Scalable Reinforcement Learning for Multi-Agent Networked Systems

Guannan Qu\textsuperscript{1,2} and Na Li\textsuperscript{1}

Harvard University
California Institute of Technology

CDC 2019
Control of Large Scale Networks

- Large scale
- Model assumption not correct
- Parameters not correct
- Large data available

→ Tractable/Efficient methods?

→ Reinforcement Learning
Markov Decision Process (MDP) over Networks

\[ a_i, s_i: \text{finite space} \]

State \( s = (s_1, \ldots, s_n) \in S_1 \times S_2 \times \cdots \times S_n := S \)

Action \( a = (a_1, \ldots, a_n) \in A_1 \times A_2 \times \cdots \times A_n := A \)
Markov Decision Process over Networks

**State Transition**

\[ P(s(t+1) | s(t), a(t)) = \prod_{i=1}^{n} P_i(s_{i}(t+1) | s_{N_i}(t), a_{i}(t)). \]

**Stage Reward**

\[ r(s) = \sum_{i=1}^{n} r_i(s_i, a_i) \]

**Objective**

\[
\max_{T \to \infty} \lim_{T \to \infty} \mathbb{E} \left[ \frac{1}{T} \sum_{t=0}^{T} r(s(t), a(t)) | s(0) = s \right]
\]
Challenges

Challenge 1  State/Action Space Exponentially Large!

\[ |S| = \prod_{i=1}^{n} |S_i| \quad |A| = \prod_{i=1}^{n} |A_i| \]

Methods for centralized MDP/RL

- Value/Policy Iteration, Q-learning [Watkins 1989]
- Actor-Critic [Konda 2000]
- Linear Programming [Bertsekas 1976]

Time/space complexity exponential in \( n \)

Methods in Multi-Agent MDP/RL to deal with scalability

- Independent Learners [Claus and Boutilier, 1998]
- Linear function approximation [Zhang 2018]
- Neuro Networks [Lowe et al., 2017]

Unclear how to choose approximator that guarantee (near)-optimality

Computation Complexity Results

Blondel and Tsitsiklis, 2000; Whittle, 1988; Papadimitriou and Tsitsiklis, 1999
Challenge 2: Information constraint

Local policy (deterministic): $a_i(t) = \zeta_i(s_i(t))$ where $\zeta_i : S_i \rightarrow A_i$
Is it possible to exploit network structure to design scalable algorithms that find a (near)-optimal localized policy?

- assume model is known (this CDC paper)
- model-free reinforcement learning (latest work)

Network structure (Local dependence) → Efficient methods!

Key: exponential decay property
Decomposition of Average Reward

Average Reward: \[ R(\zeta) = \mathbb{E}_{s \sim \pi(\zeta)} r(s, a) = \sum_{i=1}^{n} \mathbb{E}_{(s_i, a_i) \sim \pi_i(\zeta)} r_i(s_i, a_i) = \sum_{i=1}^{n} R_i(\zeta_1, \ldots, \zeta_n) \]

- Localized Policy: \[ a_i(t) = \zeta_i(s_i(t)) \]
- Joint Policies: \[ \zeta = (\zeta_1, \ldots, \zeta_n) \]
- Marginalized stationary distribution at node \( i \): \( \pi_i(\zeta) \)
- Expected reward at node \( i \) in stationarity: \( R_i(\zeta) \)

Depends on policies of all nodes
Outline of our approach:

- Approximate $R_i$ using a much simpler function
- Efficient alg. to optimize policies based on the approximate
- Analyze error between the approximate and true reward
Approximation of $R_i$

Local Transition Structure

$$P(s(t+1)|s(t), a(t)) = \prod_{i=1}^{n} P_i(s_i(t+1)|s_{N_i}(t), a_i(t)).$$

$\pi_i(\zeta)$ Marginalized stationary dist. at node $i$

$R_i(\zeta)$ Expected reward at node $i$ in stationarity

$$R_i(\zeta) = \mathbb{E}_{(s_i, a_i) \sim \pi_i(\zeta)} r_i(s_i, a_i)$$

$a_i(t) = \zeta_i(s_i(t))$ Localized Policies

$\zeta = (\zeta_1, \ldots, \zeta_n)$ Joint Policies
Original MDP

Local transition probabilities

... \( P_{i-2} \) \( P_{i-1} \) \( P_i \) \( P_{i+1} \) \( P_{i+2} \) ...

\[ \pi_i(\zeta) \] Marginalized stationary dist. at node i

\[ R_i(\zeta) \] Expected reward at node i in stationarity

\[ \hat{\pi}_i(\zeta_{N_i}^k) \] Marginalized stationary dist. for truncated model

\[ \hat{R}_i(\zeta_{N_i}^k) \] Approximate expected reward at i

Truncated MDP at node i:

Independent of nodes outside

\[ \hat{P}_{i-2} \] \( P_{i-1} \) \( P_i \) \( P_{i+1} \) \( \hat{P}_{i+2} \)

k-hop neighborhood (with k=1)
Original MDP

\[ \ldots P_{i-2} P_{i-1} P_i P_{i+1} P_{i+2} \ldots \]

\[ \pi_i(\zeta) \text{ Marginalized stationary dist. at node } i \]

\[ R_i(\zeta) \text{ Expected reward at node } i \text{ in stationarity} \]

\[ \hat{\pi}_i(\zeta_{N_i^k}) \text{ Marginalized stationary dist. for truncated model} \]

\[ \hat{R}_i(\zeta_{N_i^k}) \text{ Approximate expected reward at } i \]

Truncated MDP at node i:

\[ \hat{P}_{i-3} P_{i-2} P_{i-1} P_i P_{i+1} P_{i+2} \hat{P}_{i+3} \]

Independent of nodes outside

k-hop neighborhood (with k=2)
Outline of our approach:

- Approximate $R_i$ using a much simpler function
- Efficient alg. to optimize policies based on the approximate
- Analyze error between the approximate and true reward

\[
\max \quad R(\zeta) = \sum_{i=1}^{n} R_i(\zeta_1, \ldots, \zeta_n)
\]

\[
\max \quad \hat{R}(\zeta) = \sum_{i=1}^{n} \hat{R}_i(\zeta_{N_i^k})
\]

Much smaller combinations
Outline of our approach:

- Approximate \( R_i \) using a much simpler function
- Efficient alg. to optimize policies based on the approximate
- Analyze error between the approximate and true reward

\[
\max \ R(\zeta) = \sum_{i=1}^{n} R_i(\zeta_1, \ldots, \zeta_n)
\]

Much smaller combinations

\[
\max \ \hat{R}(\zeta) = \sum_{i=1}^{n} \hat{R}_i(\zeta_{N_i})
\]
Efficient Alg for $\max_{\zeta_i} \sum \hat{R}_i(\zeta_{N_i}^k) : \text{Dynamic Programming}$

The case of a line, and $k=1$ hop neighbor truncation,

$$
\max_{\zeta} \hat{R}(\zeta) = \max_{\zeta_1, \ldots, \zeta_n} \hat{R}_1(\zeta_1, \zeta_2) + \ldots + \hat{R}_i(\zeta_{i-1}, \zeta_i, \zeta_{i+1}) \ldots + \hat{R}_n(\zeta_{n-1}, \zeta_n)
$$

$$
= \max_{\zeta_1, \ldots, \zeta_i} \hat{R}_1(\zeta_1, \zeta_2) + \ldots + \hat{R}_{i-1}(\zeta_{i-2}, \zeta_{i-1}, \zeta_i) + \max_{\zeta_{i+1}, \ldots, \zeta_n} \hat{R}_i(\zeta_{i-1}, \zeta_i, \zeta_{i+1}) \ldots + \hat{R}_n(\zeta_{n-1}, \zeta_n)
$$

Properties:

$$V_i(\zeta_{i-1}, \zeta_i) = \max_{\zeta_{i+1}} \hat{R}_i(\zeta_{i-1}, \zeta_i, \zeta_{i+1}) + V_{i+1}(\zeta_i, \zeta_{i+1})$$

$$\hat{R}(\zeta) = \hat{R}_1(\zeta_1, \zeta_2) + V_2(\zeta_1, \zeta_2)$$
Efficient Alg for $\max_{\zeta_i} \sum \hat{R}_i(\zeta_{N_i}^k)$: Dynamic Programming

The case of a line, and $k=1$ hop neighbor truncation,

**Dynamic Programming**

\[
\begin{align*}
\text{Backward Sweep} & : & V_i(\zeta_{i-1}, \zeta_i) &= \max_{\zeta_{i+1}} \hat{R}_i(\zeta_{i-1}, \zeta_i, \zeta_{i+1}) + V_{i+1}(\zeta_i, \zeta_{i+1}) \\
\text{Forward Sweep} & : & (\zeta_1^*, \zeta_2^*) &= \arg \max_{\zeta_1, \zeta_2} \hat{R}_1(\zeta_1, \zeta_2) + V_2(\zeta_1, \zeta_2) \\
& & \zeta_{i+1}^* &= \arg \max_{\zeta_i} \hat{R}_i(\zeta_{i-1}^*, \zeta_i^*, \zeta_{i+1}) + V_{i+1}(\zeta_i^*, \zeta_{i+1})
\end{align*}
\]

**Proposition (Informal)** When the graph is a tree, this method finds a maximizer $\zeta^* = (\zeta_1^*, \ldots, \zeta_n^*)$ of $\hat{R}(\zeta_1, \ldots, \zeta_n)$ within time scaling in the policy space size of the largest $k$-hop neighborhood.

Comparison: much more efficient than directly optimize $R$!
Outline of our approach:

• Approximate $R_i$ using a much simpler function:  
  \[ \max \hat{R}(\zeta) = \sum_{i=1}^{n} \hat{R}_i(\zeta_{N_i}) \]

• Efficient alg. to optimize the approximate reward: Dynamic Programming

• Error between the approximate and true reward

\[ a_i(t) = \zeta_i(s_i(t)) \]
Outline of our approach:

- Approximate $R_i$ using a much simpler function:  
  $$\max \hat{R}(\zeta) = \sum_{i=1}^{n} \hat{R}_i(\zeta_1, \ldots, \zeta_n)$$

- Efficient alg. to optimize the approximate reward: Dynamic Programming

- Error between the approximate and true reward
Error between true and approximate reward

Error decays exponentially in $k$

Doing truncation might not have a big error, even for small $k$
Exponential Decaying Property: Formal Definition

\[ \pi_i(\zeta) \] Marginalized Stationary Distribution at node i of the full model

\[ \hat{\pi}_i(\zeta_{N_i}) \] Marginalized Stationary Distribution at node i of the truncated model

\((c, \rho)\)-exponential decay holds if for all \( i \), all policy \( \zeta = (\zeta_1, \ldots, \zeta_n) \)

\[ TV(\pi_i(\zeta), \hat{\pi}_i(\zeta_{N_i})) \leq c\rho^{k+1} \]

for some constant \( c > 0, \rho \in (0, 1) \)
Define interaction strength matrix $C = [C_{ij}]$

$$C_{ij} = \begin{cases} 
0 & \text{if } j \notin N_i \\
\sup_{s_{Ni}/j, a_i} \sup_{s_j, s'_j} TV(P_i(\cdot | s_j, s_{Ni}/j, a_i), P_i(\cdot | s'_j, s_{Ni}/j, a_i)) & \text{if } j \in N_i, j \neq i \\
\sup_{s_{Ni}/j} \sup_{s_i, s'_i, a_i, a'_i} TV(P_i(\cdot | s_i, s_{Ni}/i, a_i), P_i(\cdot | s'_i, s_{Ni}/i, a'_i)) & \text{if } j = i 
\end{cases}$$

**Lemma** If $\|C\|_1 \leq \rho < 1$, then the $(\frac{1}{1-\rho}, \rho)$-exponential decaying property holds, i.e.

$$TV(\pi_i(\zeta), \hat{\pi}_i(\zeta_{N_i}^k)) \leq \frac{1}{1-\rho} \rho^{k+1}$$

for all $i$, all policy $\zeta = (\zeta_1, \ldots, \zeta_n)$.

**Note:** For MDP with discounting rewards ($R = \sum_t \gamma^t r(t)$), exponential decaying property naturally holds under the ergodic condition (Qu, Weirman, Li, 2019).
Summary:

• Approximate $R_i$ using a much simpler function: $\max \hat{R}(\zeta) = \sum_{i=1}^{n} \hat{R}_i(\zeta_1, \ldots, \zeta_n)$

• Efficient alg. to optimize the approximate reward: Dynamic Programming

• Error between the approximate and true reward: Exponential decaying $\rho^k$!
What if we do not know the model parameters?
Review: Policy Gradient in the full information case (one agent)

**Parameterized Policy:** \( a(t) = \zeta^\theta(s(t)) \)

**Q Function:**
\[
Q^\theta(s, a) = \lim_{T \to \infty} \mathbb{E}^\theta\left[ \frac{1}{T} \sum_{t=0}^{T} r(s(t), a(t)) \middle| s(0) = s, a(0) = a \right]
\]
Limiting Average reward

\[
Q^\theta(s, a) = \mathbb{E}^\theta\left[ \sum_{t=0}^{\infty} \gamma^t r(s(t), a(t)) \middle| s(0) = s, a(0) = a \right]
\]
Discounting reward

**Policy Gradient Theorem** [Sutton 2000]

\[
\nabla R(\theta) = \mathbb{E}_{s \sim \pi^\theta, a \sim \zeta^\theta(\cdot | s)} Q^\theta(s, a) \nabla \log \zeta^\theta(a | s).
\]

**Actor-Critic Methods** [Konda 2000]

**Actor:** \( \theta \leftarrow \theta + \eta_t \times \nabla R(\theta) \)

**Critic:** \( Q(s(t-1), a(t-1)) \leftarrow Q(s(t-1), a(t-1)) + \alpha_t \times (\text{TD Error}) \)
Full Q function: $Q^0(s, a) = \sum_i^n Q_i(s, a)$

$Q_i(s_1, \ldots, s_n, a_1, \ldots, a_n)$

“Truncated” Q function.

$\hat{Q}_i(s_{N^k_i}, a_{N^k_i})$

Only depends on $k$-hop neighborhood

Exponentially large table
**Scalable Actor-Critic (SAC) for Networked MDP Learning**

Full Q function

\[ Q_i(s_1, \ldots, s_n, a_1, \ldots, a_n) \]

Full Policy Gradient

\[ \nabla_{\theta_i} J(\theta) = \mathbb{E}_{s \sim \pi^\theta, a \sim \zeta^\theta(a|s)} \nabla_{\theta_i} \log \zeta_i^{\theta_i}(a_i|s_i) Q^\theta(s, a) \]

Truncated Q Function

\[ \hat{Q}_i(s_{N_i^k}, a_{N_i^k}) \]

Truncated Policy Gradient

\[ h_i(\theta) = \mathbb{E}_{s \sim \pi^\theta, a \sim \zeta^\theta(a|s)} \nabla_{\theta_i} \log \zeta_i^{\theta_i}(a_i|s_i) \sum_{j \in N_i^k} \hat{Q}_j^\theta(s_{N_j^k}, a_{N_j^k}) \]

**Lemma (informal)** if the exponential decay property holds, then

\[ \| h_i(\theta) - \nabla_{\theta_i} J(\theta) \| \leq O(\rho^k) \quad \text{very small error even for small } k \]

Truncated Q + Truncated Policy Gradient \rightarrow Scalable RL for Networked Systems

Exploit network structure to design scalable algorithms that find a (near)-optimal localized policy

Key: exponential decay property

- Assume model is known (This CDC paper)
  - Truncation
  - Exponential Decay Property
- Model-free reinforcement learning (latest work)
  - Truncated Q function
  - Scalable Actor-Critic (SAC)

Thank you!