Approximate Dynamic Programming and Performance Guarantees

Edwin K. P. Chong

Colorado State University

Chinese Control Conference
Keynote, 27 July 2021

Ack.: Ali Pezeshki, Yajing Liu, Zhenliang Zhang, Bowen Li.
Partially supported by NSF grant CCF-1422658 and CSU ISTeC.

AI and control

- Current AI boom.
- Many AI problems are *control* problems.
- Sequential decision making $\equiv$ control.
- Usual control framework: Stochastic optimal control.

Success stories

Examples of successful automated sequential decision making:

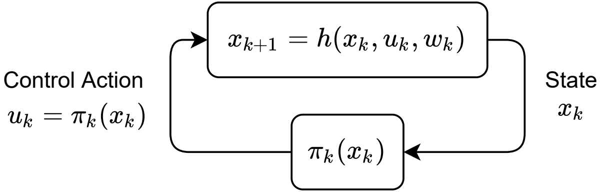
- 1997: IBM — Deep Blue vs. Garry Kasparov (chess).
- 2011: IBM — Watson in *Jeopardy!* (quiz show).
- 2017: DeepMind (Google) — AlphaGo (Wéiqí).
  Then AlphaZero (chess, etc.).
- 2019: Facebook and CMU — Pluribus (poker).
- 2021: Matt Ginsberg — Dr. Fill (crossword).

https://commons.wikimedia.org/w/index.php?curid=15223468

Motivation

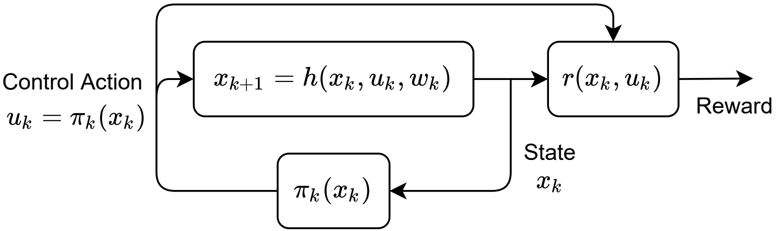
- Sequential decision making: Typically computationally intractable.
- Usual approach: Resort to approximations and heuristics.
- Downside: Often no performance guarantees.
- Current solution: Rely on empirical verification.
- This talk: Introduce method to bound performance.
    - From [Liu, Chong, Pezeshki, and Zhang ("LCPZ") (LCSS 2020)] and related past and ongoing research.
- Caveat: Cannot explain all mathematical details here. Will highlight only key points.

Stochastic optimal control: Closed-loop system



- $\mathcal{X}$ — set of *states*.
  - $x_k \in \mathcal{X}$ — state at "time" k (discrete).
- $\mathcal{U}$ — set of *control actions*.
  - $u_k \in \mathcal{U}$ — control action at time k.
- $h : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X}$ — state-transition function.
  - $x_{k+1} = h(x_k, u_k, w_k)$ with $\{w_k\}$ i.i.d. on $\mathcal{W}$; x_1 given.
- $\pi_k : \mathcal{X} \to \mathcal{U}$ — *policy* (state-feedback control law).
  - $u_k = \pi_k(x_k)$ (π_k can be random).

Stochastic optimal control: Reward



- $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ — reward function.
  - $r(x_k, u_k)$ — *reward* at state x_k with control action u_k (r can be random).

Stochastic optimal control: Optimization problem

- Objective function — *expected cumulative reward*.
  - Total reward over time horizon K (integer):

$$\sum_{k=1}^{K} \mathrm{E}[r(x_k, \pi_k(x_k))|x_1]$$

- Decision variable — policy $(\pi_1, \ldots, \pi_K)$.

$$\begin{aligned} \underset{(\pi_1,\ldots,\pi_K)}{\text{maximize}} \quad & \sum_{k=1}^{K} \mathrm{E}[r(x_k, \pi_k(x_k))|x_1] \\ \text{subject to} \quad & x_{k+1} = h(x_k, \pi_k(x_k), w_k), \;\; k = 1, \ldots, K-1 \\ & x_1 \text{ given.} \end{aligned}$$

Stochastic optimal control: Remarks

- State trajectory depends on policy.
- Also called *Markov decision problem (MDP)* (or *process*).
- Framework also for sequential decision making in AI.
  - AI planning $\approx$ optimal control;
    see, e.g., [Bertsekas and Tsitsiklis (1996)].
  - Brief history in [Chong, Kreucher, and Hero (DEDS 2009)].
- Can also incorporate *partial observations* (POMDP).
  - Output-feedback control.

Example of classical stochastic optimal control

- Our optimal-control problem statement is very general.
- Well-known classical example: Linear-Quadratic (LQ) control.

$$\begin{aligned} h(x_k, u_k, w_k) &= Ax_k + Bu_k + w_k \\ r(x_k, u_k) &= x_k^\top Q x_k + u_k^\top R u_k \end{aligned}$$

- Kalman et al., circa 1960. Now well covered in textbooks.
- But still a current research topic:
  - e.g., [Bioffi, Tu, and Slotine (2020)], [Gama and Sojoudi (2020)], [Zheng, Tang, and Li (2021)].
- For technical reasons, we focus on *finite* $\mathcal{X}$ and $\mathcal{U}$.
  - More common in modern applications and implementations.

Dynamic programming

- *Optimal* policy (notation: superscript $*$):

$$(\pi_1^*, \ldots, \pi_K^*) := \underset{(\pi_1,\ldots,\pi_K)}{\operatorname{argmax}} \sum_{k=1}^{K} \mathrm{E}[r(x_k, \pi_k(x_k))|x_1]$$

- *Expected value-to-go*:

$$V_{k+1}^*(x_k, u_k) := \sum_{i=k+1}^{K} \mathrm{E}[r(x_i^*, \pi_i^*(x_i^*))|x_k, u_k].$$

- *Dynamic-programming* equation [Bellman (1957)]:

$$\boxed{\pi_k^*(x_k) = \underset{u\in\mathcal{U}}{\operatorname{argmax}} \{r(x_k, u) + V_{k+1}^*(x_k, u)\},} \quad k = 1, \ldots, K.$$

Approximate dynamic programming (ADP)

- Can compute optimal policy from dynamic-programming equation.
  - Value iteration, policy iteration, linear programming, etc.
- But practically intractable.
  - *Curse of dimensionality* [Bellman (1957)].
- Approximate expected value-to-go V^*_{k+1} by $\hat{V}_{k+1}$.
- *ADP policy* (notation: hat):

$$\boxed{\hat{\pi}_k(x_k) = \operatorname*{argmax}_{u \in \mathcal{U}} \{r(x_k, u) + \hat{V}_{k+1}(x_k, u)\}.}$$

Same as dynamic-programming equation except V^*_{k+1} replaced by $\hat{V}_{k+1}$.

Examples of ADP schemes

- Myopic — $\hat{V}_{k+1} = 0$.
- Reinforcement learning — $\hat{V}_{k+1}$ by training neural net.
- Rollout — $\hat{V}_{k+1}$ from *base policy*.
  - Model-predictive control (MPC)
  - Open-loop feedback control (OLFC)
  - Parallel rollout (multiple base policies)
- Hindsight optimization — $\hat{V}_{k+1}$ by optimizing action sequence per sample path.
- See, e.g., Bertsekas' ADP book (2012). Also [Chong, Kreucher, and Hero (DEDS 2009)].

Overview of approach

Goal: Bound the performance of an ADP scheme.

Approach:

1. Prove *key bounding theorem* for *greedy* schemes.
   - Bound depends on **curvature** of objective function.
2. Apply key bounding theorem to derive bounding result for ADP.
3. Develop method to estimate curvature.
   - Use Monte Carlo sampling.
   - Must be computationally "easy."

What kind of bound?

- Recall goal: Bound the performance of an ADP scheme.
- Form of result: "Objective function value of ADP scheme relative to optimal is no worse than ..."
- Two kinds:
  - *Difference* between values of ADP and optimal policy.
  - *Ratio* of values of ADP and optimal policy.
    - Normalized difference bound $\equiv$ ratio bound.
- Difference bound: See Bertsekas' textbook (2017).
- Here: **Ratio bound**.

General string-optimization problem

- Temporarily put optimal control and ADP aside.
- Instead, consider general *string-optimization problem*.
- $\mathbb{A}$ — set of *symbols*.
- $A = a_1 a_2 \cdots a_k$ — *string* of symbols with length $|A| = k$.
- $\mathbb{A}_K$ — set of all possible strings of length up to K, including empty string $\varnothing$. (*Uniform matroid of rank* K.)
- $f : \mathbb{A}_K \to \mathbb{R}_+$ — objective function. WLOG, $f(\varnothing) = 0$.

$$\boxed{\begin{array}{rl} \text{maximize} & f(A) \\ \text{subject to} & A \in \mathbb{A}_K. \end{array}}$$

More terminology and notation

- Terminology and notation used in discrete event systems.
- Given $A = a_1 a_2 \cdots a_m$ and $B = b_1 b_2 \cdots b_n)$ in $\mathbb{A}_K$, define *concatenation*: $A \oplus B := a_1 \cdots a_m b_1 \ldots b_n$.
- A is a *prefix* of C if $C = A \oplus B$. Notation: $A \preceq C$.
- f is *prefix monotone* if $\forall\ A \preceq B \in \mathbb{A}_K$, $f(A) \leq f(B)$.
- f is *subadditive* if $\forall\ A \preceq B \in \mathbb{A}_K$ and $a \in \mathbb{A}$, $f(B \oplus (a)) - f(B) \leq f(A \oplus (a)) - f(A)$.
- Subadditivity also called *diminishing-return* property.

Optimal and greedy solutions

- Default assumption: f prefix monotone
  $\implies$ $\exists$ optimal solution with length K.
- **Optimal** solution: $O_K = (o_1, \ldots, o_K)$.
- **Greedy** solution: $G_K = (g_1, g_2, \ldots, g_K)$ is called *greedy* if $\forall\ k = 1, 2, \ldots, K$,

$$g_k = \operatorname*{argmax}_{a \in \mathbb{A}} f((g_1, g_2, \ldots, g_{k-1}, a)).$$

- Greedy scheme $\equiv$ At each time, select best symbol independently of other times.

Curvatures

- Recall goal: Introduce general theorem on bounding greedy schemes for string optimization.
- Ratio bound: $f(G_K)/f(O_K) \geq$ ...
- Bound depends on certain numbers called *curvatures*.
- Two types: forward curvature and total curvature.
- Notation: Given any $A = (a_1, a_2, \dots, a_k) \in \mathbb{A}_K$ and $i, j \in \{1, \dots, k\}$, denote $A_{i:j} := (a_i, \dots, a_j)$ if $i \leq j$ and $A_{i:j} = \varnothing$ if $i > j$ (MATLAB notation).

Forward curvature

- Define *forward curvature* of f as

$$\boxed{\sigma := \max_{0 \le i < j \le K} \left(1 - \frac{f(G_{1:i} \oplus (o_j)) - f(G_{1:i})}{f(G_{1:i} \oplus O_{i+1:j}) - f(G_{1:i} \oplus O_{i+1:j-1})} \right)}$$

where $G_{1:0} := \varnothing$ and $O_{i+1:i} := \varnothing$ for all $i \in \{0, \dots, K-1\}$.

- Expression akin to a normalized second-order difference.
  - To see this, complete the fraction.
  - $\sigma =$ bound on normalized second-order difference.
- f prefix monotone $\Rightarrow 0 \le \sigma \le 1$.
- f subadditive $\Rightarrow \sigma = 0$.

Total curvature

- Define *total curvature* of f as

$$\boxed{\eta := \max_{\substack{1 \le i \le K-1 \\ G_{i:1} \neq 0}} \frac{K}{K-i} \left(1 - \frac{f(G_{1:i} \oplus O_{i+1:K}) - \frac{K-i}{K} f(O_K)}{f(G_{1:i})} \right)}$$

- f prefix monotone $\Rightarrow \eta \le f(O_k)/f((g_1))$.
- f subadditive $\Rightarrow \eta \ge 0$.

Key bounding theorem

Theorem

Key bounding theorem. *Given $f : \mathbb{A}_K \to \mathbb{R}_+$ prefix monotone,*

$$\frac{f(G_K)}{f(O_K)} \geq \frac{1}{\eta}\left(1 - \left(1 - \eta\frac{1-\sigma}{K}\right)^K\right).$$

- Slightly stronger than in [LCPZ (LCSS 2020)].
- Inspired by bounds in *submodular* optimization theory (orig. [Nemhauser (1978)]), akin to convex optimization.
- Submodular $\equiv$ prefix monotone and subadditive.
  - See survey paper [LCPZ (DEDS 2020)] and its references.

Remarks on key bounding theorem

- Key bounding theorem **does not require submodularity**.
- Bound is tight.
- Both curvatures involve O_K. Best we can do is bound curvatures from above (discussed later).
- Bound is decreasing in σ and $\eta \leq K/(1-\sigma)$.
  $\therefore$ If replace σ and η by upper bounds, theorem still holds.
- As $\eta \searrow 0$, bound $\nearrow 1-\sigma$.
- As $K \to \infty$, bound $\searrow \left(1 - e^{-\eta(1-\sigma)}\right)/\eta$.
- If $\sigma = 0$ and $\eta = 1$, then limit $= (1 - e^{-1})$.
  - Familiar in submodular optimization theory; e.g., [Nemhauser (1978)].

Key idea

- Now back to optimal control and ADP.
- Recall optimal-control objective function:

$$\sum_{k=1}^{K} \mathrm{E}[r(x_k, \pi_k(x_k))|x_1]$$

Decision variable: $(\pi_1, \ldots, \pi_K)$.

- Key idea: Given an ADP scheme,
  - define associated string-optimization problem,
  - then apply key bounding theorem.
- String: $(\pi_1, \ldots, \pi_K)$.
- Here, symbol = policy.

String-optimization problem for optimal control

- Define (for $k = 1, \ldots, K-1$ and $\hat{V}_{K+1}(\cdot,\cdot) := 0$)

$$f((\pi_1, \ldots, \pi_k)) := \sum_{i=1}^{k} \mathrm{E}[r(x_i, \pi_k(x_i))|x_1] + \mathrm{E}[\hat{V}_{k+1}(x_k, \pi_k(x_k))|x_1]$$

$$= \mathrm{E}[r(x_k, \pi_k(x_k)) + \hat{V}_{k+1}(x_k, \pi_k(x_k))|x_1] + \sum_{i=1}^{k-1} \mathrm{E}[r(x_i, \pi_i(x_i))|x_1].$$

- When $k = K$, f becomes objective function for original optimal-control problem (expected cumulative reward).
- $\boxed{\text{Maximizing } f \text{ solves optimal-control problem.}}$

Greedy policy-selection scheme for optimal control

- Define *greedy policy-selection (GPS)* scheme: For $k = 1, \ldots, K$,

$$\pi_k^g := \operatorname*{argmax}_{\pi} \; \mathrm{E}[r(x_k^g, \pi(x_k^g)) + \hat{V}_{k+1}(x_k^g, \pi(x_k^g)) | x_1]$$

  where $x_{i+1}^g = h(x_i^g, \pi_i^g(x_i^g), w_i)$, $i = 1, \ldots, k-1$, and $x_1^g = x_1$ (given).
- $\boxed{\text{GPS scheme is greedy scheme for } f.}$
- Thus, key bounding theorem applies.

ADP scheme for optimal control

- Recall *ADP* scheme: For $k = 1, \ldots, K$,

$$\hat{\pi}_k(\hat{x}_k) := \operatorname*{argmax}_u \{r(\hat{x}_k, u) + \hat{V}_{k+1}(\hat{x}_k, u)\}$$

  where $\hat{x}_{i+1} = h(\hat{x}_i, \hat{\pi}_i(\hat{x}_i), w_i)$ for $i = 1, \ldots, k-1$, $\hat{x}_1 = x_1$ (given), and $\hat{V}_{K+1}(\cdot, \cdot) := 0$.
- Looks just like GPS except:
  - argmax is over control action $u \in \mathcal{U}$
  - No expectation (E)

ADP is also GPS

- ADP control action depends on state trajectory.
- But ADP scheme still defines a particular policy.

Theorem

Any ADP scheme is also a GPS scheme.

Proof: By induction on k.

- ADP scheme is also greedy scheme for f.
- $\boxed{\text{Key bounding theorem applies to ADP scheme.}}$

Bounding ADP

Combining the previous ideas, we get our **main result**:

Theorem

Let $(\pi_1^*, \dots, \pi_K^*)$ be an optimal policy. If f is prefix monotone, then any ADP policy $(\hat{\pi}_1, \dots, \hat{\pi}_K)$ satisfies

$$\frac{f((\hat{\pi}_1, \dots, \hat{\pi}_K))}{f((\pi_1^*, \dots, \pi_K^*))} \geq \frac{1}{\eta}\left(1 - \left(1 - \eta\frac{1-\sigma}{K}\right)^K\right)$$

where η and σ are curvatures of f.

But how to compute or estimate η and σ?

Upper bound for curvature

- Given f, estimate **upper bounds** for curvatures η and σ.
  - Recall: Cannot compute curvatures exactly because they involve O_K.
  - Key bounding theorem applies to upper bounds on curvatures.
- Focus on η (similar treatment applies to σ).
- By definition of η, immediate upper bound given by

$$\eta \leq \max_{\substack{A \in \mathbb{A}_K, |A|=K \\ 1 \leq i \leq K-1}} \frac{K}{K-i}\left(1 - \frac{f(G_{1:i} \oplus A_{i+1:K}) - \frac{K-i}{K} f(A)}{f(G_{1:i})}\right).$$

- Computing G is easy.
- But $\max$ over (A, i) probably hard because of $A \in \mathbb{A}_K$.

Approach

- Use Monte Carlo sampling to estimate upper bound $\hat{\eta}$.
- Want $\hat{\eta}$ correct with high probability.
- **Curvature-estimation algorithm**:
  Given $\varepsilon, \delta \in (0,1)$, output $\hat{\eta}$ with the following desired properties relative to true curvature η:

$$\begin{aligned} \mathrm{P}\{\eta \geq (1-\varepsilon)\hat{\eta}\} &= 1 \quad (\hat{\eta} \text{ not too large}) \\ \mathrm{P}\{\eta \leq \hat{\eta}\} &\geq 1-\delta \quad (\hat{\eta} \text{ not too small}). \end{aligned}$$

- Related work: Testing submodularity for *order-agnostic* problems [Parnas and Ron 2002], [Sheshadhri and Vondrak (2010)], [Blais and Bommireddi (2016)].

Curvature-estimation algorithm

1. Generate J samples $s_1, \dots, s_J$ where $s_j = (A(j), i(j))$, $A(j) \in \mathbb{A}_K$, $|A(j)| = K$, and $1 \le i(j) \le K - 1$.
2. For each sample s, define $H(s) :=$

$$\frac{K}{K - i(s)} \left(1 - \frac{f(G_{1:i(s)} \oplus A_{i(s)+1:K}(s)) - \frac{K - i(s)}{K} f(A(s))}{f(G_{1:i(s)})} \right).$$

3. Output

$$\hat{\eta} := \left(\frac{1}{1 - \varepsilon} \right) \max_{1 \le j \le J} H(s_j).$$

Properties

- Our algorithm automatically satisfies first property:

$$\mathrm{P}\{\eta \geq (1-\varepsilon)\hat{\eta}\} = 1.$$

- Does it satisfy second property:

$$\mathrm{P}\{\eta \leq \hat{\eta}\} \geq 1-\delta?$$

  Depends on ε, δ, sampling distribution, and number of samples J. Also depends on distribution of f if we view f as random.

- Fix ε, δ, sampling distribution, and distribution of f. Treat J as variable.

Sample complexity

- Exhaustive search: $J =$ total number of possible pairs (A, i).
  - $J = |\mathbb{A}|^K(K-1)$ (i.e., *scaling law* is exponential in K).
  - $|\mathbb{A}|$ might be exponential in some other problem parameter (e.g., number of states).
  - Exponential in problem size $\implies$ impractical.
- *Sample complexity* of algorithm: Number of samples J needed to satisfy second property $\mathrm{P}\{\eta \leq \hat{\eta}\} \geq 1-\delta$ (or $\mathrm{P}\{\hat{\eta} < \eta\} \leq \delta$; i.e., $\delta =$ constraint on prob. of error).
- Sample complexity must be small relative to exhaustive search (e.g., $J =$ polynomial in problem size).
- Turns out not too difficult.

Probability of error

- Need J sufficiently large for $\mathrm{P}\{\hat{\eta} < \eta\} \leq \delta$.
- Recall:

$$(1-\varepsilon)\hat{\eta} = \max_{1 \leq j \leq J} H(s_j).$$

- Therefore,

$$\begin{aligned}\mathrm{P}\{\hat{\eta} < \eta\} &= \mathrm{P}\left\{\max_{j=1,\ldots,J} H(s_j) < (1-\varepsilon)\eta\right\} \\ &= \mathrm{P}\{\forall j = 1, \ldots, J,\ H(s_j) < (1-\varepsilon)\eta\}\end{aligned}$$

i.e., probability that *all* J samples erroneous.

  - Will decrease as J increases.

Example: i.i.d. sampling

- Suppose sampling is i.i.d.
- Using previous equation with $p(\varepsilon) := \mathrm{P}\{H(s_j) \geq (1-\varepsilon)\eta\}$ (probablity of correct sample),

$$\begin{aligned} \mathrm{P}\{\hat{\eta} < \eta\} &= \mathrm{P}\{\forall j = 1, \ldots, J,\ H(s_j) < (1-\varepsilon)\eta\} \\ &= \prod_{j=1}^{J} \mathrm{P}\{H(s_j) < (1-\varepsilon)\eta\} \\ &= (1 - p(\varepsilon))^J. \end{aligned}$$

- Taking natural $\log$, sample complexity given by

$$J \geq \frac{\log(1/\delta)}{-\log(1-p(\varepsilon))}.$$

Example: i.i.d. sampling (cont.)

- Simplify using inequality

$$\frac{1}{-\log(1-p(\varepsilon))} \leq \frac{1}{p(\varepsilon)}.$$

- We get the following simple *sufficient* condition on J:

$$\boxed{J \geq \frac{\log(1/\delta)}{p(\varepsilon)}.}$$

- Sample complexity increases with decreasing δ and $p(\varepsilon)$.
    - As expected.

Example: uniform sampling

- Suppose sampling is *uniform* i.i.d.
- Then $p(\varepsilon) =$ fraction of possible samples s such that $H(s) \geq (1-\varepsilon)\eta$; i.e., all possible samples for which $H(s)$ is within a factor of $(1-\varepsilon)$ of its maximum possible value.
- Recal: Usually express sample complexity in terms of scaling law as problem size grows.
- Reasonable assumption: As problem size grows, $p(\varepsilon) = \Omega(1)$ (i.e., bounded away from 0).
- This implies that sample complexity is $O(1)$ (i.e., bounded).
- Even if $p(\varepsilon)$ decreases polynomially, sample complexity grows only polynomially.

Summary

Alas, time's up!

- Introduced method to bound performance of ADP schemes.
- Showed derivation and key results.
- Described algorithm to estimate curvature and analyzed sample complexity.
- No time to show practical examples. (Future talk ...)

Questions?

edwin.chong@colostate.edu
www.edwinchong.us