Thomas J. Walsh**[1] **Daniel Hernandez**[1] **Peter R. Wurman**[1] **Peter Stone**[1,2]

[1]Sony AI, New York, NY, USA
[2]Department of Computer Science, The University of Texas at Austin, Austin, TX USA

Abstract

Modern reinforcement learning systems produce many high-quality policies throughout the learning process. However, to choose which policy to actually deploy in the real world, they must be tested under an intractable number of environmental conditions. We introduce RPOSST, an algorithm to select a small set of test cases from a larger pool based on a relatively small number of sample evaluations. RPOSST treats the test case selection problem as a two-player game and optimizes a solution with provable k-of-N robustness, bounding the error relative to a test that used all the test cases in the pool. Empirical results demonstrate that RPOSST finds a small set of test cases that identify high quality policies in a toy one-shot game, poker datasets, and a high-fidelity racing simulator.

1 INTRODUCTION

Reinforcement learning (RL) [Sutton and Barto, 2018] policies have made a number of stunning breakthroughs in multiplayer games [Silver et al., 2016, Moravčík et al., 2017, Brown and Sandholm, 2018, Vinyals et al., 2019, Brown and Sandholm, 2019, Wurman et al., 2022, FAIR et al., 2022, Perolat et al., 2022]. However, the process of choosing an RL policy for production usage, either in an exhibition or deployment for end users, is challenging. Practitioners often generate many policies that perform well during training but which require thorough vetting on alternative conditions or opponents. Ideally, we would construct a test case for every conceivable deployment scenario, evaluate each policy on each test case, and rank each policy according to a weighted average of test case results. However, such a procedure is typically infeasible because of the sheer numbers of policies and deployment scenarios, especially if test cases are lengthy or involve people. In this work, we present a method for selecting a small number of test cases from a larger pool that minimizes the reduction in test quality.

Practitioners from other fields, *e.g.*, educational testing [van der Linden, 2005], will recognize this problem as *test construction*–selecting a small yet robust set of test cases, based on limited data, to evaluate many candidates. This set of test cases should contain enough information to indicate performance over the whole test case pool. For instance, if a policy can defeat a skilled opponent, we can infer that it can defeat an unskilled opponent. However, complicated domains contain complex intransitive relationships between policies, necessitating test case diversity. In addition, there is considerable uncertainty over what policies may be produced in the future and what test cases are the most important to game designers. This uncertainty needs to be considered because once test cases are chosen, the future policies to assess may be the most difficult ones for the test to evaluate accurately. Therefore, a robust solution is required.

We introduce a framework, *robust population optimization for a small set of test cases* (*RPOSST*), to compose an efficient robust test of a fixed size. RPOSST tunes its test to approximate the test scores of adversarially selected policies and test case averaging weights, given test case results on a small set of policies. We present two RPOSST algorithms representing different use cases, focusing on RPOSST$_{\text{SEQ}}$, which is better suited to current RL deployment pipelines. We provide robustness guarantees for RPOSST$_{\text{SEQ}}$ and CVaR RPOSST$_{\text{SEQ}}$ (a convenient special case) for k-of-N robustness measures [Chen and Bowling, 2012]. These guarantees provide confidence that RPOSST test scores for future deployment candidates are reliable.

Our contributions include the RPOSST framework, including two algorithm versions, robustness guarantees, and empirical validation in domains widely ranging in complexity. Empirical results are presented for a toy one-shot game simulating race car passing, computer poker competition datasets, and the high fidelity racing simulator, Gran Turismo™ 7.

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:1456–1466.

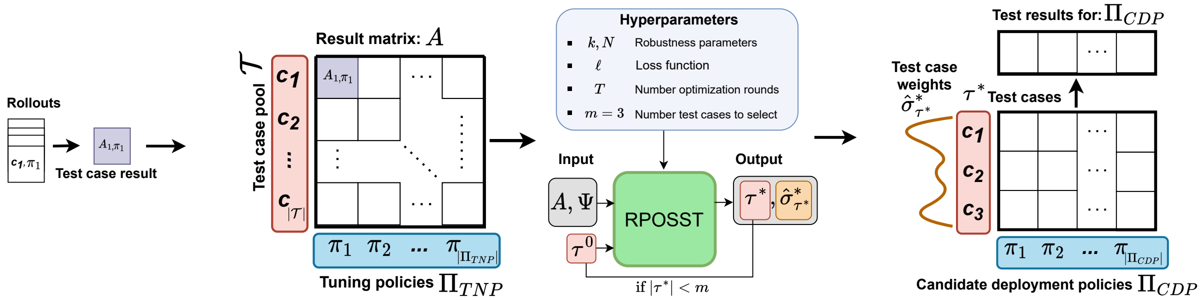

Rollouts
c_1, π_1
A_{1,π_1}
Test case result
Test case pool $\mathcal{T}$
c_1
c_2
$c_{|\mathcal{T}|}$
Result matrix: A
A_{1,π_1}
π_1 π_2 ... $\pi_{|\Pi_{TNP}|}$
Tuning policies Π_{TNP}
Hyperparameters
k, N Robustness parameters
ℓ Loss function
T Number optimization rounds
$m = 3$ Number test cases to select
Input
A, Ψ
τ^0
RPOSST
Output
$\tau^*, \hat{\sigma}^*_{\tau^*}$
if $|\tau^*| < m$
Test results for: Π_{CDP}
Test case weights $\hat{\sigma}^*_{\tau^*}$
τ^* Test cases
c_1
c_2
c_3
π_1 π_2 ... $\pi_{|\Pi_{CDP}|}$
Candidate deployment policies Π_{CDP}

Figure 1: Policy testing with RPOSST. From left to right, the result matrix A is constructed from rollouts, *i.e.*, $A_{i,j}$ is the average rollout outcome for tuning policy j (Π_{TNP} is the set of tuning policies) on test case i. RPOSST analyzes A, taking into account uncertainty distribution Ψ and a (possibly empty) initial set of test cases that must be used, τ^0. RPOSST outputs an efficient robust test $\langle \tau^*, \hat{\sigma}^*_{\tau^*} \rangle$, here using only $m = 3$ test cases (if $\mathcal{T}$ is too large to select all 3 test cases at once, τ^* can be fed back into RPOSST as τ^0). New candidate deployment policies, Π_{CDP}, are tested against each test case in τ^* and each result is weighted according to $\hat{\sigma}^*_{\tau^*}$, producing a test score for each candidate deployment policy.

They show that RPOSST can dramatically reduce (compared to the full set) the number of test cases needed to identify good deployment policies.

2 PROBLEM DEFINITION

The goal of *policy testing* is to evaluate the strengths and weaknesses of a large set of *candidate deployment policies*, $\Pi_{\text{CDP}} \subseteq \Pi$, in order to choose one for deployment. A *policy* $\pi \in \Pi$ in this setting can be any mapping from environment observations to a distribution over actions (*e.g.*, Markov policies; Sutton and Barto [2018]). A policy is evaluated on a *test* consisting of various test cases chosen from a pool, $\mathcal{T}$. Each *test case* simulates an important aspect of the deployment environment, for example, different parameter settings like weather conditions or different opponent policies in a competative game. For straightforward comparisons between policies, we summarize a policy π's test results with a scalar *test score*, computed as the weighted average of π's test case results according to *test case weights*, $\sigma \in \triangle^{|\mathcal{T}|}$.

If $\mathcal{T}$ is small, then right before deployment we could simply test each policy, rank the policies in Π_{CDP} according to the test scores, and deploy the best one. However, if policies will encounter a wide range of conditions during deployment, *e.g.*, hundreds or thousands of different players for a policy deployed to a popular video game, then $\mathcal{T}$ ostensibly needs to be large in order to adequately reflect such diversity. The linear scaling in $|\mathcal{T}|$ presents not just a computational burden, but also costs in sample complexity (if the test cases are lengthy) or even in person-time if human quality assurance testers might be needed for test cases.

This work addresses the problem of composing an efficient *test*, $\langle \tau, \hat{\sigma}_\tau \rangle$, by selecting a small number of test cases $\tau \subset \mathcal{T}$ and test case weights $\hat{\sigma} \in \triangle^{|\tau|}$ to approximate a full test, $\langle \mathcal{T}, \sigma \in \triangle^{|\mathcal{T}|} \rangle$. Complicating this task are two sources of uncertainty to which the efficient test must be robust. First, $\langle \tau, \hat{\sigma}_\tau \rangle$ ought to be used on new candidate deployment policies, so Π_{CDP} is unknown before $\langle \tau, \hat{\sigma}_\tau \rangle$ is chosen. Second, the desired *target distribution*, σ, defining the full test to approximate may drift after $\langle \tau, \hat{\sigma}_\tau \rangle$ is chosen.

We assume access to a small set of representative *tuning policies* $\Pi_{\text{TNP}} \subset \Pi$ for immediate testing (Section 4 discusses practical considerations in the composition of Π_{TNP}). Additionally, our algorithm takes as input a joint distribution Ψ over Π_{TNP} and $\triangle^{|\mathcal{T}|}$ to represent the combined uncertainty about which policies the output test will be applied to and which target distribution to approximate. See Figure 1 for an illustration of the test composition pipeline.

As a concrete example of the terms above and the need for robustness in the face of uncertainty, consider a car-racing agent developed for a one-on-one racing game. The first source of uncertainty is over the future policies we may want to test. Consider the case where, at test construction time, we have policies from two training runs–one that produces aggressive (collision-prone) policies, and another that produces more polite policies, but we are uncertain about which type will be best suited for the game. In this case, we want the selected test cases to provide good evaluations on policies from either set, and thus require Ψ to reflect this uncertainty. Policies from both sets should be included in Π_{TNP} and our algorithm needs to be robust to policies within Π_{TNP}.

The second source of uncertainty is over which test cases are most important. Imagine that we have some test cases that specifically target and penalize off-track infractions. In the future, game designers could request fewer infractions or allow for more risky racing lines. To hedge against both of these possibilities we can add two target distributions to Ψ, one where off-track tests cases have higher weights than the other test cases and another where they have lower weights. The job of an algorithm (such as RPOSST) is then to ensure its tests are accurate according to both target distributions.

3 BACKGROUND

In order to compose an efficient and robust test, we utilize established game-theoretic frameworks for modeling robustness and learning optimal decisions (specifically, regret minimization). The following subsections present background material on these two topics.

3.1 ROBUSTNESS

The idea of *robustness* is to prepare for an unfavorable portion of possible outcomes sampled from an uncertainty distribution. In our formulation of policy testing the uncertainty distribution covers the future policies in Π_{CDP} and the target distribution. A *percentile robustness measure* [Charnes and Cooper, 1959], μ, is a formal representation of a robustness criterion as a probability distribution over percentiles. For example, if μ has all of its weight on 0.01, then an m-size test with weights $\hat{\sigma}_\tau$ that is robust according to μ, then the test minimizes test score error on $\hat{\sigma}_\tau$'s worst 1% of policy–target-distribution pairs sampled from Ψ.

The *k-of-N robustness measures* [Chen and Bowling, 2012] are percentile robustness measures defined by parameters $k, N \in \mathbb{N}$, $1 \leq k \leq N$, that permit *tractable* optimization procedures. This parameterization reflects the mechanics of how an efficient test $\langle \tau, \hat{\sigma}_\tau \rangle$ is evaluated on such a measure: N policy–target-distribution pairs are sampled from Ψ and $\hat{\sigma}_\tau$'s performance is averaged over the k worst pairs for $\hat{\sigma}_\tau$. Every k-of-N robustness measure is a non-increasing function, *i.e.*, more weight is placed on smaller percentiles, and the fraction ${}^{k}\!/_{N}$ represents the percentile (technically the fractile) around which the measure decreases.

In our test construction setting, the choice of k and N reflects the designer's tolerance for test scores that are bad because of "unlucky" outcomes from Ψ (that is, test scores with large error on policy–target-distribution pairs sampled from Ψ, even if they are sampled infrequently). Optimizing for performance under small percentiles (*e.g.*, setting $k = 1,\ N = 100$) yields tests with a small maximum test score error across Π_{TNP}. Then, even if each candidate deployment policy resembles the tuning policy that has the largest test score error, the optimized test will yield small test score errors. In contrast, optimizing for the uniform measure ($k = N$) optimizes for mean performance across Π_{TNP}, essentially assuming $\Pi_{\text{CDP}} = \Pi_{\text{TNP}}$, which can lead to large test score error on the actual candidate deployment policies.

As $N \to \infty$, the k-of-N robustness measure approaches the *conditional value at risk* (*CVaR*) robustness measure at the ${}^{k}\!/_{N}$ fractile [Chen and Bowling, 2012], which evenly weights all of the fractiles $\leq {}^{k}\!/_{N}$ and puts a weight of zero on all larger fractiles. Formally, the robustness optimization objective is to minimize the *percentile performance loss*:

$$L_{\mu,\Psi}(\hat{\sigma}_\tau) = \inf_{y \in \mathcal{Y}} \int_{\eta \in [0,1],\, \mathbb{P}[\ell(\hat{\sigma}_\tau; \pi, \sigma) \leq y(\eta)] \geq \eta} y(\eta) \mu(d\eta), \tag{1}$$

under a loss function $\ell : \triangle^{|\mathcal{T}|} \times \Pi_{\text{TNP}} \times \triangle^{|\mathcal{T}|} \to \mathbb{R}$ where we overload ℓ for incomplete test case weight vectors by filling in zeros for missing elements, $\langle \pi, \sigma \rangle \sim \Psi$, and $\mathcal{Y}$ is the class of real-valued, bounded, μ-integrable functions on $[0, 1]$. An *efficient* (m-size) *μ-robust test* is a minimizer of $L_{\mu,\Psi}$ across all $\hat{\sigma}_\tau$ where $\tau = m$.

The optimization of the percentile performance loss under k-of-N robustness measure, $\mu_{k\text{-of-}N}$, can be modeled as a zero-sum imperfect information game [Chen and Bowling, 2012]. Here, a protagonist player constructs efficient tests and an antagonist chooses a tuning policy to test and a target distribution. For their payoffs, the antagonist receives the test score error of the protagonist's test given the antagonist's tuning policy and target distribution while the protagonist receives the negation. The k and N parameters determine which target distributions and tuning policies that the antagonist can choose from and how many pairs must be averaged across. At the start of the game, N target-distribution–tuning-policy pairs are sampled. From these N pairs, the antagonist must select k of them. Finally, one of these k pairs is sampled, both players receive their payoffs, and the game ends. A *minimax* test for the protagonist, *i.e.*, one that minimizes the protagonist's maximum loss in this game is a $\mu_{k\text{-of-}N}$-robust test.

3.2 REGRET

While the game above models the optimization process, it does not instruct the protagonist on *how* to choose test cases to win. A no-regret *online decision process* (*ODP*) algorithm can find approximate minimax decisions by repeatedly playing out the game and improving over time from payoff feedback. Formally, on each round t of the game, an ODP algorithm chooses an efficient test $\langle \tau^t, \hat{\sigma}^t_{\tau^t} \rangle$ and receives the *payoff function* $v^t = -\nabla_{\hat{\sigma}^t_{\tau^t}} \ell(\hat{\sigma}^t_{\tau^t}; \pi^t, \sigma^t)$ as feedback given $\langle \pi^t, \sigma^t \rangle$ chosen by the antagonist. If the antagonist always plays a best response to the ODP algorithm, that is, the tuning-policy–target-distribution pair that maximizes the loss of $\hat{\sigma}^t_{\tau^t}$ on each round $t \in \{1, \ldots, T\}$, $T \geq 1$, then the *no-regret* property ensures that at least one of the tests in the sequence $\langle\langle \tau^t, \hat{\sigma}^t_{\tau^t}\rangle\rangle_{t=1}^T$ is at most $\mathcal{O}(G/\sqrt{T})$ away from the minimax value, where $G > 0$ is the maximum magnitude of the loss gradient (see Lockhart et al. [2019a,b] and Appendix Proposition 2.4 for more details).

Regret matching$^+$ [Tammelin, 2014, Tammelin et al., 2015] is a no-regret algorithm for simplex decision sets, *e.g.*, the m dimensional test case weight space $\triangle^m$, that selects $\hat{\sigma}^t_{\tau^t} = \frac{q^{1:t-1}}{\mathbf{1}^\top q^{1:t-1}}$ using *pseudoregrets* $q^{1:t} = [q^{1:t-1} + \rho^t]_+$, $q^{1:0} = \mathbf{0}$, where $\rho^t = v^t - (v^t)^\top \hat{\sigma}^t_{\tau^t}$ is the *instantaneous regret* vector ($\hat{\sigma}^t_{\tau^t} = \frac{1}{d}\mathbf{1}$ if none of the pseudoregrets are positive).

4 RPOSST

Our approach, *robust population optimization for a small set of test cases* (*RPOSST*) begins by evaluating each tuning policy $\pi \in \Pi_{\text{TNP}}$ on each test case $c \in \mathcal{T}$, yielding a $|\mathcal{T}| \times |\Pi_{\text{TNP}}|$ result matrix A of test case results. As an optimization approach, RPOSST aims to minimize prediction errors, as measured by a convex function $\varDelta : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, *e.g.*, the absolute difference $\varDelta(\hat{x}, x) = |\hat{x} - x|$. RPOSST robustly optimizes for a small set of test cases and a weighting over them according to how well it reproduces test scores admitted by A as measured by a loss function

$$\ell : \hat{\sigma}; \pi_j, \sigma \mapsto \varDelta(\underbrace{E_{i\sim\hat{\sigma}}[A_{i,j}]}_{\hat{\sigma}\text{'s test score for } \pi_j.}, \underbrace{E_{i\sim\sigma}[A_{i,j}]}_{\sigma\text{'s test score for } \pi_j.}),$$

on test case distribution $\hat{\sigma} \in \triangle^{|\mathcal{T}|}$ compared to $\sigma \in \triangle^{|\mathcal{T}|}$ with respect to test results from the j^{th} tuning policy π_j. Since $\hat{\sigma}$ is being used to produce test scores that approximate those under σ, we call σ a *target distribution* in this context. Our goal is to select a small number of test cases, so we constrain RPOSST to output weights $\hat{\sigma}_\tau \in \triangle^m$ for groups of test cases $\tau \subset \mathcal{T}$ of size m.

Though $\mathcal{T}$ is large, the cost of computing A is balanced by the savings of using fewer test cases for future policies. RPOSST is robust to any distribution over Π_{TNP}, so as long as this set covers the space of Π_{CDP} (*i.e.*, all $\pi \in \Pi_{\text{CDP}}$ are convex mixtures of Π_{TNP}), this robustness imparts a minimum test accuracy guarantee even on deployment candidates. Intuitively, this means the quality of RPOSST's tests will tend to improve with more diverse tuning policies. Accordingly, it should be beneficial for a tuning policy to represent an extreme point in a reasonable region of policy space, or at least for it to be generated with a method similar to that which will generate deployment candidates (*e.g.*, sampled from checkpoints of RL training runs). That way, the tuning policies include a diverse collection of skilled and unskilled policies with random variations, while retaining architectural and algorithmic similarities to future deployment candidates.

Following the earlier discussion of k-of-N robustness, we frame the optimization in RPOSST as a zero-sum game. By adversarially choosing policies to test, the antagonist forces RPOSST to compose tests that are better at accurately testing the more difficult-to-assess policies in the tuning set, providing a degree of robustness to the distribution of future deployment candidates. Similarly, by adversarially choosing the target distribution, the antagonist also forces RPOSST to be robust along this dimension. The steps of each round $t = 1, \ldots, T$ of our optimization game follows.

1. The protagonist must choose an m-tuple of test cases $\tau^t \subset \mathcal{T}$ and weights $\hat{\sigma}^t_{\tau^t} \in \triangle^m$.
2. N policies to test and target distributions, $\langle\langle \pi_{j_i}, \sigma_i\rangle\rangle_{i=1}^N$, are sampled from uncertainty distribution Ψ.
3. The antagonist chooses the k worst policies and target distributions, *i.e.*, those that maximize $\ell(\hat{\sigma}^t_{\tau^t}; \pi_{j_i}, \sigma_i)$.
4. One of the k worst configurations is sampled uniformly, leading to the end of the round, at which point the protagonist receives the payoff $v^t_{\tau^t,(i)} = -\ell\big(\hat{\sigma}^t_{\tau^t}; \pi_{j_{(i)}}, \sigma_{(i)}\big)$, where the subscript (i) denotes the i^{th} element of a sorted list in descending order (the i^{th} worst for the protagonist).

The protagonist is allowed to update their strategy at the end of each round based on the expected payoff, $\mathbb{E}_{i\sim\text{Unif}(\{1,\ldots,k\})}\left[v^t_{\tau,(i)}\right]$, for each $\tau \in \mathcal{T}^m$ they could have chosen. The more rounds of the game that are run (the larger T is), the closer RPOSST gets to returning a minimax strategy, and consequently, a robust optimal selection of test cases and weights. Thus, in application, T can be set as large as is convenient under computational and time constraints. Theorem 4.1 gives a precise rate for RPOSST's improvement, with high probability, as a function of T. Although the protagonist must consider an exponential (in m) number of test case combinations, the premise of RPOSST is that we want a small set of test cases, so m will be small. To decrease computational requirements, RPOSST can be run in a loop to select test cases iteratively until m have been selected, at a potential cost to test accuracy compared to optimizing for the entire m-tuple at once.

4.1 ANTAGONIST INFORMATION MODELS

We consider two RPOSST algorithm variants that utilize different models of the information that the antagonist in our optimization game has before they make their choice. These models correspond to two policy testing use cases. The first, "simultaneous move" model is less pessimistic, but has impractical aspects, which are addressed by the subsequent "sequential move" model.

Simultaneous move. The simultaneous move model is a naïve application of the original k-of-N game by [Chen and Bowling, 2012]. In this model, the antagonist does not

```
Inputs: $\langle k, N, T_1, m, \Psi, \tau^0, \ell, T_2 \rangle$
$q_\tau^{1:0} \leftarrow \mathbf{0} \in \mathbb{R}^{m+|\tau^0|}$ for $\tau \in \mathcal{T}^m$
$T' \sim \mathrm{Unif}(\{1, \ldots, T_1\})$
for $t \leftarrow 1, \ldots, T'$ do
    for $\tau \in \mathcal{T}^m$ do
        $z^t \leftarrow \mathbf{1}^\top q_\tau^{1:t-1}$
        $\hat{\sigma}_\tau^t \leftarrow q_\tau^{1:t-1} / z^t$ if $z^t > 0$ else $\mathbf{1}/m$
        // Add zeros to ensure $\hat{\sigma}_\tau^t \in \triangle^{|\mathcal{T}|}$.
        $\hat{\sigma}_\tau^t(x) \leftarrow 0$ for $x \in \mathcal{T} \setminus (\tau \cup \tau^0)$
        $[\ell_{\tau,(i)}]_{i=1}^k \leftarrow \mathcal{L}_{\text{k-of-N}}(\hat{\sigma}_\tau^t, \langle k, N \rangle, \Psi, \ell)$
        $v_\tau^t \leftarrow \frac{-1}{k} \sum_{i=1}^k \frac{\partial \ell_{\tau,(i)}, \pi_{j_{(i)}}}{\partial \hat{\sigma}_\tau^t}$
        // Update regret matching$^+$.
        $\rho_\tau^t \leftarrow v_\tau^t - (\hat{\sigma}_\tau^t)^\top v_\tau^t$
        $q_\tau^{1:t} \leftarrow [q_\tau^{1:t-1} + \rho_\tau^t]_+$
$\tau^* \leftarrow$
    $\mathrm{SR}\Big(\tau \mapsto \frac{1}{2kL} \mathbf{1}^\top \mathcal{L}_{\text{k-of-N}}\Big(\hat{\sigma}_\tau^{T'}, \langle k, N \rangle, \Psi, \ell\Big), T_2\Big)$
return $\tau^*, \hat{\sigma}_{\tau^*}^{T'}$

Procedure $\mathcal{L}_{\text{k-of-N}}$    Inputs: $\langle \hat{\sigma}, k, N, \Psi, \ell \rangle$
    for $i = 1 \ldots N$ do
        // Sample antagonist actions.
        $\pi_{j_i}, \sigma_i \sim \Psi$
        // Evaluate $\hat{\sigma}$.
        $\ell_i \leftarrow \ell(\hat{\sigma}; \pi_{j_i}, \sigma_i)$
    // Sort to identify the worst $k$.
    $\mathrm{Sort}\Big([\ell_i]_{i=1}^N\Big)$
    return $[\ell_{(i)}]_{i=1}^k$
```

Algorithm 1: RPOSST$_{\text{SEQ}}$ with regret matching$^+$ and Successive Rejects

observe which m-tuple of test cases, τ^t, is selected by the protagonist on each round t. Instead, it is randomized with a distribution $\hat{\sigma}_{\mathcal{T}}^t \in \triangle^{|\mathcal{T}|^m}$. This model corresponds to the policy testing use case where a new m-tuple of test cases is sampled independently for each test that is performed. Every test only evaluates m cases, as desired from a computational efficiency perspective, however, the particular test cases used in each test could be different, making results incomparable across tests. See Appendix Section 5 for additional details.

Sequential move. In the sequential move model, the antagonist observes τ^t before acting. The antagonist is thus able to tailor their choice of $\big\langle \big\langle \pi_{j_{(i)}}, \sigma_{(i)} \big\rangle \big\rangle_{i=1}^k$ to whichever τ^t is selected, and randomizing over the m-tuple of test cases has no benefit to the protagonist. Since the antagonist observes τ^t, the protagonist must update all the weights that they would apply to each test case tuple τ as if $\tau^t = \tau$. Thus, the selection of τ^t does not impact the protagonist's updates and we need not explicitly select an m-tuple until the very end of the algorithm, after $T' \sim \mathrm{Unif}(\{1, \ldots, T_1\})$ rounds.[1]

Since the set of N losses observed on each round are generally random, we cannot reuse them to identify which m-tuple leads to the lowest loss using the the test case weights computed after running for T' rounds, $\langle \hat{\sigma}_\tau^{T'} \rangle_{\tau \in \mathcal{T}^m}$. In addition, we cannot access expected k-of-N losses directly; we must estimate them by sampling from Ψ. Therefore, the selection of a single τ is a "best arm identification" problem, where $\mathcal{T}^m$ is the set of arms. The Successive Rejects (SR) [Audibert et al., 2010] algorithm is an exploration-only bandit algorithm that can be used to solve this problem with a worst-case guarantee on the probability that it identifies the best arm. The more SR iterations that are run, the more likely it is to select the best arm. Algorithm 1 shows how to implement RPOSST for the sequential move model using regret matching$^+$ for tuning the test case weights and SR for the final selection of an m-tuple.

In specific applications, an example of which we will see in Section 4.2 and our experiments, we can construct our optimization game so that it is deterministic, and consequently, we can replace SR with a simple argmax.

The RPOSST$_{\text{SEQ}}$ objective is the percentile performance loss

$$\min_{\substack{\tau \in \mathcal{T}^m \\ \hat{\sigma}_\tau \in \triangle^m}} \inf_{y \in \mathcal{Y}} \int_{\substack{\eta \in [0,1] \\ \mathbb{P}[\ell(\hat{\sigma}_\tau; \pi_j, \sigma) \le y(\eta)] \ge \eta}} y(\eta) \mu_{k\text{-of-}N}(d\eta), \qquad (2)$$

where $\langle \pi_j, \sigma \rangle \sim \Psi$.

The sequential move model represents the policy testing use case where we select and fix m test cases and test case weights for all future test policies. Test scores are easily reproducible and comparable across test applications since the test cases never change.

Theorem 4.1. *After* $T' \sim \mathrm{Unif}(\{1, \ldots, T_1\})$, $T_1 > 0$, *rounds of its optimization game, Algorithm 1 selects an m-tuple of test cases, τ^* and weights $\hat{\sigma}_{\tau^*}^{T'} \in \triangle^m$ that, with probability* $(1-p)(1-q)(1-\alpha)$, $p, q, \alpha > 0$, *are $\frac{\varepsilon}{q}$-optimal for Equation* (2), *where* $\varepsilon = \mathcal{O}\Big(\sqrt{\frac{1}{T_1} m} + \sqrt{\frac{1}{T_1} \log(1/p)}\Big)$ *and* $\alpha = \mathcal{O}\big(\mathrm{e}^{-T_2}\big)$.

All proofs deferred to the Appendix. In the extreme case where Π_{TNP} covers Π, then this optimality result, (in terms of an upper bounded percentile loss integral), extends to all deployment candidates Π_{CDP}.

4.2 DETERMINISTIC CVAR RPOSST

While in general, an RPOSST algorithm has a randomized procedure and a non-deterministic optimality guarantee,

[1] RPOSST is run for T' rather than T_1 rounds because we cannot guarantee a decrease in worst-case loss after every round. See the proof of Theorem 4.1 for more details.

we can actually select hyperparameters so that RPOSST is deterministic, making the procedure simpler and more reliable. If we fix the ratio ${}^{k}/_{N}$ and allow $N \to \infty$, the k-of-N robustness measure converges toward the CVaR measure at the ${}^{k}/_{N}$ fractile. A k-of-N algorithm where $N \to \infty$ cannot be implemented with the usual sampling procedure, but it can be implemented if the distribution characterizing our uncertainty, Ψ, has finite support.

Sampling Ψ infinitely would result in sampling all tuning-policy–target-distribution pairs in its support exactly in proportion to their probabilities. Rather than selecting k tuning-policy–target-distribution pairs, the antagonist must select pairs until their cumulative probability sums to ${}^{k}/_{N}$. Effectively, the antagonist assigns weights

$$\alpha_{(i)} = \min\left\{ \Psi\big(\langle \pi_{j_{(i)}}, \sigma_{(i)} \rangle\big), {}^{k}/_{N} - \sum_{h=1}^{i-1} \alpha_{(h)} \right\}$$

to each tuning-policy–target-distribution pair in Ψ's support, where the ordering between pairs is determined by the loss each induces for the protagonist. Finally, these tuning-policy–target-distribution pairs are sampled according to the normalized weights $\frac{\alpha_{(i)} N}{k}$.

The robustness guarantees become deterministic because the entire RPOSST algorithm, denoted as CVaR(η) RPOSST for the $\eta = {}^{k}/_{N}$ fractile, can be run using exact expectations (excluding randomness in A, which is taken as given in RPOSST). Determinism in RPOSST$_{\text{SEQ}}$ allows us to directly check the exact expected loss of each test case distribution on each round, letting us track the lowest loss test case distribution across all rounds. This tracking, in turn, allows us to avoid both sampling T' and running the SR algorithm to do the final selection. Instead, we can simply return the lowest loss test case distribution across all T rounds.

If there are d tuning-policy–target-distribution pairs in Ψ's support, then the expected CVaR(η) loss of the protagonist on round t is $L^t = \min_{\tau \in \mathcal{T}^m} \sum_{i=1}^{d} \frac{\alpha_{(i)}}{\eta} \ell(\hat{\sigma}^t_\tau; \pi_{j_{(i)}, \sigma_{(i)}})$. The round with the lowest expected loss is $t^* = \arg\min_{t \in \{1,\ldots,T\}} L^t$, and this definition allows us to state the following corollary.

Corollary 4.2. *Assume that $\Psi \in \triangle^d$ for some finite $d \geq 1$. After T rounds of the CVaR(η) RPOSST$_{\text{SEQ}}$ optimization game, where the protagonist chooses m-size tests according to regret matching$^+$ against a best response antagonist, τ^* and $\sigma^{t^*}_{\tau^*}$ are ε-optimal for Equation* (2) *under the η-fractile CVaR robustness measure, where $\varepsilon = \mathcal{O}\Big(\sqrt{\tfrac{1}{T} m}\Big)$.*

Pseudocode for CVaR(η) RPOSST$_{\text{SEQ}}$ is presented in Appendix Algorithm 1.

In addition, we can construct a series of ablations of CVaR RPOSST$_{\text{SEQ}}$ to act as baselines for experiments, and to make a connection to the test-construction literature. CVaR RPOSST$_{\text{SEQ}}$ generalizes an intuitive algorithm: find the m-tuple of test cases that minimizes the maximum error assuming a uniform distribution over the tuple. This *minimax uniform* algorithm is implemented by executing only the initialization and selection steps of CVaR(0) RPOSST$_{\text{SEQ}}$ ($T = 0$). Further simplifying, *minimax(TTD) uniform* performs the antagonist maximization only over target distributions and assumes a uniform distribution over tuning policies. *Minimax(TNP) uniform* performs the antagonist maximization only over tuning policies and assumes a uniform target distribution. *Miniaverage uniform* assumes both a uniform distribution over tuning policies and for the target distribution.

Additionally, we could select test cases one at a time to minimize the maximum error, echoing greedy algorithms from the test-construction literature (Chapter 4 of van der Linden [2005]). This *iterative minimax* algorithm is almost the same as running the initialization and return steps of CVaR(0) RPOSST$_{\text{SEQ}}$ to select a single test case in a loop. The sole difference being that iterative minimax could select the same test case multiple times within its loop to adjust the test case weighting away from uniform.

5 EXPERIMENTS

We explore CVaR RPOSST$_{\text{SEQ}}$'s performance in three two-player games spanning the range of complexity from a toy one-shot game to a high-fidelity racing simulator, in comparison with minimax and miniaverage baselines. We show that robustness does tend to decrease test score errors on holdout policies and that RPOSST specifically either outperforms or performs about as well as each baseline in each domain.

5.1 EXPERIMENTAL SETUP

In each domain, we start with data from playing out every pairing of $n > 0$ policies, yielding a matrix of scores for the column policy. Each policy along the rows of this matrix is then treated as a test case, making the score at row i and column j the result of evaluating policy j on test case i.

To emulate unknown deployment candidate policies to be tested, we hold out $h > 0$ columns of this matrix and call the policy associated with a holdout column a *holdout policy*. The remaining columns represent the test case results for the set of tuning policies. The resulting $n \times (n - h)$ matrix is shifted and rescaled so that all entries are between zero and one, and then it is set as the test result matrix A that our methods take as input. Note, although h test cases are generated by holdout policies, as test cases they cannot provide any special information about what tests would be effective on the holdout policies. To simulate scenarios where the set of tuning policies covers the set of future candidate deployment policies to varying degrees, we run

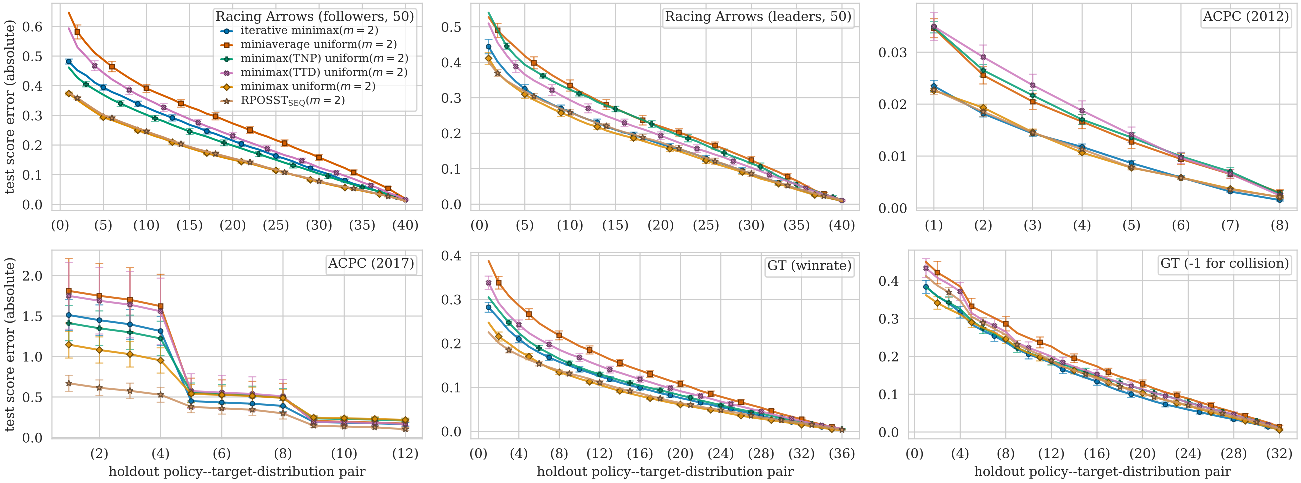
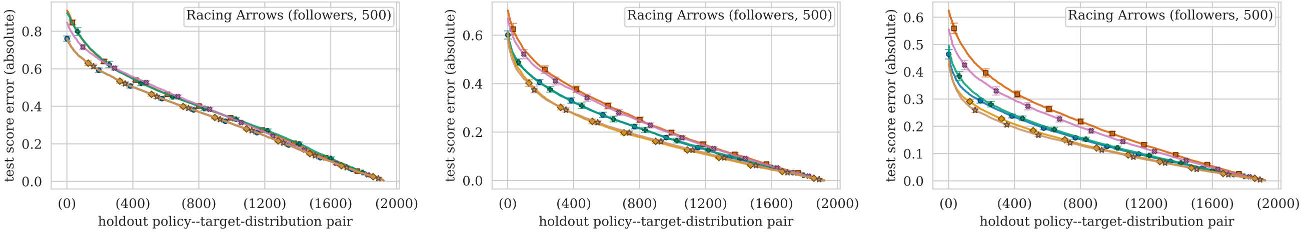


Figure 2: Expected test score error (absolute difference) across holdout-policy–target-distribution pairs on (top left and middle) Racing Arrows, (top right) the 2012 two-player, limit competition of the ACPC, (bottom left) the 2017 two-player, no-limit competition of the ACPC, (bottom middle and right) Gran Turismo™ 7 races, between CVaR(1%) RPOSST$_{\text{SEQ}}$ and baseline tests on 100 randomly sampled sets of holdout policies (20% of the full set of policies; 80% used as tuning policies). Holdout-policy–target-distribution pairs are sorted according to test score error. Each RPOSST$_{\text{SEQ}}$ instance was run for 500 rounds ($T = 500$). Errorbars represent 95% t-distribution confidence intervals.



(a) $m = 1$

(b) $m = 2$

(c) $m = 3$

Figure 3: Expected test score error across holdout-policy–target-distribution pairs on Racing Arrows where test cases are follower policies. Here, 500 Racing Arrows policies were sampled for both the follower and leader role and then 96% of policies of both roles were held out before running RPOSST and each baseline. Each column uses a different test size m. 100 sets of holdout policies were sampled and each RPOSST$_{\text{SEQ}}$ instance was run for 500 rounds ($T = 500$).

experiments with three different values of h: $0.2n$, $0.4n$, and $0.6n$. 100 different holdout sets are randomly sampled for each value of h and in each domain.

Given results for n test cases, the goal is to produce a distribution over $m < n$ test cases that provides accurate test results on the set of holdout policies, according to a set of target distributions. For our experiments, we use $m \in \{1, 2, 3\}$ and the set of target distributions generated from the softmax function applied to the negative average test case result under four different scales, specifically, $\exp\left(\frac{-\beta}{n} A\mathbf{1}\right) / \mathbf{1}^\top \exp\left(\frac{-\beta}{n} A\mathbf{1}\right)$ for $\beta \in \{0, 1, 2, 4\}$, so that the distributions put varying degrees of emphasis on test cases that are more difficult on average across the tuning policies. We set the RPOSST uncertainty distribution, Ψ, to be uniform over each tuning-policy–target-distribution pair. We set the CVaR percentile to 1% so that it is nearly optimizing for the worst-case, but is slightly less pessimistic, to add an additional distinguishing factor to RPOSST compared to the minimax and minaverage baselines. We use the absolute difference loss for both optimization and evaluation.

5.2 DOMAINS

We test RPOSST on the following three domains of varying complexity. Each domain has two variants arising from asymmetry, multiple datasets, or alternative scoring rules. Appendix Section 6 provides further details on each domain.

Racing Arrows. Racing Arrows is a two-player, zero-sum, one-shot, continuous action game invented for our experiments to replicate aspects of a passing scenario in a race featuring a "leader" player and faster "follower" player. The follower tries to pass the leader while the latter tries to block. Scores are recorded as 0 or +1 for a loss or win, respectively, for the column player, which is either the leader or the follower, depending on the configuration. We run RPOSST on both configurations. For our experiments, we sample 50 or 500 different leader and follower policies evenly spread through the valid policy space, angles in $[0, \pi]$, by taking 50 or 500 evenly spaced angles between $[0.05\pi, (1 - 0.05)\pi]$ and then shifting them independently with uniform samples in $[-0.05\pi, 0.05\pi]$.

Annual Computer Poker Competition. We take two open datasets from the Annual Computer Poker Competition (ACPC) [Bard et al., 2013] containing pairwise match data for poker agents submitted to the 2017 two-player, no-limit competition and the 2012 two-player, limit competition. These competitions contain different agent populations since they are separated by five years and are in different game formats (limit and no-limit). The 2017 competition consists of 15 agents and the 2012 competition consists of 12 agents. Scores are recorded as chip differentials of duplicate matches (two sets of hands where players play with the same set of shuffled decks in both seats).

Gran Turismo™ one-on-one races. Gran Turismo™ 7 (GT)[2] is a high fidelity racing simulator on the PlayStation™ platform. Previous versions of GT served as benchmarks for training RL policies [Fuchs et al., 2021, Song et al., 2021] including policies that outraced the best human competitors [Wurman et al., 2022] in four-on-four racing. We consider a simpler one-on-one racing scenario (see Appendix Section 6.3 for details). We carry out two experiments, one where test case results are average winrates, and another where policies receive 0 for a loss, $+1$ for a win, and -1 if there was a collision, making the game non-zero-sum. The test case pool is comprised of 43 trained RL policies and 3 built-in "AI" policies.

5.3 RESULTS AND ANALYSIS

The results of running CVaR(1%) RPOSST$_{\text{SEQ}}$ on each domain, with $m = 2$ and 20% of policies marked as holdout policies, are shown in Figure 2. The same set of figures with $m = 1$ and $m = 3$, as well as 40% and 60% holdout policies, are qualitatively similar, except that the differences between the algorithms are typically smaller, and are provided in Appendix Section 6.4.

Looking across each domain and variant, we can see that RPOSST$_{\text{SEQ}}$ performs nearly as well or better than all of the minimax and miniaverage baselines, particularly in terms of maximum error across holdout-policy–target-distribution pairs. Interestingly, RPOSST$_{\text{SEQ}}$ has noticeably lower error in ACPC 2017 and GT (winrate) on the four most difficult holdout-policy–target-distribution pairs to accurately evaluate. The improvement over the next best method is substantial in ACPC 2017 because RPOSST is the only method with an unlimited ability to optimize with a non-uniform test case weighting.[3] On the other variant in each domain, RPOSST$_{\text{SEQ}}$ is within the group of the lowest error methods. In the two Racing Arrows domains, RPOSST$_{\text{SEQ}}$ and minimax uniform substantially outperform the other methods, at least on the most difficult holdout-policy–target distribution pairs. This result shows that robustness is indeed beneficial here, but the uniform distribution over the selected two opponents happens to be quite effective. The GT variant where -1 is assigned to a collision appears to be more difficult than the winrate variant, as all the methods cluster together in this variant at higher errors than in the winrate variant.

These results illustrate the utility of incorporating robustness generally, as all of the robust methods tend to outperform miniaverage uniform. Minimax uniform and iterative minimax are the only baselines that minimize their maximum error over both tuning policy and target distribution uncertainty, and they are usually the next best methods after RPOSST$_{\text{SEQ}}$. Minimax(TNP) uniform typically outperforms minimax(TTD) uniform, showing that it is more important to be robust to the tuning policy than the target distribution, in these domains. When the target distributions are the same in the optimization and holdout evaluation phases, robustness should directly improve the minimum performance across holdout realizations. Since no effort was made to enforce any relationship between the tuning and holdout policies, this result suggests that robustness to the tuning policy can yield large error reductions when Π_{TNP} are even somewhat similar to the holdout policies.

As an example of RPOSST's capabilities, consider the pairs of opponent policies chosen as test cases in GT (winrate) over 100 experiment seeds (Appendix Table 2). RPOSST$_{\text{SEQ}}$ is both more accurate (Figure 2) and very consistent, choosing the same pair 90% of the time. Figure 4 illustrates the portion of the result matrix for just the two test cases most frequently chosen by RPOSST$_{\text{SEQ}}$ (test races against opponents 16 and 41). The race against policy 41 (bottom row) is chosen because that policy wins/loses about half the time, providing a 50/50 information split. Policy 16 is a weaker policy in many ways (more blue in the top row) but it serves to differentiate the worst policies (darker red squares in the left side of the matrix) from the rest of the policies, and to highlight the strongest policies. Specifically, the best performing policies almost always win against policy 16, which provides a strong complementary signal to the noisier but more competitive policy 41 test case. Overall, the two test cases indicate policies 1, 29, and 43 (darkest blue columns) are the strongest for deployment. Policy 1 is a built-in AI in an overpowered car but 29 and 43 are very strong RL policies. Looking at the overall winrate matrix (Appendix Figure 1b) we see that the same conclusion (the three darkest blue columns overall) would have been chosen using

[2] `https://www.gran-turismo.com/us/`

[3] Iterative minimax can change its test case distribution away from uniform, but only indirectly by selecting a test case it already selected on a previous iteration before it fills its test-case quota.

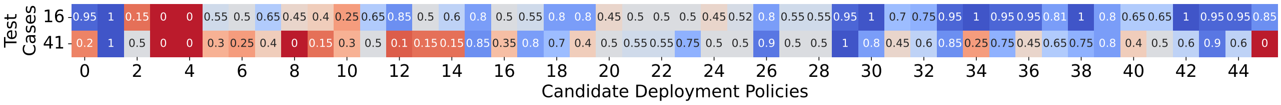


Figure 4: The GT test results of candidate deployment policies against the test case pair most favoured by RPOSST. Blue and red indicates positive and negative winrates respectively for the candidate deployment policy.

all 46 test cases. Compressing from 46 test cases to two presents a massive saving in test time for future policies, and shows RPOSST$_{\text{SEQ}}$ can construct small tests to select deployment policies in a real and complex video game.

The results in Figure 3 repeat the previous analysis in Racing Arrows but with ten times the number of policies. Only the results where follower policies are treated as test cases are shown, but the corresponding results where leader policies are test cases appear similar and are shown in Appendix Section 6.4. 96% of policies are held out, including those used as test cases, so there are only 20 test cases and tuning policies for RPOSST and the other algorithms to utilize. This experiment emulates a scenario where an efficient test is constructed once with a relatively small number of tuning policies and then reused for many future deployment candidates. As in the previous experiments, RPOSST is almost always one of the best methods.

6 RELATED WORK

The bulk of the work on policy selection in RL focuses on selecting opponents for *training* with self-play algorithms [Hernandez et al., 2021]. In that case, diversity is key for training additional policies to cooperate [Rahman et al., 2022] or compete [Liu et al., 2021, McAleer et al., 2022] with pre-existing policies. However, the selection of policies as training opponents is often guided by aggregate performance metrics across entire populations [Li et al., 2019, Lanctot et al., 2017, Omidshafiei et al., 2019, Balduzzi et al., 2018] and thus do not reduce the number of opponent pairings (test cases) required for assessments.

On the testing side, researchers in complex domains develop procedures for testing skill competency using hand-calibrated [Wurman et al., 2022] or randomly generated tests with complex percentile-scoring functions [Team et al., 2021]. Our work seeks to automate and target test construction in such scenarios. Complementary work [Rowland et al., 2019] treats the computation of a result matrix as a multi-armed bandit problem, each entry represented by one arm. While this method can greatly reduce sampling costs in the presence of low-variance outcomes, it does not generalize to policies outside its input population, with the testing of a new policy requiring adding extra arms to be estimated from scratch. However, this method could be used in tandem with RPOSST to reduce the samples required to compute A.

Learning to rank methods [Oosterhuis and de Rijke, 2021, Bruch, 2021, Hu et al., 2018] aim to find a function that ranks a set of items (*e.g.*, documents) based on the relevance of a given query, with hopes to generalize to future queries. Indeed, Akiyama et al. [2016] use learning to rank to evaluate action sequences. However, predicting unseen policy performances under this model requires the tuning policies to be the queries, which would produce a ranking of the test cases themselves. The scores from such tests would therefore be incomparable across policies, violating one of our main objectives.

Test construction in educational modeling [van der Linden, 2005] starts from an item bank and a statistical model (*e.g.*, Item Response Theory [Embretson et al., 2000]) predicting the probability of answering each item correctly given a student's (unobserved) skill level. That model yields an *information matrix* and then automatic test construction methods, including linear optimization or greedy heuristics, can then build a finite-sized test. By contrast, we do not assume a model of the response variance or a univariate skill measurement, so a closed-form calculation of information is often infeasible. However, we do empirically compare our optimization approach to the greedy heuristic.

7 CONCLUSION AND FUTURE WORK

RPOSST is, to the best of our knowledge, the first algorithm to directly address test construction for reinforcement learning policies. By leveraging the k-of-N framework, RPOSST provides bounds on the approximation error of the resulting test despite uncertainty over the exact policies that will be evaluated and the desired test case weighting in the future. Thus, RPOSST provides a much needed tool for policy selection in real-world deployment scenarios. An interesting direction for future work is generating the test cases themselves [Marris et al., 2021, Pugh et al., 2016], which is challenging on its own [Balduzzi et al., 2019].

Acknowledgements

Thanks to Francesco Riccio for reviewing this work. Thanks to the whole Sony AI team for experiment infrastructure.

References

Hidehisa Akiyama, Masashi Tsuji, and Shigeto Aramaki. Learning evaluation function for decision making of soccer agents using learning to rank. In *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 239–242. IEEE, 2016.

Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best arm identification in multi-armed bandits. In *COLT*, pages 41–53. Citeseer, 2010.

David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. Re-evaluating evaluation. *Advances in Neural Information Processing Systems*, 31, 2018.

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pages 434–443. PMLR, 2019.

Nolan Bard, John Hawkin, Jonathan Rubin, and Martin Zinkevich. The annual computer poker competition. *AI Magazine*, 34(2):112–112, 2013.

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

Sebastian Bruch. An alternative cross entropy loss for learning-to-rank. In *Proceedings of the Web Conference 2021*, pages 118–126, 2021.

Abraham Charnes and William W Cooper. Chance-constrained programming. *Management science*, 6(1): 73–79, 1959.

Katherine Chen and Michael Bowling. Tractable objectives for robust policy optimization. *Advances in Neural Information Processing Systems*, 25:2069–2077, 2012.

S.E. Embretson, S.E. Embretson, and S.P. Reise. *Item Response Theory for Psychologists*. Multivariate applications book series. L. Erlbaum Associates, 2000. ISBN 9780805828184.

Meta FAIR, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624): 1067–1074, 2022.

Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Super-human performance in Gran Turismo Sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, 2021. doi: 10.1109/LRA.2021.3064284.

Daniel Hernandez, Kevin Denamganai, Sam Devlin, Spyridon Samothrakis, and James Alfred Walker. A comparison of self-play algorithms under a generalized framework. *IEEE Transactions on Games*, 14(2):221–231, 2021.

Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 368–377, 2018.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1791–1799, 2019.

Xiangyu Liu, Hangtian Jia, Ying Wen, Yujing Hu, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Yaodong Yang. Towards unifying behavioral and response diversity for open-ended learning in zero-sum games. *Advances in Neural Information Processing Systems*, 34:941–952, 2021.

Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. In *IJCAI 2019*, 2019a.

Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. *arXiv preprint arXiv:1903.05614*, 2019b.

Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *International Conference on Machine Learning*, pages 7480–7491. PMLR, 2021.

Stephen McAleer, Kevin Wang, John B Lanier, Marc Lanctot, Pierre Baldi, Tuomas Sandholm, and Roy Fox. Anytime psro for two-player zero-sum games. 2022.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific reports*, 9(1):1–29, 2019.

Harrie Oosterhuis and Maarten de de Rijke. Robust generalization and safe query-specializationin counterfactual learning to rank. In *Proceedings of the Web Conference 2021*, pages 158–170, 2021.

Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, page 40, 2016.

Arrasy Rahman, Elliot Fosong, Ignacio Carlucho, and Stefano V Albrecht. Towards robust ad hoc teamwork agents by creating diverse training teammates. *arXiv preprint arXiv:2207.14138*, 2022.

Mark Rowland, Shayegan Omidshafiei, Karl Tuyls, Julien Perolat, Michal Valko, Georgios Piliouras, and Remi Munos. Multiagent evaluation under incomplete information. *NeurIPS*, 32, 2019.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Dürr, and Davide Scaramuzza. Autonomous Overtaking in Gran Turismo Sport Using Curriculum Reinforcement Learning. In *ICRA*, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Oskari Tammelin. Solving large imperfect information games using cfr+. *arXiv preprint arXiv:1407.5042*, 2014.

Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michaël Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents. *ArXiv Pre-Print*, abs/2107.12808, 2021. URL `https://arxiv.org/abs/2107.12808`.

Wim J. van der Linden. *Linear models for optimal test design*. Springer, 2005.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602 (7896):223–228, 2022.