Appendix

A. Other Normalization Methods

A simple normalization technique is to map scores on an environment j that are in the range $[a_j, b_j]$ to $[0, 1]$, i.e., $g(x, j) \coloneqq (x - a_j)/(b_j - a_j)$ (Bellemare et al., 2013). However, this can result in normalized performance measures that cluster in different regions of $[0, 1]$ for each environment. For example, consider one environment where a the minimum is -100, the maximum is 10 and a uniform random policy can score around 0. Similarly consider a second environment where the minimum score is 10, maximum is 1,000, and random gets around 20. On the first environment, algorithms will tend to have a normalized performance near 1 and in the second case most algorithms will have a normalized performance near 0. So in the second environment algorithms will likely appear worse than algorithms in the first regardless of how close to optimal they are. This means the normalized performances are not really comparable.

A different version of this approach uses the minimum and maximum mean performance of each algorithm (Bellemare et al., 2013; Balduzzi et al., 2018). Let $\hat{\mu}_{i,j}$ be the sample mean of $X_{i,j}$. Then this normalization method uses the following function, $\bar{g}(i, j) \coloneqq (\hat{\mu}_{i,j} - \min_{i'} \hat{\mu}_{i',j})/(\max_{i'} \hat{\mu}_{i',j} - \min_{i'} \hat{\mu}_{i',j})$. This sets the best algorithm's performance on each environment to 1 and the worst to 0, spreading the range of values out over the whole interval $[0, 1]$. This normalization technique does not correct for nonlinear scaling of performance. As a result algorithms could be near 0 or 1 if there is an outlier algorithm that does very well or poorly. For example, one could introduce a terrible algorithm that just chooses one action the whole time. This makes the environment seem easier as all scores would be near 1 except for this bad algorithm. We would like the evaluation procedure to be robust to the addition of poor algorithms.

An alternative normalization technique proposed by Whiteson et al. (2011) uses the probability that one algorithm outperforms another on an environment, j, i.e., $\Pr(X_{i,j} \geq X_{k,j})$. This technique is intuitive and straight forward to estimate but neglects the difference in score magnitudes. For example, consider that algorithm i always scores a 1.0 and algorithm k always scores 0.99, the probability that i is better than k is 1, but the difference between them is small, and the normalized score of 1.0 neglects this difference.

B. α-Rank and our Implementation

The α-Rank procedure finds a solution to a game by computing the stationary distribution of strategy profiles when each player is allowed to change their strategy. This is done by constructing a directed graph where nodes are pure strategies and edges have weights corresponding to the probability that one of the players switches strategies. This graph can be represented by a Markov matrix, $C \in [0, 1]^{|\mathcal{S}| \times |\mathcal{S}|}$. The entry $C_{s,s'}$ corresponds to a probability of switching from a strategy s to s'. Only one player is allowed to change strategies at a time, so the possible transitions for a strategy $s = (i, (j, k))$, are any strategies $s' = (i', (j, k))$ or $s' = (i, (j', k'))$ for all $i', k' \in \mathcal{A}$ and $j' \in \mathcal{M}$.

The typical α-Rank procedure uses transition probabilities, $C_{s,s'}$, that are based on a logistic transformation of the payoff difference $u_l(s') - u_l(s)$. These differences are scaled by a parameter α and as α approaches ∞, the transition matrix approximates the Markov Conley chain (MCC), which is the motivation for using α-Rank as a solution concept for games. See the work of Omidshafiei et al. (2019) for more detailed information. The entries of the matrix for valid transitions are:

$$C_{s,s'} = \begin{cases} \eta \frac{1 - \exp\left(-\alpha(u_l(s') - u_l(s))\right)}{1 - \exp\left(-\alpha n(u_l(s') - u_l(s))\right)} & \text{if } u_l(s') = u_l(s) \\ \frac{\eta}{n} & \text{if } u_l(s') = u_l(s) \end{cases} \text{ and } C_{s,s} = 1 - \sum_{s' \neq s} C_{s,s'},$$

where $\eta = (\sum_k |\mathcal{S}^k| - 1)^{-1}$, l represents the player who switched from strategy s to s', n is the population constant (we set it to 50 following the prior work). The equilibrium over strategies is then given by the stationary distribution d of the Markov chain induced by C, i.e., d is a distribution such that $d = d\tilde{C}$. The equilibrium solution p^*, q^* are then the sum of probabilities in d for each strategy, i.e., $p^*_{s_1} = \sum_{s_2 \in \mathcal{S}_2} d_{s_1,s_2}$ and $q^*_{s_2} = \sum_{s_1 \in \mathcal{S}_1} d_{s_1,s_2}$. The aggregate performance can then be computed using q^* as in (1).

Theoretically, the hyperparameter α could be chosen arbitrarily high and the matrix C would still be irreducible, i.e., for all $s \in \mathcal{S}$, $d_s > 0$ and d is unique. However, due to numerical precision issues, a high value of α sets transition probabilities to zero for some dominated strategies, i.e., $u_l(s') < u_l(s)$, which can result in a matrix that is reducible. The suggested method to chose α is to tune it on a logarithmic scaled to find the highest value such that the transition matrix, C, is still irreducible (Omidshafiei et al., 2019).

This strategy works when the payoffs are known, but when they represent empirical samples of performance, then the value of α chosen will depend on the empirical payoff functions. Setting α based solely on the empirical payoffs could introduce bias to the matrix based on that sample. So we need a different solution without a data dependent hyperparameter.

In the MCC graph construction, all edges leading to strategies with strictly greater payoffs have the same positive weight. All edges that lead to strategies with the same payoff have he same weight but less than that of the strictly greater payoff. There are no transitions to strategies with worse payoffs. As $\alpha \to \infty$ the transitions probabilities quickly saturate to η if $u_l(s') > u_l(s)$ and 0 if $u_l(s') < u_l(s)$. So we construct the transition matrix, C, differently using the saturation values to set the transition probabilities C is close to the MCC construction. The entries for C that represent valid transitions in the graph are:

$$C_{s,s'} \coloneqq \begin{cases} \eta & \text{if } u_l(s') > u_l(s) \\ \frac{\eta}{m} & \text{if } u_l(s') = u_l(s) \\ 0 & \text{otherwise} \end{cases} \quad C_{s,s} \coloneqq 1 - \sum_{s' \neq s} C_{s,s'}. \tag{2}$$

However, this often makes the transition matrix reducible, i.e., the stationary distribution might have mass on only one strategy. To ensure C is irreducible we follow the damping approach used in PageRank (Page et al., 1999), i.e., $\tilde{C} = \gamma C + (1-\gamma)(1/|\mathcal{S}|)$, where $\gamma \in (0,1)$ is a hyperparameter and $1-\gamma$ represents the probability of randomly switching to any strategy in $\mathcal{S}$. During a Monte-Carlo simulation of transitions through $\tilde{C}$, states will transition from s to s' according to C, but with probability $1-\gamma$ the transition ignores C and switches to some strategy $s' \in \mathcal{S}$ chosen uniformly at random.

For $\gamma = 1$ the matrix is unchanged and represents the MCC solution, but is reducible. For γ near one, the stationary distribution will be similar to the solution given by the MCC solution with high weight placed on dominate strategies and small weight given to weak ones. As $\gamma \to 0$ the stationary distribution becomes more uniform as it is only considering shorter sequences of transitions before a random switch occurs. This method differs from the infinite-α approach presented by Rowland et al. (2019), but in the limit as $\gamma \to 1$ and $\alpha \to \infty$, the solutions have small differences. The approach using γ has a benefit in that there is no data dependent hyperparameter and it has a simple interpretation.

We chose to set $\gamma = (|\mathcal{S}| - 1)/(|\mathcal{S}|)$ so that the expected number of transitions to occur before a random jump is $|\mathcal{S}|$. This allows for propagation of transition probabilities to cover every strategy combination. We could have chosen to set γ near one, e.g., $\gamma = 1 - 10^{-8}$, but this would make the computation of the confidence intervals take longer. This is because optimizing the C within confidence intervals $[C^-, C^+]$ (defined in the next section) is equivalent finding the optimal value function of a Markov decision process (MDP) with a discount parameter of γ. See the work of de Kerchove et al. (2007); Fercoq et al. (2013); Csáji et al. (2014) for more information on this connection. Solving and MDP with a discount γ near 1.0 causes the optimization process of value iteration and policy iteration to converge slower than if γ is small. So we chose γ such that it could still find solutions near the MCC solution, but remain computationally efficient.

Using this new definition of C we use the following alternative but equivalent method to compute the aggregate performance more efficiently (Fercoq et al., 2013):

$$y_i = \frac{1-\gamma}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} v(s) \quad v = (I - \gamma C)^{-1} R_i, \tag{3}$$

where $R_i \in \mathbb{R}^{|\mathcal{S}|}$ is a vector with entries $R_i(s) \coloneqq \mathbf{E}[F_{X_{k,j}}(X_{i,j})]$ with $s = (s_1, s_2)$ and $s_2 = (j,k)$. Notice that s_1 is ignored because i is already specified by R_i.

C. Confidence Intervals on the Aggregate Performance

In this section, we detail the PBP procedure for computing confidence intervals Y^- and Y^+ on the aggregate performance y and prove that they hold with high probability. That is, we show that for any confidence level $\delta \in (0, 0.5]$;

$$\Pr(\forall i \in \mathcal{A},\ y_i \in [Y_i^-, Y_i^+]) \geq 1 - \delta.$$

We will first describe the PBP procedure to compute confidence intervals and then prove that they valid. A list of the symbols used in the construction of confidence intervals and their description are provided in Table 3 to refresh the reader. The steps to compute the confidence intervals are outlined in Algorithm 1.

Symbol List	
Symbol	**Description**
$\mathcal{A}$	set of algorithms in the evaluation
$\mathcal{M}$	set of environments in the evaluation
$X_{i,j}$	random variable representing performance of algorithm i on environment j
$T_{i,j}$	number of samples of performance for algorithm i on environment j
$x_{i,j,t}$	the t^{th}sample of performance of algorithm i on environment j and sorted such that $x_{i,j,t-1} \leq x_{i,j,t}$
D	data set containing all samples of performance for each algorithm on each environment
$y \in \mathbb{R}^{\lvert\mathcal{A}\rvert}$	y_i is the aggregate performance for each algorithm i
$Y^-, Y^+ \in \mathbb{R}^{\lvert\mathcal{A}\rvert}$	lower and upper confidence intervals on y computed using D
$\delta \in (0, 0.5]$	confidence level for the aggregate performance
$F_{X_{i,j}}$	cumulative distribution function (CDF) of$X_{i,j}$ and is also used for normalization
$\hat{F}_{X_{i,j}}$	empirical cumulative distribution function constructed using samples $x_{i,j,\cdot}$
$F^-_{X_{i,j}}, F^+_{X_{i,j}}$	lower and upper confidence intervals on $F_{X_{i,j}}$ computed using D
$z_{i,j,k}$	performance of algorithm i, i.e., $z_{i,j,k} = \mathbf{E}[F_{X_{k,j}}(X_{i,j})]$
$Z^-_{i,j,k}, Z^+_{i,j,k}$	lower and upper confidence intervals on $z_{i,j,k}$ computed using D.
$s_1 \in \mathcal{S}_1$	strategy for player p where $\mathcal{S}_1 = \mathcal{A}$ and s_1 is often denoted using i
$s_2 \in \mathcal{S}_2$	strategy for player q where $\mathcal{S}_2 = \mathcal{M} \times \mathcal{A}$ and s_2 is often denoted using (j, k)
$s \in \mathcal{S}$	joint strategy where $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$, and $s = (s_1, s_2)$ is often denoted as $(i, (j, k))$
$p \in \Delta(\mathcal{S}_1)$	strategy for player p represented as a distribution over $\mathcal{S}_1$
$q \in \Delta(\mathcal{S}_2)$	strategy for player q represented as a distribution over $\mathcal{S}_2$
$u_p(s)$	payoff for player p when s is played, i.e.,$u_p(s) = \mathbf{E}[F_{X_{k,j}}(X_{i,j})]$
$u_q(s)$	payoff for player q when s is played, i.e., $u_q(s) = -u_p(s)$
$u^-_l(s), u^+_l(s)$	confidence intervals on $u_l(s)$ for player $l \in \{p, q\}$ computed using D

Table 3. List of symbols used to create confidence intervals on the aggregate performance.

Recall that the aggregate performance for an algorithm i is

$$y_i \coloneqq \sum_{j=1}^{|\mathcal{M}|} \sum_{k=1}^{|\mathcal{A}|} q^*_{j,k} \mathbf{E}[F_{X_{k,j}}(X_{i,j})],$$

where q^* is the equilibrium solution to the game specified in Section 4.2. To compute valid confidence intervals Y^-, Y^+ on y using a dataset D, the uncertainty of q^* and mean normalized performance $z_{i,j,k} = \mathbf{E}[F_{X_{k,j}}(X_{i,j})]$. PBP accomplishes this by three primary steps. The first step is to compute confidence intervals $Z^-_{i,j,k}, Z^+_{i,j,k}$ on $z_{i,j,k}$ such that

$$\Pr\left(\forall (i,j,k) \in \mathcal{A} \times \mathcal{M} \times \mathcal{A},\ z_{i,j,k} \in [Z^-_{i,j,k}, Z^+_{i,j,k}]\right) \geq 1 - \delta.$$

The second step is to compute the uncertainty set $\mathcal{Q}$ containing all possible q^* that are compatible with Z^- and Z^+. The last step is to compute the smallest and largest possible aggregate performances for each algorithm over these sets, i.e.,

$$Y^-_i = \min_{q \in \mathcal{Q}} \sum_{j=1}^{|\mathcal{M}|} \sum_{k=1}^{|\mathcal{A}|} q_{j,k} Z^-_{i,j,k} \text{ and } \quad Y^+_i = \max_{q \in \mathcal{Q}} \sum_{j=1}^{|\mathcal{M}|} \sum_{k=1}^{|\mathcal{A}|} q_{j,k} Z^+_{i,j,k}.$$

PBP follows this process, except in the last two steps $\mathcal{Q}$ is never explicitly constructed to improve computational efficiency. Intuitively, the procedure provides valid confidence intervals because all values to compute the aggregate performance depend on the normalized performance. So by guaranteeing with probability at least $1 - \delta$ that the true mean normalized performances will be between Z^- and Z^+, then so long as the the upper (lower) confidence interval computed is at least as large (small) as the maximum (minimum) of the aggregate score for any setting of $z \in [Z^-, Z^+]$, the confidence intervals will be valid.

Algorithm 1 Performance Bound Propagation (PBP)

Input: dataset D containing samples of performance and a confidence level $\delta \in (0, 0.5]$)
Output: Y^-, Y^+ confidence intervals on the aggregate performance

$\delta' \leftarrow \delta/(|\mathcal{A}||\mathcal{M}|)$;
$\texttt{sort\_ascending}(\{x_{i,j,t}\}_{t=1}^{T_{i,j}})$;
// Compute confidence intervals for the CDFs
for $i, j \in \mathcal{A} \times \mathcal{M}$ **do**
  $F^-_{X_{i,j}}, F^+_{X_{i,j}} \leftarrow \texttt{dkw\_bound}(\{x_{i,j,t}\}_{t=1}^{T_{i,j}}, \delta')$; // computation shown in (4)
end for
// Compute confidence intervals on the mean normalized performance
for $i, j, k \in \mathcal{A} \times \mathcal{M} \times \mathcal{A}$ **do**
  $Z^-_{i,j,k} \leftarrow F^-_{X_{k,j}}(x_{i,j,T}) - \sum_{t=0}^{T-1} \left[F^-_{X_{k,j}}(x_{i,j,t+1}) - F^-_{X_{k,j}}(x_{i,j,t})\right] F^+_{X_{i,j}}(x_{i,j,t})$;
  $Z^+_{i,j,k} \leftarrow F^+_{X_{k,j}}(x_{i,j,T+1}) - \sum_{t=1}^{T} \left[F^+_{X_{k,j}}(x_{i,j,t+1}) - F^+_{X_{k,j}}(x_{i,j,t})\right] F^-_{X_{i,j}}(x_{i,j,t})$;
end for
// Construct game quantities
$\mathcal{S} = \mathcal{A} \times (\mathcal{M} \times \mathcal{A})$; strategy profile set
$\gamma \leftarrow \frac{|\mathcal{S}|-1}{|\mathcal{S}|}$
$C^-, C^+ \leftarrow \texttt{bound\_markov\_matrix}(Z^-, Z^+)$ as defined in (7).
// Optimize aggregate performance over all possible $C \in [C^-, C^+]$
for $i \in \mathcal{A}$ **do**
  $v \leftarrow \texttt{find\_optimal\_valuefunction}(C^-, C^+, R_i = -Z^-_{i,\cdot,\cdot})$; // solve (8)
  $Y^-_i \leftarrow \frac{(1-\gamma)}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |v(s)|$
  $v \leftarrow \texttt{find\_optimal\_valuefunction}(C^-, C^+, R_i = Z^+_{i,\cdot,\cdot})$; // solve (8)
  $Y^+_i \leftarrow \frac{(1-\gamma)}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |v(s)|$
end for

We break the rest of this section into two subsections. The first subsection discusses constructing the confidence intervals on the mean normalized performance and proving their validity. The second subsection describes how to construct the confidence intervals on the aggregate performance proves their validity.

C.1. Confidence intervals on the normalized performance

The normalized performance $z_{i,j,k} = \mathbf{E}[F_{X_{k,j}}(X_{i,j})]$ has two unknowns, $F_{X_{k,j}}$ and the distribution of $X_{i,j}$. To compute confidence intervals on $z_{i,j,k}$ for all i, j, k, confidence intervals are needed on the output on all distribution functions $F_{X_{i,j}}$. The confidence intervals on the distributions can then be combined to get confidence intervals on $z_{i,j,k}$.

To compute confidence intervals on $F_{X_{i,j}}$ we assume that $X_{i,j}$ is bounded on the interval $[a_j, b_j]$ for all $i \in \mathcal{A}$ and $j \in \mathcal{M}$. Let $\hat{F}_{X_{i,j}}$ be the empirical CDF with

$$\hat{F}_{X_{i,j}}(x) \coloneqq \frac{1}{T_{i,j}} \sum_{t=1}^{T_{i,j}} \mathbf{1}_{x_{i,j,t} \leq x},$$

where $T_{i,j}$ is the number of samples of $X_{i,j}$, $x_{i,j,t}$ is the t^{th} sample of $X_{i,j}$, and $\mathbf{1}_A = 1$ if event A is true and 0 otherwise. Using the Dvoretzky–Kiefer–Wolfowitz (DKW) inequality (Dvoretzky et al., 1956) with tight constants (Massart, 1990), we

define $F^-_{X_{i,j}}$ and $F^+_{X_{i,j}}$ to be the lower and upper confidence intervals on $F_{X_{i,j}}$, i.e.,

$$F^+_{X_{i,j}}(x) \coloneqq \begin{cases} 1 & \text{if } x \geq b \\ \min(1.0, \hat{F}_{X_{i,j}}(x) + \epsilon) & \text{if } a \leq x < b \\ 0 & \text{if } x < a \end{cases} \qquad \text{and } \epsilon = \sqrt{\frac{\ln \frac{2}{\delta'}}{2T_{i,j}}}, \tag{4}$$

$$F^-_{X_{i,j}}(x) \coloneqq \begin{cases} 1 & \text{if } x \geq b \\ \max(0.0, \hat{F}_{X_{i,j}}(x) - \epsilon) & \text{if } a \leq x < b \\ 0 & \text{if } x < a \end{cases}$$

where $\delta' \in (0, 0.5]$ is a confidence level and we use $\delta' = \delta/(|\mathcal{A}||\mathcal{M}|)$. By the DKW inequality with tight constants the following property is known:

Property 1 (DKW with tight constants confidence intervals)**.**

$$\Pr\left(\forall x \in \mathbb{R},\ \ F_{X_{i,j}}(x) \in [F^-_{X_{i,j}}(x), F^+_{X_{i,j}}(x)]\right) \geq 1 - \delta'.$$

Proof. See the works of Dvoretzky et al. (1956) and Massart (1990). □

Further, by the union bound we have that

$$\Pr\left(\forall i \in \mathcal{A}, \forall j \in \mathcal{M}, \forall x \in \mathbb{R},\ \ F_{X_{i,j}}(x) \in [F^-_{X_{i,j}}(x), F^+_{X_{i,j}}(x)]\right) \geq 1 - \delta. \tag{5}$$

To construct confidence intervals on the mean normalized performance, we will use Anderson's inequality (Anderson, 1969). Let X be a bounded random variable on $[a, b]$, with sorted samples $x_1 \leq x_2 \leq \cdots \leq x_T$, $x_0 = a$, and $x_{T+1} = b$. Let $g\colon \mathbb{R} \to \mathbb{R}$ be a monotonically increasing function. Anderson's inequality specifies for a confidence level $\delta \in (0, 0.5]$ the following high confidence bounds on $\mathbf{E}[g(X)]$:

$$\begin{aligned} \mathbf{E}[g(X)] &\geq g(x_T) &&- \sum_{t=0}^{T-1} [g(x_{t+1}) - g(x_t)]\, F^+_X(x_t) \\ \mathbf{E}[g(X)] &\leq g(x_{T+1}) &&- \sum_{t=1}^{T} [g(x_{t+1}) - g(x_t)]\, F^-_X(x_t), \end{aligned}$$

where $F^{+/-}_X$ uses the DKW inequality with tight constants and as defined in (4).

Anderson's inequality can be used to bound the mean normalized performance since $F_{X_{k,j}}$ is a monotonically increasing function and $\delta \in (0, 0.5]$. Since $F_{X_{k,j}}$ is unknown, we replace g in Anderson's inequality with $F^-_{X_{k,j}}$ for the lower bound and $F^+_{X_{k,j}}$ for the upper bound. This gives the following confidence intervals for $z_{i,j,k}$:

$$\begin{aligned} Z^-_{i,j,k} &= F^-_{X_{k,j}}(x_{i,j,T}) &&- \sum_{t=0}^{T-1} \left[F^-_{X_{k,j}}(x_{i,j,t+1}) - F^-_{X_{k,j}}(x_{i,j,t})\right] F^+_{X_{i,j}}(x_{i,j,t}) \\ Z^+_{i,j,k} &= F^+_{X_{k,j}}(x_{i,j,T+1}) &&- \sum_{t=1}^{T} \left[F^+_{X_{k,j}}(x_{i,j,t+1}) - F^+_{X_{k,j}}(x_{i,j,t})\right] F^-_{X_{i,j}}(x_{i,j,t}), \end{aligned} \tag{6}$$

where $T = T_{i,j}$, $x_{i,j,0} = a_j$, and $x_{i,j,T+1} = b_j$. We now prove the following lemma:

Lemma 1. *If Z^- and Z^+ are computed by (6), then:*

$$\Pr\left(\forall i, k \in \mathcal{A}, \forall j \in \mathcal{M},\ \ z_{i,j,k} \in [Z^-_{i,j,k}, Z^+_{i,j,k}]\right) \geq 1 - \delta.$$

Proof. By Anderson's inequality we know that $Z^+_{i,j,k}$ is an high confidence upper bound on $\mathbf{E}[F^+_{X_{k,j}}(X_{i,j})]$ and similarly $Z^-_{i,j,k}$ is a high confidence lower bound on $\mathbf{E}[F^+_{X_{k,j}}(X_{i,j})]$, i.e.,

$$\Pr\left(\mathbf{E}[F^-_{X_{k,j}}(X_{i,j})] \geq Z^-_{i,j,k}\right) \geq 1 - \delta'/2$$
$$\Pr\left(\mathbf{E}[F^+_{X_{k,j}}(X_{i,j})] \leq Z^+_{i,j,k}\right) \geq 1 - \delta'/2.$$

By Property 1 we know that $\Pr\left(\forall x \in \mathbb{R},\ \ F_{X_{k,j}}(x) \in [F^-_{X_{k,j}}(x), F^+_{X_{k,j}}(x)]\right) \geq 1 - \delta'$, thus

$$\Pr\left(\forall i, k \in \mathcal{A}, \forall j \in \mathcal{M},\ \ z_{i,j,k} \in [Z^-_{i,j,k}, Z^+_{i,j,k}]\right) \geq 1 - 2\delta',$$

where $2\delta'$ comes from combining the failure rates of confidence intervals on the CDFs $F_{X_{i,j}}$ and $F_{X_{k,j}}$. The confidence intervals on the mean normalized performances can only fail if the confidence intervals on CDFs fail. As stated in (5), all confidence intervals on the CDFs contain the true CDFs with probability at least $1 - \delta$. Thus, all mean normalized performances hold with probability at least $1 - \delta$. □

The confidence intervals given by Z^- and Z^+ are guaranteed to hold for $T \geq 1$, and $\delta \in (0, 0.5]$, but are often conservative requiring a large number samples to identify a statistically meaningful result. So we empirically test alternatives that have either stricter assumptions or weaker theoretical justification.

C.2. Confidence intervals on the aggregate performance

In this section we will provide details on how to compute the confidence intervals on the aggregate performance using the confidence intervals on the mean normalized performance and then prove that they hold with high confidence. To construct confidence intervals and prove their validity we will make the following steps. First, we show that for a fixed weighting q that valid confidence intervals can be computed directly by using interval arithmetic. Second, we describe how to characterize the uncertainty of the game. Third, we make a connection between aggregate performance using the equilibrium solution q^* and the optimal average reward for a Markov decision process. Lastly we describe an optimization procedure for computing the optimal average reward, which corresponds to finding the lower and upper confidence intervals.

Before discussing how to bound the aggregate performance using the game theoretic solution, consider the case when weights q can be any probability distribution over algorithms and environments chosen before the experiment begins. Let weights $q \in [0,1]^{|\mathcal{A}|\times|\mathcal{M}|}$, such that $\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} = 1$. Let $\tilde{y}_i = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} z_{i,j,k}$ be the aggregate performance for an algorithm $i \in \mathcal{A}$. The corresponding confidence intervals for $\tilde{y}_i$ are $\tilde{Y}_i^- = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} Z^-_{i,j,k}$ and $\tilde{Y}_i^+ = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} Z^+_{i,j,k}$.

Lemma 2. *If weights q are independent of the data D, then:*

$$\Pr\left(\forall i \in \mathcal{A},\ \tilde{y}_i \in [\tilde{Y}_i^-, \tilde{Y}_i^+]\right) \geq 1 - \delta$$

Proof. Applying the result of Lemma 1, all confidence intervals produced by $Z^-_{i,j,k}$ and $Z^+_{i,j,k}$ contain $z_{i,j,k}$ with probability $1 - \delta$. So interval arithmetic can be used to compute confidence intervals on the aggregate performance without changing the probability of failure, i.e., $\tilde{Y}_i^- = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} Z^-_{i,j,k}$ and $\tilde{Y}_i^+ = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{A}} q_{j,k} Z^+_{i,j,k}$ for each algorithm $i \in \mathcal{A}$. With these intervals then

$$\begin{aligned}\Pr\left(\forall i \in \mathcal{A},\ \tilde{y}_i \in [\tilde{Y}_i^-, \tilde{Y}_i^+]\right) &\geq 1 - \Pr\left(\bigcup_{i,k\in\mathcal{A},j\in\mathcal{M}} z_{i,j,k} \notin [Z^-_{i,j,k}, Z^+_{i,j,k}]\right) \\ &\geq 1 - \delta.\end{aligned}$$

□

This method of aggregating performance highlights how the uncertainty of normalized scores propagates to the confidence intervals of the aggregate performance for a fixed weighting. Next we will show how to compute confidence intervals on y when using the dynamic weighting produce by the equilibrium solution to the two player game.

Instead of considering all possible equilibrium solutions q^*, recall that y_i depends on the Markov matrix C as shown in (3). To construct confidence intervals on y_i the uncertainty in creating the matrix C as defined in (2) needs to be considered (Rowland et al., 2019). The definition of C assumes certainty of payoffs of each strategy, but the empirical payoffs have uncertainty corresponding to Z^- and Z^+, i.e., for $s = (i, (j, k))$, $u_p^-(s) = Z_{i,j,k}^-$, $u_p^+(s) = Z_{i,j,k}^+$, $u_q^-(s) = -Z_{i,j,k}^+$, $u_q^+(s) = -Z^- i, j, k$. As a result, when the payoff confidence intervals overlap for two strategies s and s' this creates uncertainty in C. We define $C^-, C^+ \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$ as the lower and upper confidence intervals on C with entries

$$C_{s,s'}^-, C_{s,s'}^+ \coloneqq \begin{cases} (\eta, \eta) & \text{if } u_l^-(s') > u_l^+(s) \\ (0,0) & \text{if } u_l^-(s) > u_l^+(s') \\ (\frac{\eta}{n}, \frac{\eta}{n}) & \text{if } u_l^{+/-}(s) = u_l^{+/-}(s') \\ (0, \eta) & \text{otherwise} \end{cases} \quad \forall s' \in \mathcal{S} \setminus \{s\}$$
$$C_{s,s}^-, C_{s,s}^+ \coloneqq \left(1 - \sum_{s' \neq s} C_{s,s'}^+,\ 1 - \sum_{s' \neq s} C_{s,s'}^-\right) \tag{7}$$

To get the bounds on the aggregate performance over all possible C in these intervals the uncertainty of the stationary distribution on $\tilde{C} = \gamma C + (1-\gamma)(1/|\mathcal{S}|)$ has to be consider. This can be accomplished by first computing the minimum and maximum values of the stationary distribution for each strategy (Rowland et al., 2019) and then finding the minimum and maximum aggregate performances for all possible stationary distributions in this limits. However, individually computing these two quantities leads to looser bounds than directly estimating the minimum and maximum aggregate performance over all possible C because it ignores the correlations in the confidence intervals of the stationary distribution. To compute the minimum and maximum aggregate performance over all possible C, we need to make a connection between the average performance using the stationary distribution of $\tilde{C}$ and the average performance before termination on C.

Let d be a stationary distribution over strategy profiles $\mathcal{S}$ induce by the Markov matrix $\tilde{C} = \gamma C + (1-\gamma)(1/|\mathcal{S}|)$. Let q be the distribution of strategies for player q contained in d.

Lemma 3. *The following are equivalent*

$$y_i = \sum_{(j,k)\in\mathcal{S}_2} q_{j,k} z_{i,j,k}$$
$$y_i = \frac{1-\gamma}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} v(s),$$

where $v = (I - \gamma C)^{-1} R_i$, $R_i \in \mathbb{R}^{|\mathcal{S}|}$ *such that* $R_i(s) = z_{i,j,k}$ *for* $s = (\cdot, (j,k))$.

Proof. We know that $q_{j,k} = \sum_{i\in\mathcal{S}_1} d(i,(j,k))$. This implies $y_i = \sum_{(j,k)\in\mathcal{S}_2} q_{j,k} z_{i,j,k} = \sum_{s\in\mathcal{S}} d(s) R_i(s)$. Applying 8.2.12 of Puterman (1994), we have the relation

$$v + y_i = R_i + \tilde{C} v$$
$$v + y_i = R_i + \gamma C v + \frac{(1-\gamma)}{|S|} \mathbf{1}^\top v,$$

where here $\mathbf{1} \in \mathbb{R}^{|\mathcal{S}|}$ is a vector of all ones. Then for $v = (I - \gamma C)^{-1} R_i$, $y_i = \frac{(1-\gamma)}{|S|} \mathbf{1}^\top v$. □

Finally, using techniques developed by Fercoq et al. (2013), the lower and upper bounds of the aggregate performance y_i can be bound by solving the following optimization problem:

$$\begin{aligned} \min_{v} \quad & \sum_{s\in\mathcal{S}} v(s) \\ \text{s.t.} \quad & v(s) \geq R_i(s) + \gamma \sum_{s'\in\mathcal{S}} C_{s,s'} v(s') \\ & C^- \leq C \leq C^+ \\ & \sum_{s'\in\mathcal{S}} C_{s,s'} = 1 \quad \forall s \in \mathcal{S}, \end{aligned} \tag{8}$$

where C is a free variable, $R_i(s) = -Z^-_{i,j,k}$, and $R_i(s) = Z^+_{i,j,k}$ are used to obtain the lower and upper bounds on y_i, respectively. Both bounds are computed as $Y_i^{+/-} = (1-\gamma)(1/|\mathcal{S}|)\sum_{s\in\mathcal{S}}|v(s)|$ using the respective solutions v. The absolution value of v is taken to account for the negativity of $R_i = -Z^-$. We compute the solution v in polynomial time using policy iteration based approach similar to Fercoq et al. (2013), but modify their algorithm to fit our matrix C. We detail our exact method in Appendix D.

Now to prove the main result we will use the previous lemmas in connection with PBP.

Theorem 1. *If* $(Y^-, Y^+) = PBP(D, \delta)$*, then*

$$\Pr\left(\forall i \in 1, 2, \ldots, |\mathcal{A}|,\ y_i \in [Y_i^-, Y_i^+]\right) \geq 1 - \delta.$$

Proof. From Lemma 1 we know that Z^- and Z^+ are valid $1-\delta$ confidence intervals on z. Thus, applying Lemma 2 we know that a valid $1-\delta$ confidence intervals can be computed by a weighted sum of lower and upper bounds Z^- and Z^+, for any joint probability distribution q over environments and algorithms. Through Lemma 3 this is equivalent to assuming fixed Markov matrix C. The true matrix C is unknown, so the minimum and maximum intervals need to be found over a set $\mathcal{C}$ that contains all transition matrices that are compatible with Z^- and Z^+. Let $\mathcal{C} = \prod_{s\in\mathcal{S}}\mathcal{C}_s$, where $\mathcal{C}_s$ is the polytope of transition probabilities starting from strategy profile $s \in \mathcal{S}$, i.e.,

$$\mathcal{C}_s = \{C_{s,\cdot} : \forall s' \in \mathcal{S},\ C_{s,s'} \in [C^-_{s,s'}, C^+_{s,s'}], \sum_{s'\in\mathcal{S}} C_{s,s'} = 1\}.$$

The minimum and maximum aggregate values Y^- and Y^+ computed over $\mathcal{C}$ can be found in polynomial time using linear programming, value iteration, or policy iteration (Fercoq et al., 2013). PBP finds the minimum and maximum confidence intervals over all $C \in \mathcal{C}$, thus, they represent valid $1-\delta$ confidence intervals for each algorithm. □

D. Policy Iteration for Bounding the Aggregate Performance

This section we detail our approach for optimizing the aggregate performance over the uncertainty of the Markov matrix C to compute confidence intervals. Recall that the upper and lower high-confidence bounds on the aggregate performance for algorithm i can be found by solving the optimization problem in 8. Alternatively one can use a Dynamic Programming approach either using value iteration or policy iteration for more efficient optimization (Fercoq et al., 2013). Using value iteration has better scaling to the size of the state space in the optimization problem than policy iteration. However, we found that in small and moderate sized problems policy iteration was sufficient and used it in our experiments.

Our method is similar to that of Fercoq et al. (2013, Algorithm 1), except that we use policy iteration instead of value iteration and a modification to the dynamic programming operator since the transition probabilities in our problem do not depend on the number of out going edges. Algorithm 2 shows pseudocode of the policy iteration to find the the optimal value function v^* through the modification of the matrix $C \in \mathcal{C}$. The algorithm takes as input lower and upper bounds $C^-, C^+ \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ on C, a reward function $R \in \mathbb{R}^{|\mathcal{S}|}$, a discount factor $\gamma \in (0,1)$, a tolerance on the distance to optimal, $tol \in \mathbb{R}$, and the maximum number of iterations $max\_itrs \in \mathbb{N}^+$. We set $\gamma = (1-|\mathcal{S}|)/|\mathcal{S}|$ and $tol = 1\times10^{-7}$, $max\_itrs = 400$ as default parameters. When finding the upper confidence interval, y_i^+, for algorithm i, R is a vector such that $R(s) = Z^+_{i,j,k}$ for all $s = (\cdot,(j,k)) \in \mathcal{S}$. Similarly, when finding y_i^-, R is such that $R(s) = -Z^-_{i,j,k}$.

Algorithm 2 has two main steps. First, greedily optimize C with respect to the current value function v, i.e., for all $s \in \mathcal{S}$

$$C_s = \arg\max_{C_s\in\mathcal{C}_s} R(s) + \gamma C_s v.$$

We provide pseudocode for this step in Algorithm 3. The second step is a value function update, which we compute exactly by solving the system of linear equations $v = R + \gamma C v$, by setting $v = (I-\gamma C)^{-1}R$. These steps repeat until C stops changing or a bound on the maximum absolute error in aggregate performance is below some threshold. The confidence interval on the aggregate performance is then returned as $(1-\gamma)/|\mathcal{S}|\sum_{s\in\mathcal{S}}|v(s)|$. Since C is sparse optimization to the code can be made to drastically speed up computation when $\mathcal{S}$ is large. We make some of these modifications in our implementation.

Due to small numerical errors this version of policy iteration may not keep the same policy, C, between successive iterations when at the optimal solution. To ensure that the procedure stops in a reasonable time and closely approximates the true

Algorithm 2 Policy Iteration for aggregate performance optimization

Input: $C^-, C^+ \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$, $R \in \mathbb{R}^{|\mathcal{S}|}$, $\gamma \in (0,1) = (1-|\mathcal{S}|)/|\mathcal{S}|$, $tol \in \mathbb{R} = 1\times10^{-7}$, $max\_itrs \in \mathbb{N}^+ = 400$
Initialize: $C \leftarrow C^-$, $v \leftarrow \mathbf{0} \in \mathbb{R}^{|\mathcal{S}|}$
$changed \leftarrow$ `True`
$iteration \leftarrow 0$
while changed **do**
  $changed \leftarrow$ `False`
  $iteration \leftarrow iteration + 1$
  if $iteration \geq max\_itrs$ **then**
    `break`
  end if
  for $s \in \mathcal{S}$ **do**
    $C'_s \leftarrow$ `update_transition_row`$(C^-_s, C^+_s, R(s), v, \gamma)$
    if $||C_s - C'_s||_\infty \geq 1\times10^{-8}$ **then**
      $changed \leftarrow$ `True`
      $C_s \leftarrow C'_s$
    end if
  end for
  $v' \leftarrow (I-\gamma C)^{-1}R$
  $\epsilon \leftarrow ||v - v'||_\infty$ // max change in value function
  $\epsilon_{v^*} \leftarrow (2\epsilon\gamma)/(1-\gamma)$ // error bound on distance to v^*
  $\epsilon_{aggregate} \leftarrow (1-\gamma)\epsilon_{v^*}$ // bound on the maximum error to the aggregate performance
  if $\epsilon_{aggregate} < tol$ **then**
    $changed \leftarrow$ `False`
  end if
  $v \leftarrow v'$
end while
Return: $(1-\gamma)$`mean`$(|v|)$

solution we employ three techniques: an iteration limit, halting computation when C is ϵ-close to between iterations, and stopping when a bound on the distance to true solution is below a tolerance, tol. The first two approaches are straight forward an in the pseudocode. To bound the distance to the true solution we leverage prior work on bounding the distance of the current value function to the optimal value function. Consider the value function v and the subsequent value function v' obtained after one application of the Bellman operator. In our problem the Bellman operator is

$$v(s) = \max_{C_s \in \mathcal{C}_s} R(s) + \gamma \sum_{s' \in \mathcal{S}} C_{s,s'} v(s').$$

Let $\epsilon = \max_{s\in\mathcal{S}} |v(s) - v'(s)|$. Then the distance $\epsilon_{v^*} = \max_{s\in\mathcal{S}} |v'(s) - v^*(s)|$ of v' to the optimal value function v^*, is bounded above by ϵ (Williams & Baird, 1993), i.e,

$$\epsilon_{v^*} \leq (2\epsilon\gamma)/(1-\gamma).$$

We translate this this bound into a bound on the error to the confidence interval of y_i. Let y_i be the confidence interval computed using v' and y_i^* be the confidence interval computed using v^*. Then an upper bound on the error $\epsilon_{aggregate} = |y_i^* - y_i|$ is

Algorithm 3 `update_transition_row` procedure for updating C_s

Input: $C_s^-, C_s^+ \in \mathbb{R}^{|\mathcal{S}|}$, $r \in \mathbb{R}$, $v \in \mathbb{R}^{|\mathcal{S}|}$, $\gamma \in (0, 1)$
Initialize: $C_s \leftarrow C_s^-$, $c \leftarrow \texttt{sum}(C_s^-)$
$w \leftarrow r + \gamma v$
$idxs \leftarrow \texttt{argsort}(w, \texttt{direction} = decreasing)$
for $i = 1$ **to** $|\mathcal{S}|$ **do**
  $idx \leftarrow idxs[i]$
  $\Delta c \leftarrow \min(C_s^+ - C_s^-, 1 - c)$
  $C_s[idx] \leftarrow C_s[idx] + \Delta c$
  $c \leftarrow c + \Delta c$
end for
Return: C_s

$$
\begin{aligned}
\epsilon_{aggregate} &= |y_i^* - y_i| \\
&= \left| \frac{1-\gamma}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |v^*(s)| - \frac{1-\gamma}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |v'(s)| \right| \\
&= \left| \frac{1-\gamma}{|\mathcal{S}|} (||v^*||_1 - ||v'||_1) \right| \\
&\leq \frac{1-\gamma}{|\mathcal{S}|} ||v^* - v'||_1 \\
&= \frac{1-\gamma}{|\mathcal{S}|} |\mathcal{S}| ||v^* - v'||_\infty \\
&= (1-\gamma) ||v^* - v'||_\infty \\
&\leq (1-\gamma) \max_{s \in \mathcal{S}} |v^*(s) - v'(s)| \\
&= (1-\gamma) \epsilon_{v^*} = 2\epsilon\gamma.
\end{aligned}
$$

E. Algorithm Definitions

This section provides the complete definition for each algorithm used in the experiments. Each algorithm is made complete by defining a distribution from, which hyperparameters are sampled. Table 4 shows distributions or values for any hyperparameter used in this work. For the continue state space environments, all of the algorithms use Fourier basis and linear function approximation (Konidaris et al., 2011). Note that $U(a, b)$ indicates a uniform random variable on $[a, b)$, and $U(\{\ldots\})$ indicates that a variable is sampled uniformly at random from a set of finite values.

Note that these ranges should not be considered optimal and could easily be improved for the environments in this experiment. The ranges were chosen to reflect a lack of knowledge about what optimal settings on an environment are and to be reflective of ranges one might expect optimal hyperparameters to fall in. The manual setting of these ranges leaks information based on our own experience with the algorithms and environments. However, since the definition is completely specified on these environments any favor to one algorithm could be easily identified and test for. Furthermore, any adaptive algorithm that can adjust these parameters during learning is likely to be superior than specifying better ranges through experience in this exact setup. Still, one should not tune ranges to fit any given set of environments.

F. Environments

This section describes the environments used in the experiments. All environments are listed in Table 5. Environments were recreated in the Julia language and many implementations follow closely to the that in the RLPy repository (Geramifard et al., 2015). Most environments are best described by either the paper publishing the environment or by examining the source code we provide. We also describe the discrete environments below.

There are eight discrete domains used in the work four Gridworld environments and four chain environments. The Gridworld

Algorithm	Hyperparameter	Discrete	Continuous
All	λ	$U(0, 1)$	
All	γ	$\Gamma - e^{U(\ln 10^{-4}, \ln 0.05)}$	
All	Value function	Tabular	Linear with Fourier basis
All Sarsa(λ) and Q(λ)	ϵ	$U(0, 1)$	
Sarsa(λ) and Q(λ)	α_q	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-6}, \ln 10^{-3})}$
Sarsa(λ)-s and Q(λ)-s	α_q	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-3}, \ln 10^{0})/\lvert\phi\rvert}$
All AC, PPO, and NACTD	Policy	Tabular Softmax	Linear Softmax with Fourier basis
AC, NACTD	α_v	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-6}, \ln 10^{-3})}$
AC, NACTD	α_p	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-6}, \ln 10^{-3})}$
AC-S	α_v	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-3}, \ln 10^{0})/\lvert\phi\rvert}$
AC-S, AC-Parl2	α_p	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-3}, \ln 10^{0})/(\lvert\phi\rvert \times \text{num\_actions})}$
NAC-TD	α_w	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-6}, \ln 10^{-3})}$
NAC-TD	normalize_gradient	True	
PPO	clip	$U(0.1, 0.3)$	
PPO	entropy_coef	$e^{U(\ln 10^{-8}, \ln 10^{-2})}$	
PPO	steps_per_batch	$2^{U(\log_2 64, \log_2 256)}$	
PPO	epochs	$U(\{1, \ldots, 10\})$	
PPO	batch_size	$2^{U(\log_2 16, \log_2 \min(64, \text{steps\_per\_batch}))}$	
PPO	Adam-ϵ	10^{-5}	
PPO	Adam-α	$e^{U(\ln 10^{-3}, \ln 10^{-1})}$	$e^{U(\ln 10^{-6}, \ln 10^{-3})}$
PPO	Adam-β_1	0.9	
PPO	Adam-β_2	0.999	
Fourier basis	dorder	N/A	$U(\{0, \ldots, 9\})$
Fourier basis	iorder	N/A	$U(\{1, \ldots, 9\})$
Fourier basis	trig	N/A	cos

Table 4. This table show the distributions from which each hyperparameter is sampled. The All algorithm means the hyperparameter and distribution were used for all algorithms. Steps sizes are labeled with various αs. The discount factor γ an algorithm uses is scaled down from Γ that is specified by the environment. For all environments used in this work $\Gamma = 1.0$. PPO uses the same learning rate for both the policy and value function. The max dependent order on the Fourier basis is limited such that no more than 10,000 features are generated as a result of dorder.

Environment	Num Episodes	State Space
Gridworld 5 Deterministic	100	Discrete
Gridworld 5 Stochastic	100	Discrete
Gridworld 10 Deterministic	100	Discrete
Gridworld 10 Stochastic	100	Discrete
Chain 10 Deterministic	100	Discrete
Chain 10 Stochastic	100	Discrete
Chain 50 Deterministic	100	Discrete
Chain 50 Stochastic	100	Discrete
Acrobot	500	Continuous
Cart-Pole	100	Continuous
MountainCar	100	Continuous
PinBall Empty	100	Continuous
PinBall Box	100	Continuous
Pinball Medium	100	Continuous
PinBall Single	200	Continuous

Table 5. This table list every used in this paper along with the number of episodes each algorithm was allowed to interact with the environment and its type of state space.

environments are an $N \times N$ grid with the agent starting every episode in the top left corner and goal state in the bottom right. The reward is -1 at every step until the goal state is reached, then the episode is then terminated. In every state there are four actions: up, down, left, and right. The transition dynamics are either deterministic, meaning an up action sends the agent up one state unless it is outside the map, or the dynamics are stochastic, meaning the agent might randomly move to one of the states perpendicular to the intended direction or stays in the current state. The chain environments are N chains where there are N states each with a connection only to the state directly to the left or right of it and end points only connect to the one state they are next to. The agent starts in state one (far left of the chain) and the goal state is state N (far right of the chain). The reward is -1 every step until the goal state is reached and then the episode terminates. Both gridworld and chain MDPs terminate episodes in a finite time based on the size of the problem. Gridworld problems terminate after $20N^2$ steps have been taken and chain environments terminate after $20N$ steps are taken.

G. Alternative Bounding Techniques

As pointed out in our description of PBP, we use the nonparametric concentration inequalities DKW and Anderson's inequalities. These inequalities are often conservative and lead to conservative confidence intervals. So we investigate two alternatives PBP-t a method that replaces DKW and Anderson's inequality with the one based on Students t-distribution and the percentile bootstrap.

In PBP-t everything is the same as PBP except we compute confidence intervals on the mean normalized performance as follows

$$Z^-_{i,j,k} = \mu_{i,j,k} - \frac{\hat{\sigma}}{\sqrt{T_{i,j}}} t_{1-\delta', T_{i,j}-1}$$
$$Z^+_{i,j,k} = \mu_{i,j,k} + \frac{\hat{\sigma}}{\sqrt{T_{i,j}}} t_{1-\delta', T_{i,j}-1}$$

where $z_{i,j,k,t} = \hat{F}_{X_{k,j}}(x_{i,j,t})$, $\mu_{i,j,k} = \frac{1}{T_{i,j}} \sum_{t=1}^{T_{i,j}} z_{i,j,k,t}$, $\hat{\sigma} = \sqrt{\sum_{t=1}^{T_{i,j}} (\mu_{i,j,k} - z_{i,j,k,t})^2 / (T_{i,j} - 1)}$, $t_{1-\delta',\nu}$ is the $100(1-\delta')$ percentile of Student's t-distribution with ν degrees of freedom, and we set $\delta' = \delta/(|\mathcal{A}||\mathcal{M}|)$. Notice that there are $|\mathcal{A}|^2|\mathcal{M}|$ comparisons being made, so $\delta' = \delta/(|\mathcal{A}|^2|\mathcal{M}|)$ should be used to account for more the multiple comparisons. However, it is likely that if one comparison with an algorithm i fails then that there will be failures with the other $|\mathcal{A}| - 1$ algorithms so we use the smaller δ' as a heuristic for tighter confidence intervals.

In the bootstrap procedure, we use the percentile bootstrap with a confidence level of $\delta' = \delta/(|\mathcal{A}||\mathcal{M}|)$ and $10{,}000$ bootstrap samples of the aggregate performance. Each bootstrap is formed by re-sampling the performance of each algorithm on each environment for the collected data. Then the aggregate performance for each bootstrap is computed. The lower and upper

confidence intervals are given by the $100(\delta'/2)$ and $100(1-\delta'/2)$ percentile from the bootstrap aggregate performance. Since $\delta'/2$ is often really small, $10{,}000$ bootstrap samples are needed to get confidence intervals that more accurately reflect the true confidence intervals. This requires a substantial amount of compute time and can take over four hours for an original sample size of $10{,}000$.

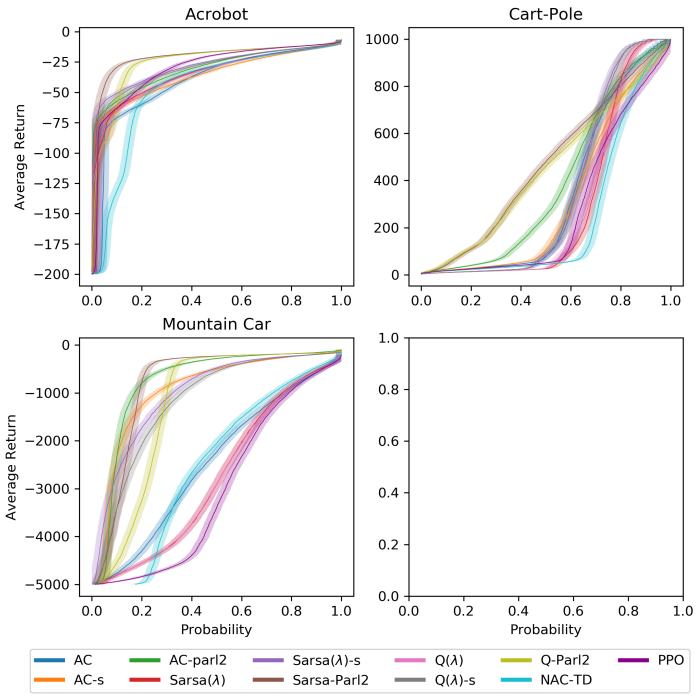
H. Confidence interval test experiment

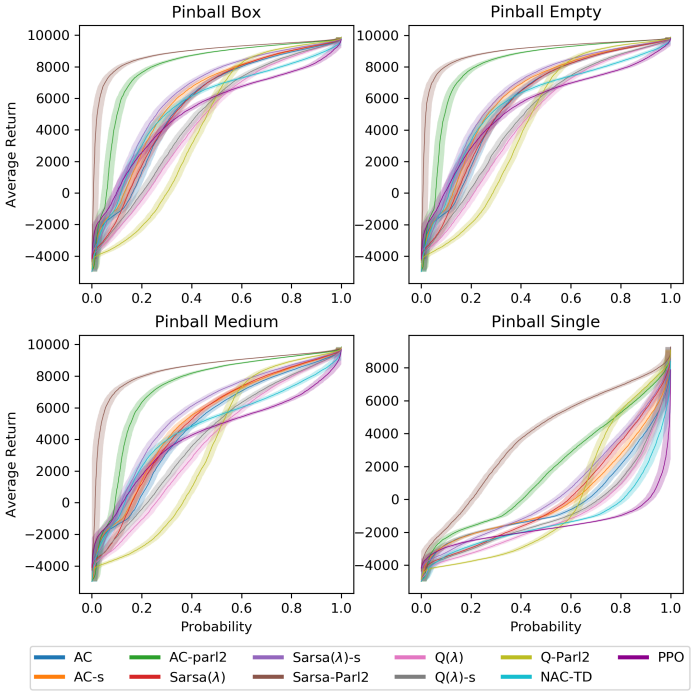
To test the different bounding techniques, we estimate the failure rate of each confidence interval technique at different sample sizes. For this experiment we execute $1{,}000$ trials of the evaluation procedure using samples sizes (trials per algorithm per environment) of 10, 30, 100, $1{,}000$, and $10{,}000$. There are a total of 11.14 million samples per algorithm per environment. To reduce computation costs, we limit this experiment to only include the Sarsa-Parl2, Q-Parl2, AC-Parl2, and Sarsa(λ)-s. Additionally, we reduce the environment set to be the discrete environments and mountain car. Then we compute the proportion of violations for any confidence interval. All methods use the same data, so the results are not independent. We choose not to run independent samples of the bounds to limit our environmental impact. The method to compute of the proportion of violations and number of significant pairwise comparison can be found in the source code.

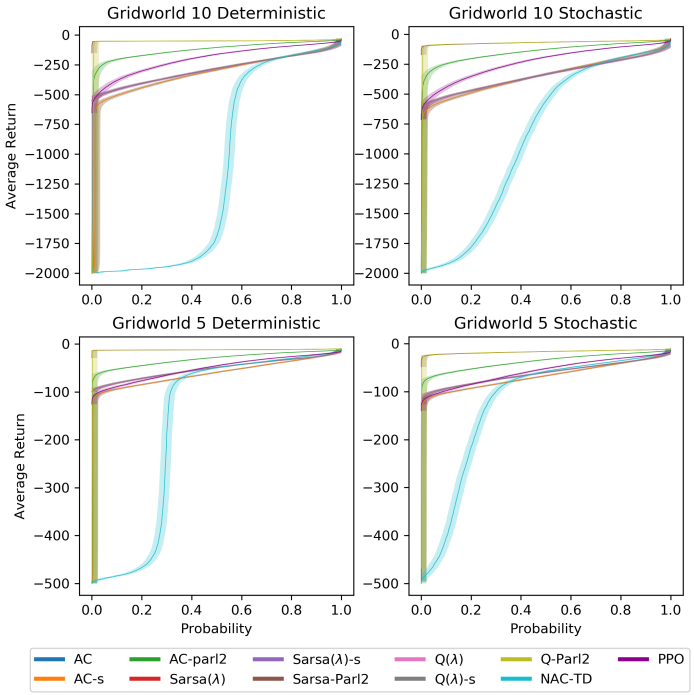
It it important to note that when this experiment was run, there was a bug in the code that made the step-size for the Parl2 methods smaller by a factor of 0.1. This does not invalidate the results, only that the algorithms run are not equivalent those used in the other experiments in this paper. The main impact of this difference was that Sarsa-Parl2 and Q-Parl2 did were not as effective on the discrete MDPs (though they diverged even less) and their scores were nearly the same. Since both of these algorithms had near identical scores on most of the environments, it became almost impossible to differentiate them, so detecting five out of six (83%) statistically significant comparisons is considered optimal for this experiment.

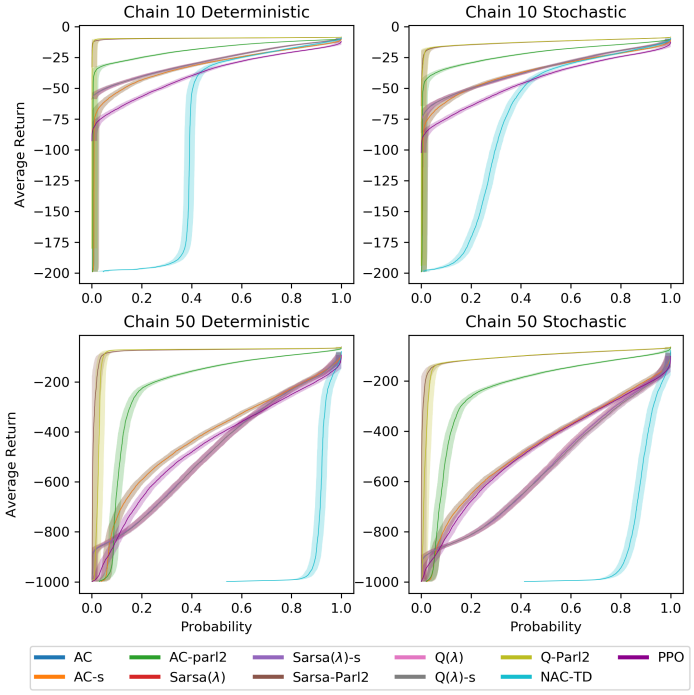
I. Performances

This section illustrates the distribution of performance of each algorithm. Tables of the average performance (rounded to the tenths place) along with the algorithm rank on that environment. In the Figures showing the performance distributions the shaded regions represent $100(1-0.05/|\mathcal{A}||\mathcal{M}|)\% \approx 99.9697\%$ confidence intervals computed via DKW. In the performance tables $100(1-0.05/|\mathcal{A}||\mathcal{M}|)\%$ confidence intervals are shown in parentheses.


Acrobot
Cart-Pole
Mountain Car
Average Return
Probability
AC
AC-s
AC-parl2
Sarsa(λ)
Sarsa(λ)-s
Sarsa-Parl2
Q(λ)
Q(λ)-s
Q-Parl2
NAC-TD
PPO


Pinball Box
Pinball Empty
Pinball Medium
Pinball Single
Average Return
Probability
AC
AC-s
AC-parl2
Sarsa(λ)
Sarsa(λ)-s
Sarsa-Parl2
Q(λ)
Q(λ)-s
Q-Parl2
NAC-TD
PPO


Gridworld 10 Deterministic
Gridworld 10 Stochastic
Gridworld 5 Deterministic
Gridworld 5 Stochastic
Average Return
Probability
AC
AC-s
AC-parl2
Sarsa(λ)
Sarsa(λ)-s
Sarsa-Parl2
Q(λ)
Q(λ)-s
Q-Parl2
NAC-TD
PPO


Chain 10 Deterministic
Chain 10 Stochastic
Chain 50 Deterministic
Chain 50 Stochastic
Average Return
Probability
AC
AC-s
AC-parl2
Sarsa(λ)
Sarsa(λ)-s
Sarsa-Parl2
Q(λ)
Q(λ)-s
Q-Parl2
NAC-TD
PPO

Acrobot		
Algorithm	Mean	Rank
Sarsa-Parl2	-20.6 (-24.6, -18.1)	1 (2, 1)
Q-Parl2	-25.7 (-29.8, -22.8)	2 (6, 1)
Sarsa(λ)-s	-28.3 (-32.3, -26.3)	3 (6, 2)
Q(λ)-s	-30.3 (-34.3, -27.9)	4 (9, 2)
PPO	-30.5 (-34.5, -26.7)	5 (9, 2)
AC-parl2	-30.6 (-34.6, -28.8)	6 (9, 2)
Sarsa(λ)	-34.7 (-38.7, -32.4)	7 (10, 4)
Q(λ)	-35.7 (-39.7, -33.3)	8 (10, 4)
AC-s	-36.0 (-40.0, -33.5)	9 (10, 4)
AC	-41.4 (-45.4, -37.3)	10 (11, 7)
NAC-TD	-47.1 (-51.1, -43.0)	11 (11, 10)

Cart-Pole		
Algorithm	Mean	Rank
Sarsa-Parl2	469.2 (448.3, 490.0)	1 (2, 1)
Q-Parl2	450.3 (429.6, 471.1)	2 (2, 1)
AC-parl2	390.4 (369.7, 411.2)	3 (3, 3)
Sarsa(λ)-s	347.5 (326.5, 368.4)	4 (7, 4)
Q(λ)-s	345.5 (324.5, 366.3)	5 (7, 4)
AC	338.7 (317.8, 359.6)	6 (7, 4)
AC-s	320.6 (300.0, 341.5)	7 (9, 4)
Q(λ)	291.0 (270.1, 311.9)	8 (10, 7)
Sarsa(λ)	287.2 (265.9, 308.6)	9 (10, 7)
PPO	276.3 (256.1, 297.1)	10 (11, 8)
NAC-TD	244.7 (224.0, 265.5)	11 (11, 10)

Mountain Car		
Algorithm	Mean	Rank
AC-parl2	-791.6 (-894.0, -688.7)	1 (3, 1)
Sarsa-Parl2	-843.2 (-945.0, -740.5)	2 (4, 1)
AC-s	-966.0 (-1068.0, -863.1)	3 (4, 1)
Sarsa(λ)-s	-1024.2 (-1125.8, -923.5)	4 (5, 2)
Q(λ)-s	-1197.3 (-1298.9, -1094.9)	5 (6, 4)
Q-Parl2	-1269.3 (-1371.2, -1166.4)	6 (6, 5)
AC	-2477.1 (-2576.0, -2374.3)	7 (8, 7)
NAC-TD	-2489.3 (-2589.1, -2386.4)	8 (8, 7)
Sarsa(λ)	-2769.1 (-2867.8, -2666.2)	9 (10, 9)
Q(λ)	-2779.3 (-2877.9, -2676.4)	10 (10, 9)
PPO	-3026.3 (-3124.9, -2923.5)	11 (11, 11)

Chain 10 Deterministic		
Algorithm	Mean	Rank
Q-Parl2	-9.2 (-13.2, -9.1)	1 (2, 1)
Sarsa-Parl2	-9.3 (-13.3, -9.2)	2 (2, 1)
AC-parl2	-19.0 (-23.0, -17.7)	3 (3, 3)
Sarsa(λ)	-27.9 (-31.8, -26.9)	4 (9, 4)
Sarsa(λ)-s	-27.9 (-31.8, -26.9)	4 (9, 4)
Q(λ)	-27.9 (-31.9, -26.9)	6 (9, 4)
Q(λ)-s	-27.9 (-31.9, -26.9)	6 (9, 4)
AC	-32.2 (-36.2, -29.9)	8 (9, 4)
AC-s	-32.2 (-36.2, -29.9)	8 (9, 4)
PPO	-38.0 (-41.9, -36.5)	10 (10, 10)
NAC-TD	-89.7 (-93.7, -85.7)	11 (11, 11)

Chain 10 Stochastic		
Algorithm	Mean	Rank
Q-Parl2	-12.7 (-16.7, -12.2)	1 (2, 1)
Sarsa-Parl2	-12.8 (-16.8, -12.4)	2 (2, 1)
AC-parl2	-22.6 (-26.6, -21.2)	3 (3, 3)
Q(λ)	-34.9 (-38.9, -33.6)	4 (9, 4)
Q(λ)-s	-34.9 (-38.9, -33.6)	4 (9, 4)
Sarsa(λ)	-35.1 (-39.0, -33.7)	6 (9, 4)
Sarsa(λ)-s	-35.1 (-39.0, -33.7)	6 (9, 4)
AC	-36.9 (-40.8, -34.7)	8 (9, 4)
AC-s	-36.9 (-40.8, -34.7)	8 (9, 4)
PPO	-43.8 (-47.6, -42.1)	10 (10, 10)
NAC-TD	-75.2 (-79.1, -71.2)	11 (11, 11)

Chain 50 Deterministic		
Algorithm	Mean	Rank
Sarsa-Parl2	-78.3 (-97.9, -71.5)	1 (2, 1)
Q-Parl2	-92.7 (-112.3, -74.6)	2 (2, 1)
AC-parl2	-231.0 (-250.5, -211.3)	3 (3, 3)
AC	-425.5 (-444.1, -405.8)	4 (6, 4)
AC-s	-425.5 (-444.1, -405.8)	4 (6, 4)
PPO	-462.2 (-480.4, -442.6)	6 (10, 4)
Sarsa(λ)	-479.5 (-498.3, -462.4)	7 (10, 6)
Sarsa(λ)-s	-479.5 (-498.3, -462.4)	7 (10, 6)
Q(λ)	-481.6 (-500.4, -464.0)	9 (10, 6)
Q(λ)-s	-481.6 (-500.4, -464.0)	9 (10, 6)
NAC-TD	-928.4 (-946.8, -908.7)	11 (11, 11)

Chain 50 Stochastic		
Algorithm	Mean	Rank
Sarsa-Parl2	-101.6 (-121.1, -96.4)	1 (2, 1)
Q-Parl2	-111.1 (-130.6, -97.5)	2 (2, 1)
AC-parl2	-239.3 (-258.7, -219.7)	3 (3, 3)
AC	-463.2 (-481.5, -443.6)	4 (6, 4)
AC-s	-463.2 (-481.5, -443.6)	4 (6, 4)
PPO	-474.4 (-492.2, -455.3)	6 (6, 4)
Sarsa(λ)	-546.9 (-565.2, -529.2)	7 (10, 7)
Sarsa(λ)-s	-546.9 (-565.2, -529.2)	7 (10, 7)
Q(λ)	-550.3 (-568.7, -532.5)	9 (10, 7)
Q(λ)-s	-550.3 (-568.7, -532.5)	9 (10, 7)
NAC-TD	-897.4 (-915.3, -877.8)	11 (11, 11)

Gridworld 10 Deterministic		
Algorithm	Mean	Rank
Sarsa-Parl2	-49.5 (-90.7, -48.9)	1 (2, 1)
Q-Parl2	-60.8 (-102.0, -48.5)	2 (2, 1)
AC-parl2	-130.6 (-171.6, -115.1)	3 (3, 3)
PPO	-199.6 (-240.4, -188.8)	4 (4, 4)
Q(λ)	-290.6 (-331.0, -278.9)	5 (10, 5)
Q(λ)-s	-290.6 (-331.0, -278.9)	5 (10, 5)
Sarsa(λ)	-291.8 (-332.3, -281.3)	7 (10, 5)
Sarsa(λ)-s	-291.8 (-332.3, -281.3)	7 (10, 5)
AC	-324.7 (-365.2, -295.3)	9 (10, 5)
AC-s	-324.7 (-365.2, -295.3)	9 (10, 5)
NAC-TD	-1141.3 (-1181.6, -1099.9)	11 (11, 11)

Gridworld 10 Stochastic		
Algorithm	Mean	Rank
Sarsa-Parl2	-67.5 (-108.6, -66.1)	1 (2, 1)
Q-Parl2	-75.4 (-116.4, -64.8)	2 (2, 1)
AC-parl2	-146.4 (-187.3, -135.8)	3 (3, 3)
PPO	-229.5 (-270.1, -217.6)	4 (4, 4)
Q(λ)	-339.2 (-379.2, -326.9)	5 (10, 5)
Q(λ)-s	-339.2 (-379.2, -326.9)	5 (10, 5)
Sarsa(λ)	-342.3 (-382.4, -330.0)	7 (10, 5)
Sarsa(λ)-s	-342.3 (-382.4, -330.0)	7 (10, 5)
AC	-347.2 (-387.4, -329.7)	9 (10, 5)
AC-s	-347.2 (-387.4, -329.7)	9 (10, 5)
NAC-TD	-864.1 (-904.2, -823.3)	11 (11, 11)

Gridworld 5 Deterministic		
Algorithm	Mean	Rank
Sarsa-Parl2	-12.2 (-22.5, -12.1)	1 (2, 1)
Q-Parl2	-12.5 (-22.7, -12.0)	2 (2, 1)
AC-parl2	-32.2 (-42.4, -30.1)	3 (3, 3)
PPO	-51.1 (-61.2, -49.0)	4 (10, 4)
Q(λ)	-51.4 (-61.5, -49.6)	5 (10, 4)
Q(λ)-s	-51.4 (-61.5, -49.6)	5 (10, 4)
Sarsa(λ)	-51.9 (-62.0, -50.1)	7 (10, 4)
Sarsa(λ)-s	-51.9 (-62.0, -50.1)	7 (10, 4)
AC	-62.0 (-72.1, -58.2)	9 (10, 4)
AC-s	-62.0 (-72.1, -58.2)	9 (10, 4)
NAC-TD	-168.1 (-178.3, -157.9)	11 (11, 11)

Gridworld 5 Stochastic		
Algorithm	Mean	Rank
Sarsa-Parl2	-17.0 (-27.2, -16.6)	1 (2, 1)
Q-Parl2	-17.1 (-27.4, -16.3)	2 (2, 1)
AC-parl2	-37.2 (-47.3, -35.4)	3 (3, 3)
PPO	-57.2 (-67.3, -54.9)	4 (10, 4)
Q(λ)	-61.5 (-71.6, -59.4)	5 (10, 4)
Q(λ)-s	-61.5 (-71.6, -59.4)	5 (10, 4)
Sarsa(λ)	-61.5 (-71.6, -59.4)	7 (10, 4)
Sarsa(λ)-s	-61.5 (-71.6, -59.4)	7 (10, 4)
AC	-67.6 (-77.6, -64.8)	9 (10, 4)
AC-s	-67.6 (-77.6, -64.8)	9 (10, 4)
NAC-TD	-125.4 (-135.5, -115.4)	11 (11, 11)

Pinball Box		
Algorithm	Mean	Rank
Sarsa-Parl2	8823.2 (8513.4, 8998.4)	1 (1, 1)
AC-parl2	7875.4 (7565.9, 8170.2)	2 (2, 2)
Sarsa(λ)-s	6288.3 (5980.0, 6569.4)	3 (4, 3)
AC-s	5961.2 (5653.1, 6251.1)	4 (7, 3)
Sarsa(λ)	5603.0 (5295.1, 5889.7)	5 (8, 4)
AC	5602.5 (5295.1, 5886.8)	6 (8, 4)
NAC-TD	5546.8 (5243.2, 5838.0)	7 (8, 4)
PPO	5184.6 (4882.6, 5447.4)	8 (9, 5)
Q(λ)-s	4728.6 (4421.8, 5020.1)	9 (11, 8)
Q(λ)	4449.9 (4143.3, 4740.8)	10 (11, 9)
Q-Parl2	4246.5 (3937.2, 4539.9)	11 (11, 9)

Pinball Empty		
Algorithm	Mean	Rank
Sarsa-Parl2	8942.1 (8631.8, 9115.3)	1 (1, 1)
AC-parl2	8041.0 (7730.8, 8333.4)	2 (2, 2)
Sarsa(λ)-s	6382.4 (6073.4, 6664.1)	3 (5, 3)
AC-s	6155.8 (5846.5, 6444.9)	4 (7, 3)
AC	5814.2 (5505.4, 6097.3)	5 (8, 3)
Sarsa(λ)	5778.6 (5469.8, 6063.7)	6 (8, 4)
NAC-TD	5710.1 (5405.2, 5998.0)	7 (8, 4)
PPO	5401.3 (5097.3, 5666.6)	8 (9, 5)
Q(λ)-s	4891.8 (4584.1, 5182.5)	9 (11, 8)
Q(λ)	4602.3 (4294.8, 4892.1)	10 (11, 9)
Q-Parl2	4487.9 (4178.3, 4781.5)	11 (11, 9)

Pinball Medium		
Algorithm	Mean	Rank
Sarsa-Parl2	8402.1 (8094.2, 8625.5)	1 (1, 1)
AC-parl2	7128.1 (6820.6, 7429.2)	2 (2, 2)
Sarsa(λ)-s	5667.4 (5361.4, 5949.4)	3 (4, 3)
AC-s	5195.5 (4890.3, 5487.2)	4 (7, 3)
Sarsa(λ)	5050.5 (4745.1, 5336.5)	5 (7, 4)
AC	4835.7 (4531.8, 5123.8)	6 (7, 4)
NAC-TD	4652.5 (4353.9, 4940.2)	7 (9, 4)
PPO	4227.1 (3932.0, 4493.3)	8 (10, 7)
Q(λ)-s	4084.7 (3780.6, 4375.1)	9 (10, 7)
Q(λ)	3689.6 (3386.2, 3981.9)	10 (11, 8)
Q-Parl2	3404.5 (3097.9, 3699.2)	11 (11, 10)

Pinball Single		
Algorithm	Mean	Rank
Sarsa-Parl2	3754.6 (3470.3, 4023.3)	1 (1, 1)
AC-parl2	1696.7 (1418.4, 1989.3)	2 (2, 2)
Sarsa(λ)-s	514.1 (240.8, 793.2)	3 (5, 3)
Sarsa(λ)	119.2 (-151.7, 408.1)	4 (7, 3)
AC-s	103.8 (-158.5, 389.6)	5 (7, 3)
Q-Parl2	-73.4 (-352.7, 214.7)	6 (7, 4)
AC	-197.1 (-451.4, 95.1)	7 (7, 4)
Q(λ)-s	-746.6 (-1005.9, -457.2)	8 (10, 8)
Q(λ)	-986.1 (-1244.4, -693.8)	9 (10, 8)
NAC-TD	-1229.4 (-1463.7, -934.7)	10 (11, 8)
PPO	-1602.2 (-1789.1, -1327.7)	11 (11, 10)