Planning in Entropy-Regularized MDPs and Games

Jean-Bastien Grill*[1,2], Omar D. Domingues*[2], Pierre Ménard[2], Rémi Munos[1,2], Michal Valko[1,2]

[1]DeepMind Paris, [2]SequeL team, Inria Lille

Overview

- Planning with a simulator: estimate $V(s)$ with accuracy ε.
- Sample complexity = # calls to the simulator to get accuracy ε.
- Non-regularized case: there are no known algorithms with polynomial guarantees in a general setting.
- Regularization $\to$ smooth Bellman operator.
- `SmoothCruiser`: exploits smoothness $\to$ sample complexity that is always polynomial.

Setting & Assumptions

- MDPs and two-player games: $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.
- $\mathcal{S}, \mathcal{A}$: state and action spaces.
- $P \triangleq \{P(\cdot|s,a)\}_{s,a \in \mathcal{S}\times\mathcal{A}}$ transition probabilities.
- $R : \mathcal{S}\times\mathcal{A} \to [0,1]$ reward function.
- $\gamma \in [0,1[$ discount factor.
- **Assumption 1**: $|\mathcal{S}|$ arbitrary, $|\mathcal{A}| = K < \infty$.
- **Assumption 2**: we have a generative model (oracle, simulator) that can sample rewards and transitions: $R, Z \leftarrow$ `oracle`(s,a).

Regularized Value Functions

- Given a function $F_s : \mathbb{R}^K \to \mathbb{R}$, we define the value function as

$$V(s) \triangleq F_s(Q_s),$$
$$Q_s(a) \triangleq \mathbb{E}_{z\sim P(\cdot|s,a)}[R(s,a) + \gamma V(z)].$$

- In MDPs: $F_s = \max$ gives Bellman equations.
- In games: $F_s = \max$ for player 1 and $F_s = \min$ for player 2.
- Let $\mathrm{LogSumExp}_\lambda(x) \triangleq \lambda \log \sum_{i=1}^K \exp(x_i/\lambda)$.
- **Entropy regularization** with parameter λ:

$$\max \text{ becomes } \mathrm{LogSumExp}_\lambda$$
$$\min \text{ becomes } -\mathrm{LogSumExp}_{-\lambda}.$$

- Property: with regularization, F_s becomes L-smooth

$$|F_s(Q) - F_s(Q') - (Q-Q')^\intercal \nabla F_s(Q')| \le L\|Q-Q'\|_2^2$$

with $L = 1/\lambda$, $\nabla F_s(Q) \succeq 0$ and $\|\nabla F_s(Q)\|_1 = 1$.

Motivation

- **Strong regularization** $\lambda \to \infty$ and $L = 0$, that is, F_s is linear for all s: $F_s(x) = w_s^\intercal x$, with $\|w_s\|_1 = 1$, $w_s \in \mathbb{R}^k$ and $w_s \succeq 0$.
  $\to$ Monte Carlo sampling, $\mathcal{O}(1/\varepsilon^2)$ calls to the oracle.
- **No regularization** $\lambda = 0$ and $L \to \infty$, that is, F_s cannot be well approximated by a linear function.
  $\to$ Sparse Sampling algorithm (Kearns et al., 1999), $\mathcal{O}\big((1/\varepsilon)^{\log(1/\varepsilon)}\big)$ calls to the oracle.
- For $0 < \lambda < \infty$: interpolate between the two extreme cases using linear approximations of F_s.

Theoretical Guarantees

Theorem 1. Let $n(\varepsilon, \delta')$ be the number of calls to the generative model before the algorithm terminates. For any state $s \in \mathcal{S}$ and $\varepsilon, \delta' > 0$,

$$n(\varepsilon,\delta') = \widetilde{\mathcal{O}}(1/\varepsilon^4).$$

Theorem 2. For any $s \in \mathcal{S}$, $\varepsilon > 0$ and $\delta > 0$, there exists a choice of δ' that depends on ε and δ such that the output $\widehat{V}(s)$ of `SmoothCruiser`$(s, \varepsilon, \delta')$ satisfies

$$\mathbb{P}\big[|\widehat{V}(s) - V(s)| > \varepsilon\big] \le \delta.$$

and such that $n(\varepsilon,\delta') = \mathcal{O}(1/\varepsilon^{4+c})$ for any $c > 0$.

Intuition

- Linear approximation of F_s around $\widehat{Q}_s = $ `estimateQ`$(s, \sqrt{\varepsilon/L})$:

$$F_s(Q_s) \approx F_s(\widehat{Q}_s) + (Q_s - \widehat{Q}_s)^\intercal \nabla F_s(\widehat{Q}_s) + \varepsilon$$

- One of the terms can be written as an expected value:

$$Q_s^\intercal \nabla F_s(\widehat{Q}_s) = \mathbb{E}\Big[Q_s(A)\Big|\widehat{Q}_s\Big], \quad \text{with } A \sim \nabla F_s(\widehat{Q}_s)$$

- Problem: Q_s is unknown.
- Solution: use $\widetilde{Q}(A) = R_{s,A} + \gamma$`sampleV`$\big(Z_{s,A}, \varepsilon/\sqrt{\gamma}\big)$.
- Finally, we have (in expectation):

$$F_s(Q_s) \approx F_s(\widehat{Q}_s) - \widehat{Q}_s^\intercal \nabla F_s(\widehat{Q}_s) + \widetilde{Q}(A) + \varepsilon.$$

SmoothCruiser

Algorithm 1 `SmoothCruiser`

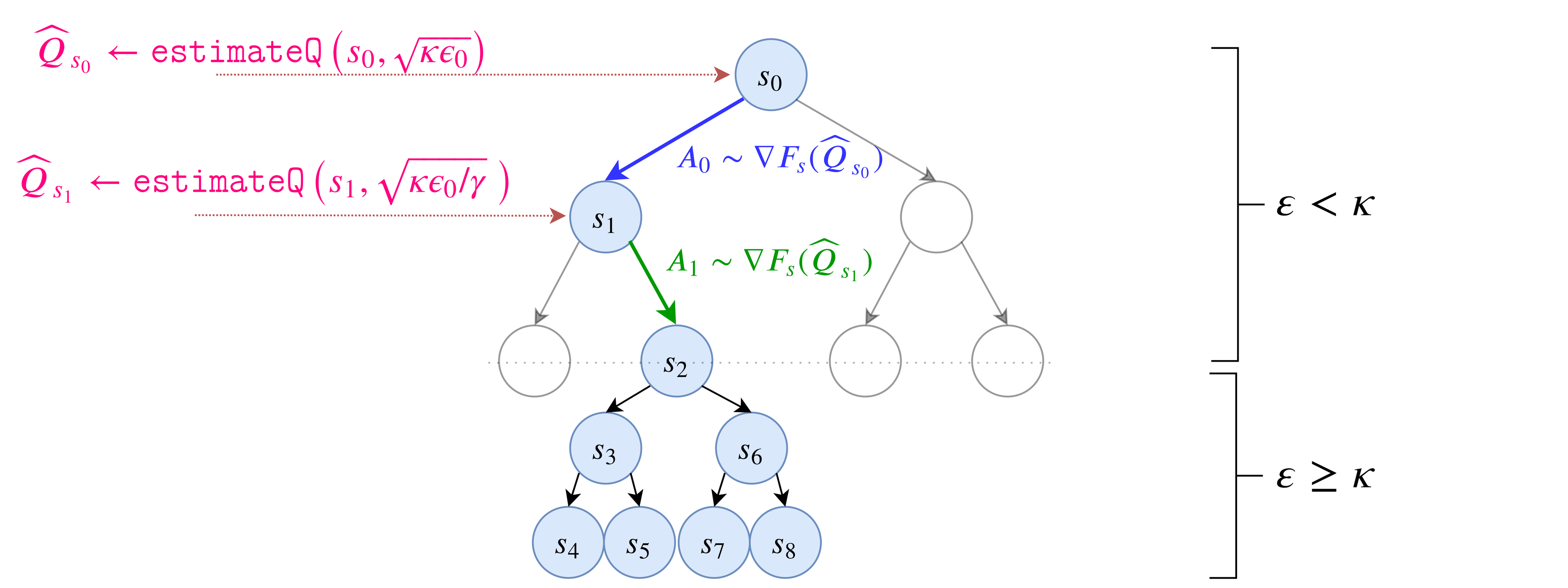
```
Input: (s, ε, δ') ∈ S × R+ × R+
M_λ ← sup_{s∈S} |F_s(0)| = λ log K
κ ← (1 − √γ)/(KL)
Set δ', κ and M_λ as a global parameters
Q̂_s ← estimateQ(s, ε)
Output: F_s(Q̂_s)
```

Algorithm 2 `estimateQ`

```
Input: (s, ε)
N(ε) ← O(log(2K/δ')/ε²)
for a ∈ A do
    Q̂_s(a) ← 0
    for i ∈ 1, ..., N(ε) do
        (R_i, Z_i) ← oracle(s, a).
        V̂_i ← sampleV(Z_i, ε/√γ)
    end for
    Q̂_s(a) = average of {R_i + γ V̂_i}_{i=1}^{N(ε)}
end for
Output: Q̂_s clipped to [0, Q_max]
```

Algorithm 3 `sampleV`

```
Input: (s, ε) ∈ S × R+
if ε ≥ (1 + M_λ)/(1 − γ) then
    Output: 0
else if ε ≥ κ then
    Q̂_s ← estimateQ(s, ε)
    Output: F_s(Q̂_s)
else if ε < κ then
    Q̂_s ← estimateQ(s, √(κε))
    A ← action drawn from ∇F_s(Q̂_s)
    (R, Z) ← oracle(s, A)
    V̂ ← sampleV(Z, ε/√γ)
    Output: F_s(Q̂_s) − Q̂_sᵀ∇F_s(Q̂_s) + (R + γV̂)
end if
```



Sample Complexity Simulation

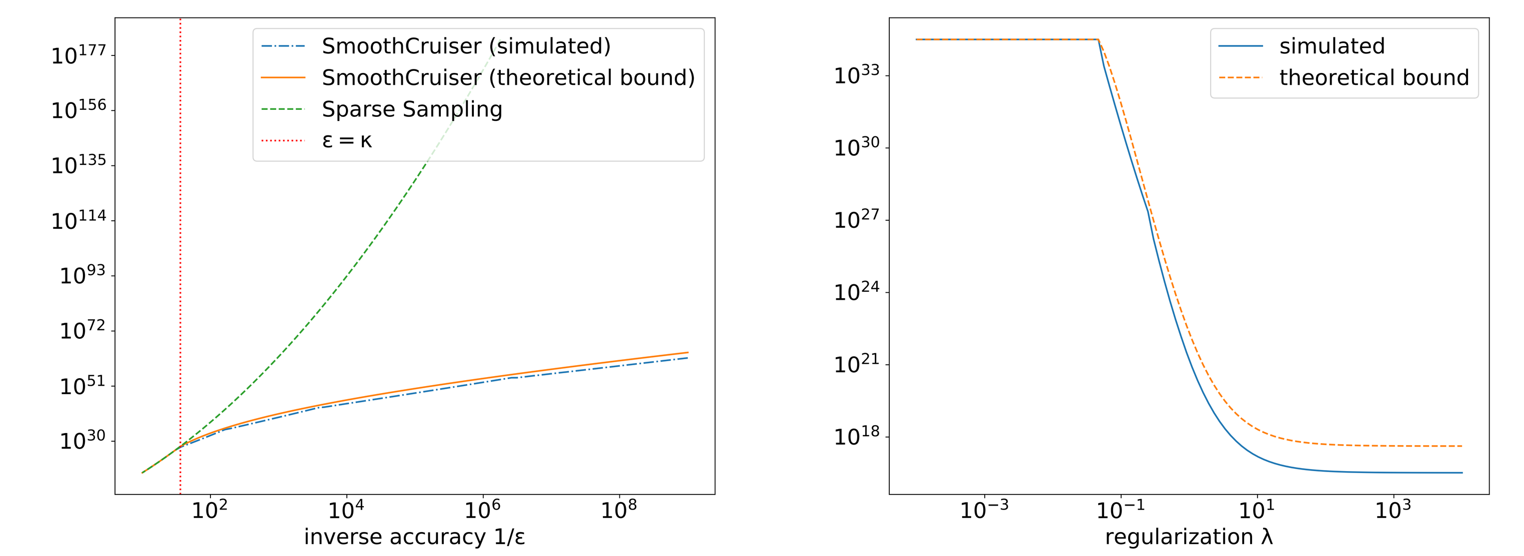


Figure 1: Left: number of calls to `sampleV` as a function of $1/\varepsilon$. Right: number of calls to `sampleV` required to achieve a 0.01 relative error as a function of the regularization λ.