RL for Operations
Day 2: Nonparametric RL, Exogenous MDPs

Sean Sinclair, Sid Banerjee, Christina Yu
Cornell University

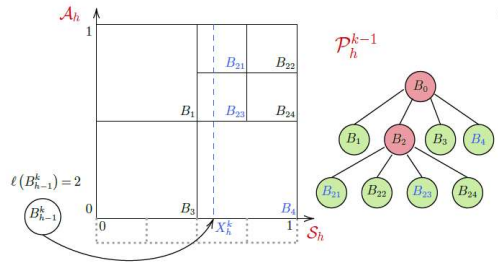


Figure 0 Illustrating the state-action partitioning scheme

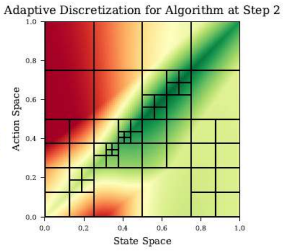
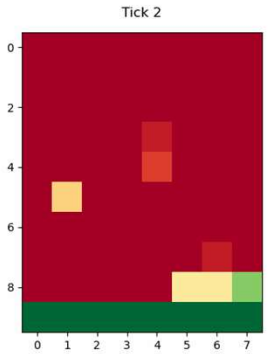
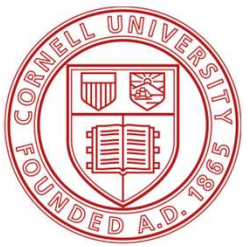


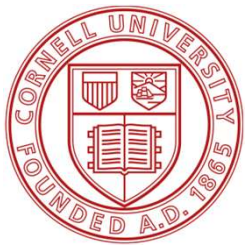
Figure 0 Partitioning in practice





RL for Operations
Day 2: Nonparametric RL, Exogenous MDPs

Sean Sinclair, Sid Banerjee, Christina Yu
Cornell University



Recap of Yesterday

MDP Basics

- Basic framework for Markov Decision Processes
- Tabular RL Algorithms with policy iteration + value iteration
- DeepRL algorithms (and their "tabular" counterparts)

Simulation Packages

- OpenAI Framework for simulation design
- Existing packages and code-bases for RL algorithm development

Simulation Implementation

- Developed simulator for problem using OpenAI Gym API

Tabular RL Algorithms

- Implement basic tabular RL algorithms to understand key algorithmic design aspects of *value estimates + value iteration*, *policy iteration*

Plan for Today

Nonparametric RL

- "Nonparametric" function approximation
- Strong guarantees across:
  Sample complexity, *space complexity*, *storage complexity*

Hindsight Learning

- Exogenous MDPs as model for OR problems
- Use of *Hindsight Planning* oracle for algorithm design
- Empirical results in VM allocation with Microsoft Azure

Tree-Partitions

- Implement tree-based adaptive discretization from nonparametric RL algorithms
- Use ORSuite to test on "continuous Ambulance routing"

Hindsight Planning for Exo-MDPs

- Use ORSuite model for revenue management and pricing (an example of an Exo-MDP)
- Implement Bayes Selector
- Use ORSuite to run simulations to compare performance against tabular algorithms

Plan for Today

Nonparametric RL

- "Nonparametric" function approximation
- Strong guarantees across:
  Sample complexity, *space complexity*, *storage complexity*

Hindsight Learning

- Exogenous MDPs as model for OR problems
- Use of *Hindsight Planning* oracle for algorithm design
- Empirical results in VM allocation with Microsoft Azure

Tree-Partitions

- Implement tree-based adaptive discretization from nonparametric RL algorithms
- Use ORSuite to test on "continuous Ambulance routing"

Hindsight Planning for Exo-MDPs

- Use ORSuite model for revenue management and pricing (an example of an Exo-MDP)
- Implement Bayes Selector
- Use ORSuite to run simulations to compare performance against tabular algorithms

Adaptive Discretization for Online Reinforcement Learning

Sean Sinclair,
Cornell University

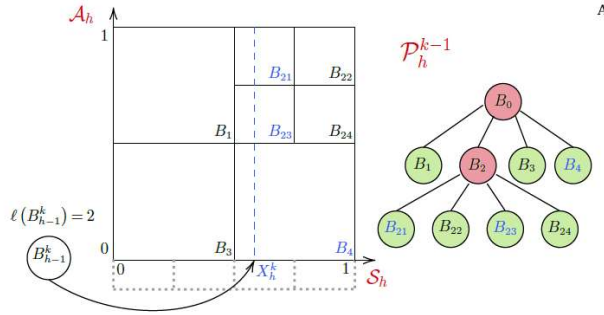


Figure 0 Illustrating the state-action partitioning scheme

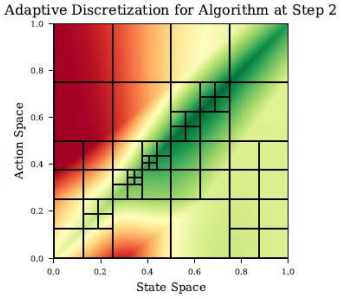
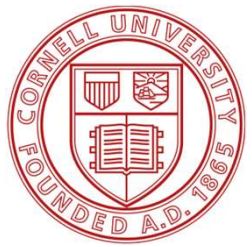


Figure 0 Partitioning in practice



Tabular Algorithms

Previously saw algorithms designed with value and policy iteration for tabular (discrete) MDPs.

However, even if problem is tabular:

```
MemoryError: Unable to allocate 31.9 GiB for an array with shape (3094, 720, 1280, 3) and data type float32
```

H = 3
S = 3094*720
A = 1280

Generalizing to OR

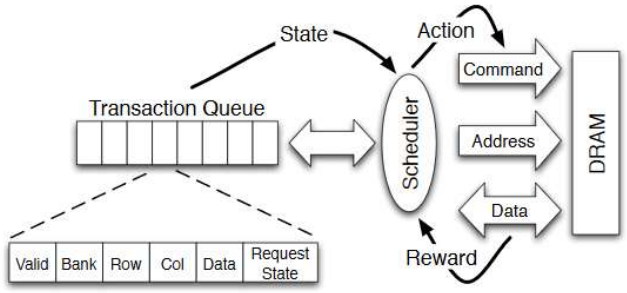
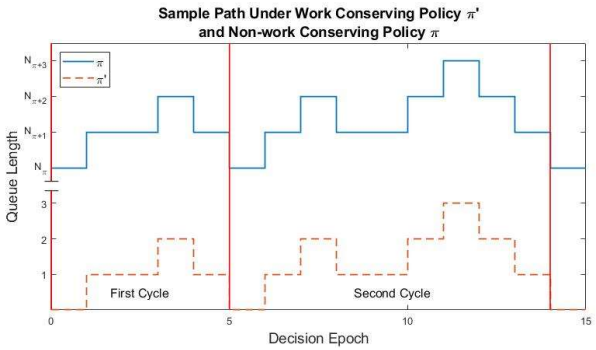
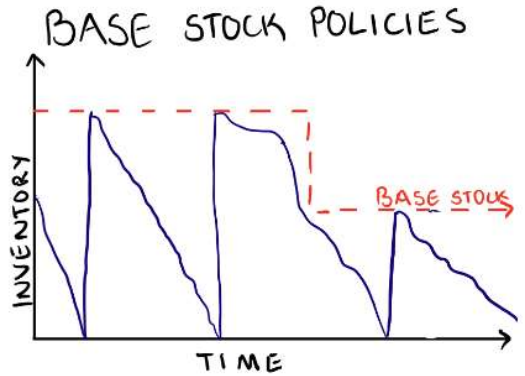


Figure 4: High-level overview of an RL-based scheduler.

[Ipek2008] Dynamic memory controllers



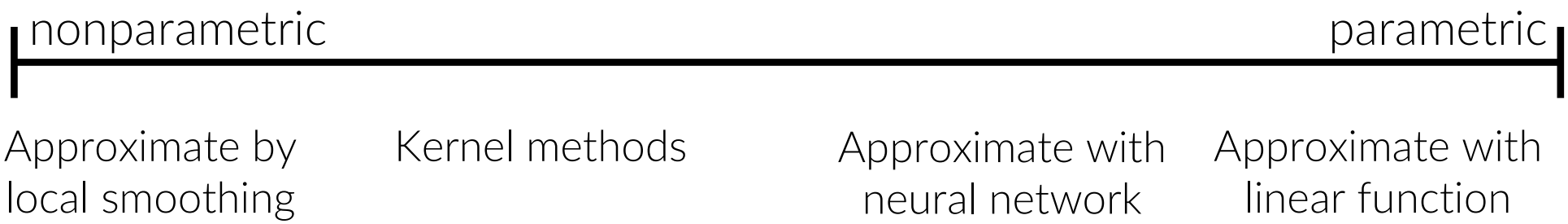
[BP2019]Controlling service rates in online service systems



[Agrawal2019] Dynamic inventory control

Adaptivity

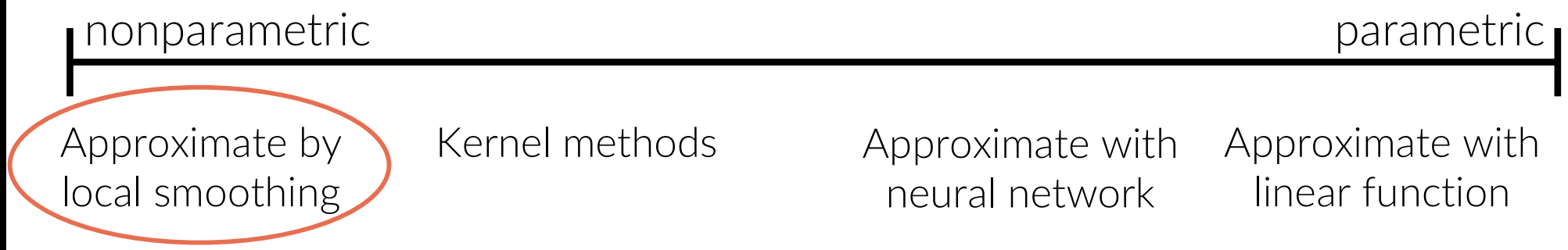
Most theoretical RL with function approximation considers worst-case problem instance, failing to exploit benign structure in the data



What is the right 'structure' for RL?

Adaptivity

Most theoretical RL with function approximation considers worst-case problem instance, failing to exploit benign structure in the data



What is the right 'structure' for RL?

Function Approximation

Talk about function approximation here and introduce adaptive discretization as the simplest to understand/analyze nonparametric function class, which essentially fits with piecewise constants, yet where the pieces are not fixed (in contrast to fixed uniform discretization)

Structures

Real applications have limitations on operation behavior of the algorithm:

- Low memory
- Low computation requirements
- Low sample complexity

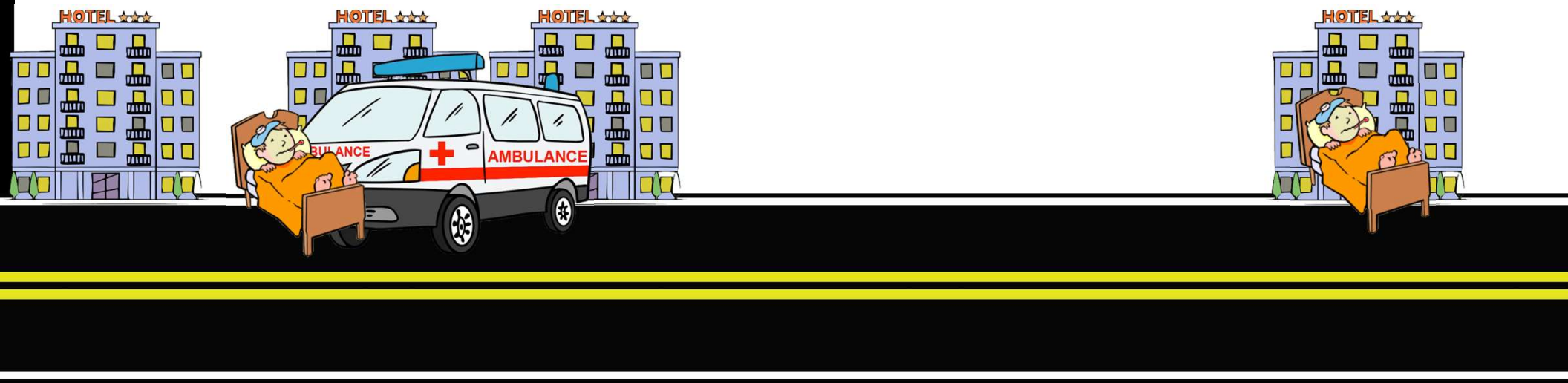
} Trifecta for RL in OR

Algorithm design typically relies heavily on:

- Optimization oracles (ERM, infinite-dimensional LP, etc)
- Completely known model + easy to evaluate (e.g. arbitrary expectations of Value function over state-distribution)

Ambulance Routing

- Operator decides location to station ambulance, paying a transportation cost
- Random request realized, ambulance pays cost for travel delay to serve patient
- Goal: learn policy which minimizes costs w/o knowledge of arrival distribution



Systems Control

Cartoon example for broader resource allocation and control problems:

- Network control policies [Liu2019]
- Controllers for short-term memory management [Ipek2008]
- Scheduling algorithms and processor assignment for data centers [Berral2010]

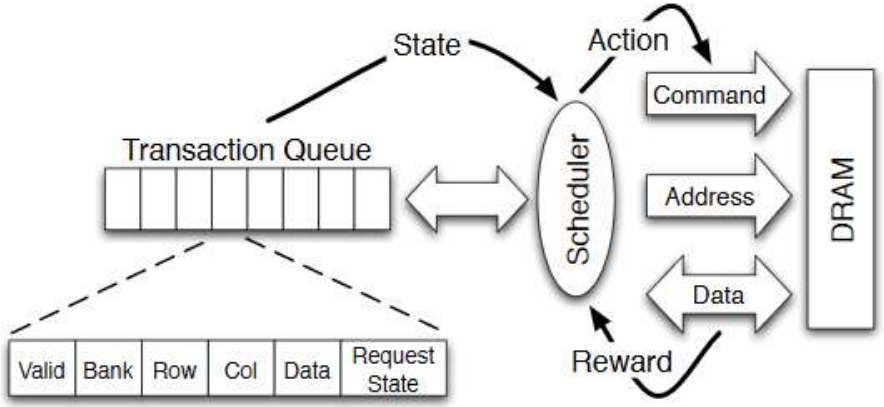


Figure 4: High-level overview of an RL-based scheduler.

Systems Control

Cartoon example for broader resource allocation and control problems:

- Network control policies [Liu2019]
- Controllers for short-term memory management [Ipek2008]
- Scheduling algorithms and processor assignment for data centers [Berral2010]

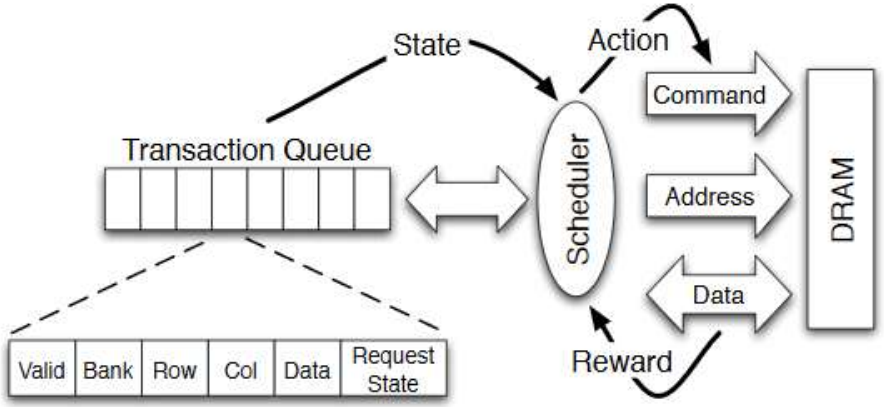
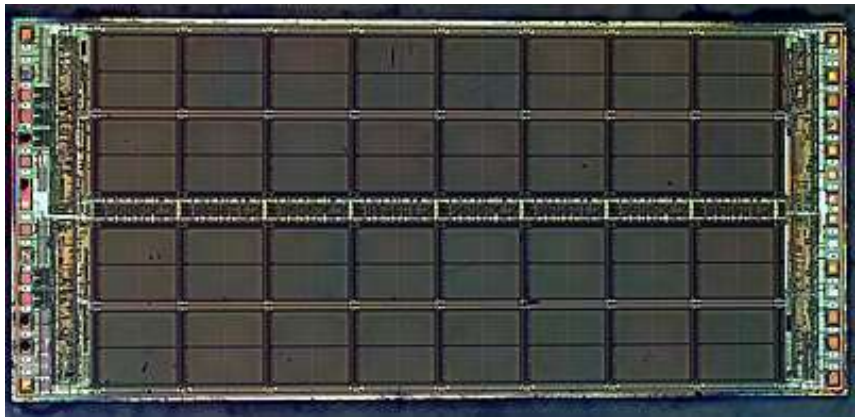


Figure 4: High-level overview of an RL-based scheduler.

Memory Management

DRAM (Dynamic Random Access Memory) – common type of memory used in computers and servers



Memory Management

DRAM (Dynamic Random Access Memory) – common type of memory used in computers and servers.

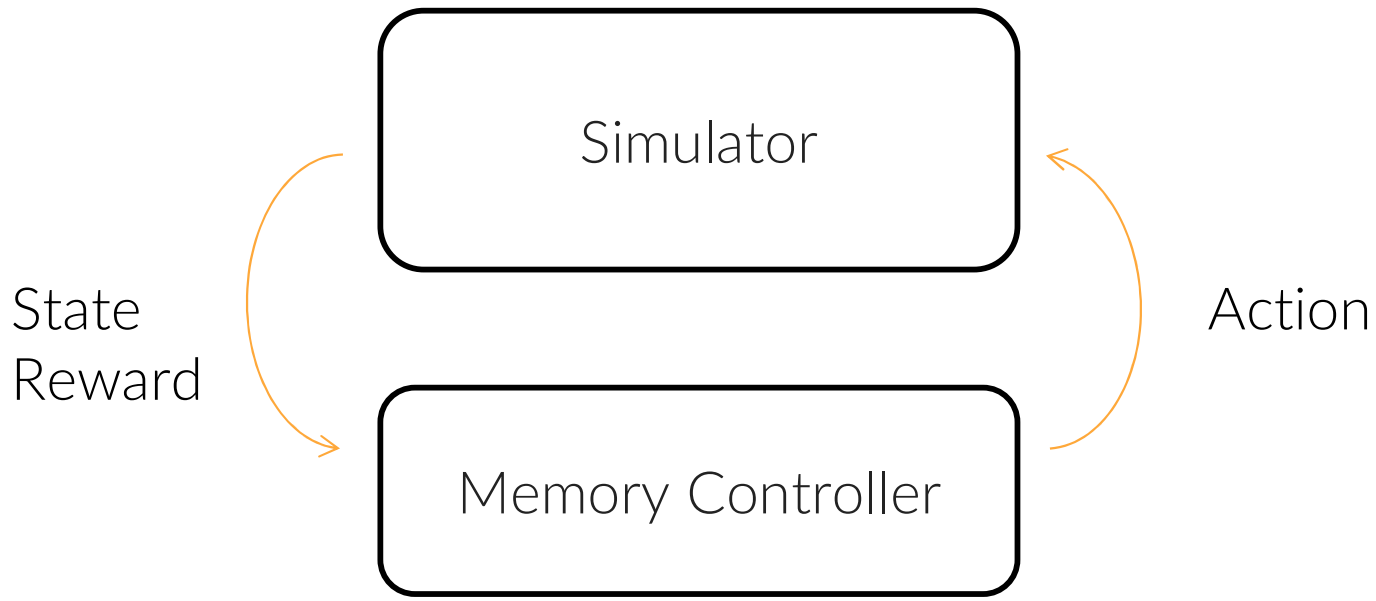
Maintains “memory transaction queue” of requests (read, write, activate, etc)

Read + Write commands must access memory and move them to “buffer” before processing command

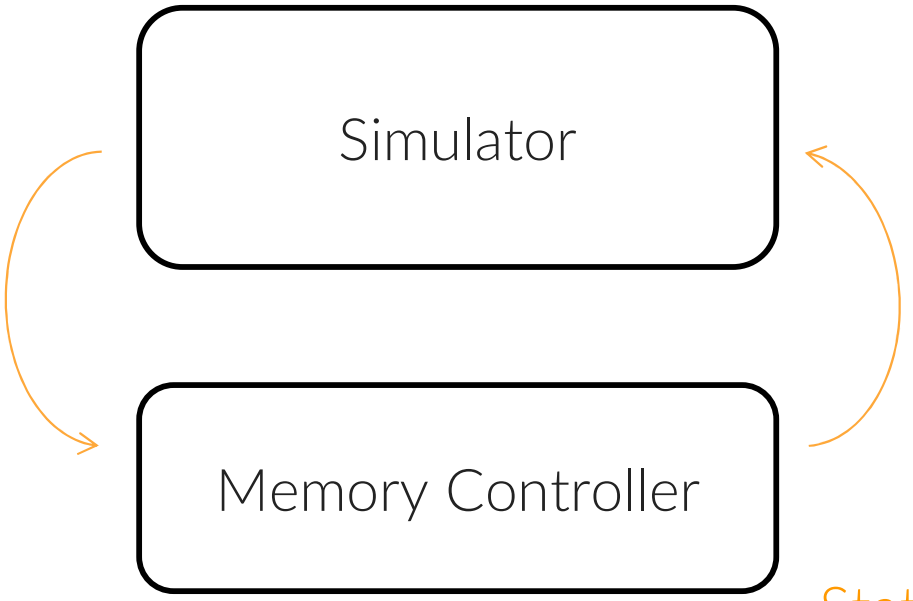
Memory controller must:

- adapt to varying workload
- satisfy timing constraints (read/write delays from hardware constraints, etc)
- prioritize certain commands to optimize system performance
- implementable *on-chip*

Memory Management



Memory Management



Action:
- Precharge
- Write
- Read
- No OP

Reward: One for read/white commands, 0 otherwise

State: Number of reads + writes in transaction queue (which are/are not load misses), etc…

(Independent of actual requests, so some ‘shaping’ done here)

Memory Management

DRAM (Dynamic Random Access Memory) – common type of memory used in computers and servers.

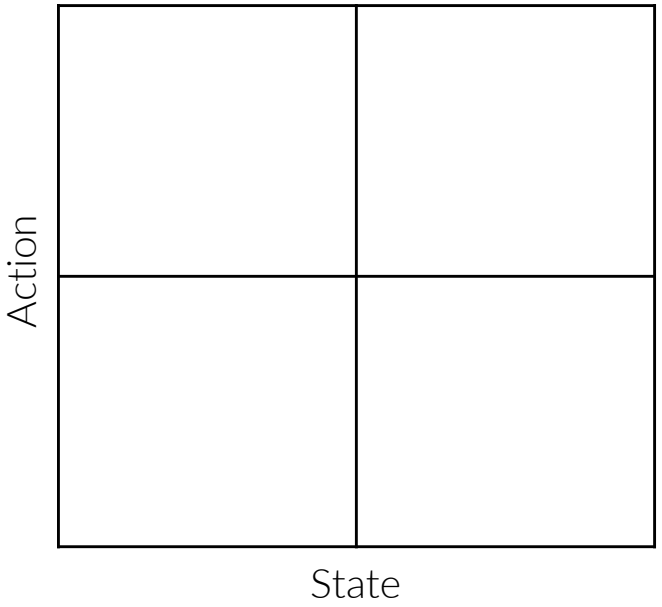
Maintains “memory transaction queue” of requests (read, write, activate, etc)

Read + Write commands must access memory and move them to “buffer” before processing command

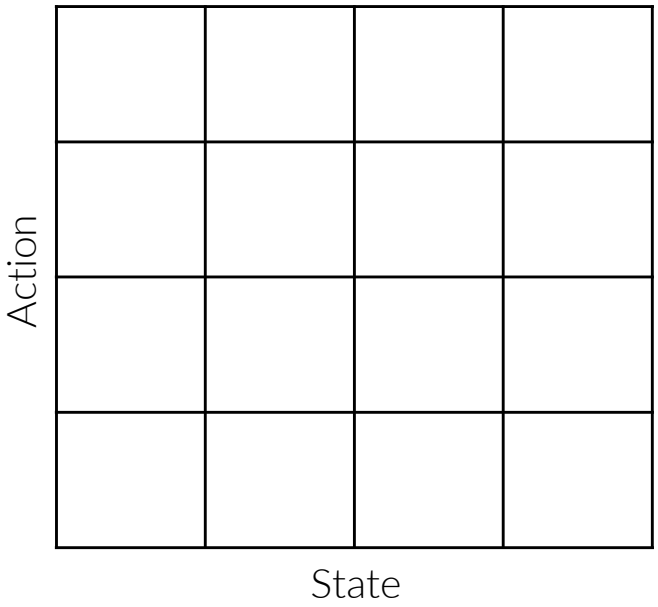
Memory controller must:

- adapt to varying workload
- satisfy timing constraints (read/write delays from hardware constraints, etc)
- prioritize certain commands to optimize system performance
- implementable *on-chip*

Memory Management

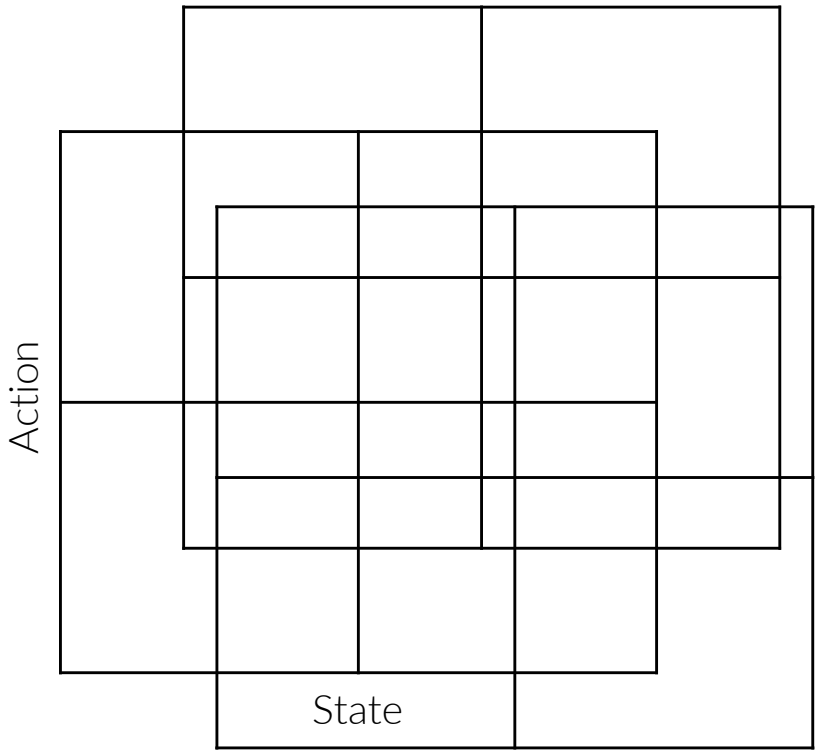


Coarse Grained



Fine Grained

Memory Management



Randomized +
Coarse Grained

CMAC:

- Balances computational efficiency and resolution accuracy
- Leads to unique on-chip implementation (hashing)
- Average estimates across regions containing specific state-action pair

This is just a randomized but kind of fixed/uniform discretization?

Systems Control

Inherent to these problems are:

- Large or continuous state and action spaces
- Operating constraints on the execution of the algorithms (simple enough to implement in hardware)
- No known structure a priori

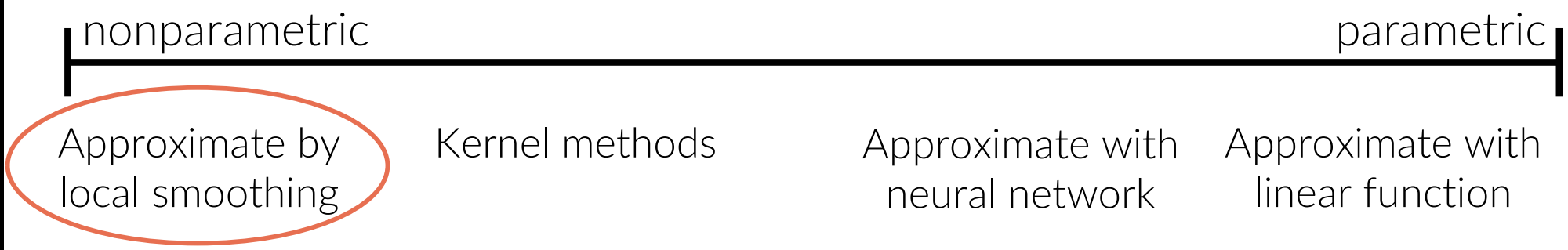
Motivates non-parametric approaches with simple implementation

Lipschitz MDP **Adaptive Discretization** **Regret Bound**

Design model-free and model-based Adaptive Discretization algorithms for Lipschitz MDPs, discuss bound on: regret, time complexity, and storage complexity.

Lipschitz MDPs

Most theoretical RL with function approximation considers worst-case problem instance, failing to exploit benign structure in the data

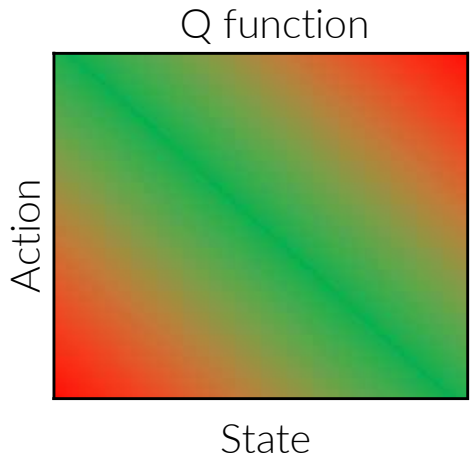


What is the right 'structure' for RL?

Lipschitz MDPs

A Lipschitz MDPs is one where $\mathcal{S}, \mathcal{A}$ are compact metric spaces with metric $\mathcal{D}$

- Optimal $Q_h^*(s, a)$ is Lipschitz continuous wrt $\mathcal{D}$
- Reward $r_h(s, a)$ is Lipschitz wrt $\mathcal{D}$ and transitions distribution $T_h(\cdot \mid s, a)$ is Lipschitz wrt d_W



Lipschitz MDPs

A Lipschitz MDPs is one where $\mathcal{S}, \mathcal{A}$ are compact metric spaces with metric $\mathcal{D}$

- Optimal $Q_h^*(s, a)$ is Lipschitz continuous wrt $\mathcal{D}$
- Reward $r_h(s, a)$ is Lipschitz wrt $\mathcal{D}$ and transitions distribution $T_h(\cdot \mid s, a)$ is Lipschitz wrt d_W

Note: Second condition implies the first, with Lipschitz constant $\sum_{h=0}^{H} L_r L_T^h$

Lipschitz constant tied to time horizon, not observed in contextual bandits

Lipschitz MDPs

Reasonableness of Lipschitz assumption:

- Most OR problems have deterministic transitions
- Linear/convex rewards

Mild nonparametric assumptions, guarantees degrade nicely w.r.t “closeness” to Lipschitz function

Uniform Discretization

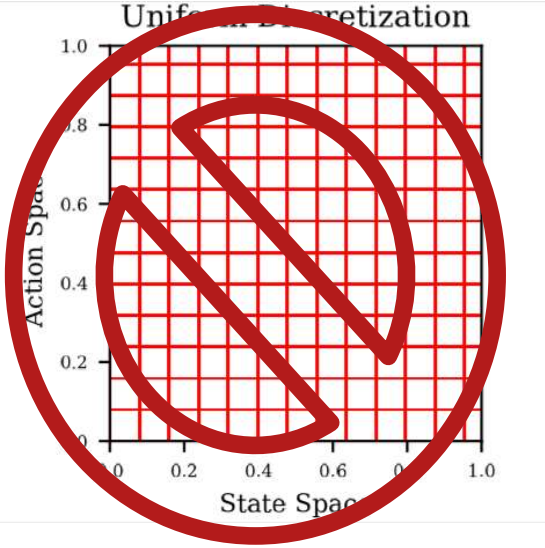
Naive Solution: Discretize the state-action space uniformly

Continuous MDP

Uniform ϵ_T-net discretization

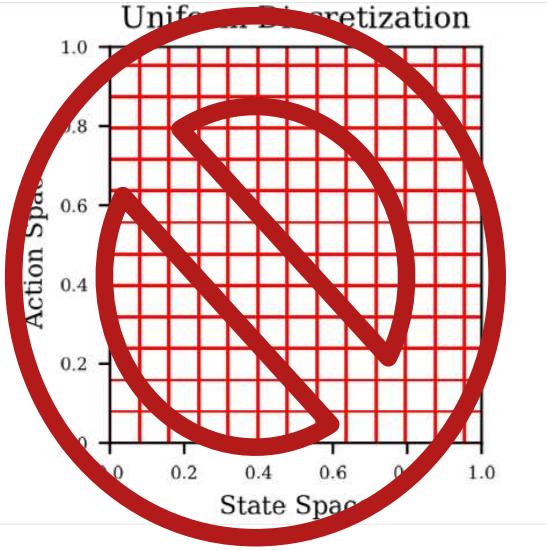
Tabular MDP

Choose ϵ_T as a fn of time horizon to balance approximation error + cost of solving the tabular MDP



- Creates unnecessary exploration
- Wastes memory by discretizing suboptimal parts of the space
- Learned policy does not adapt to the underlying ***geometry*** of the space

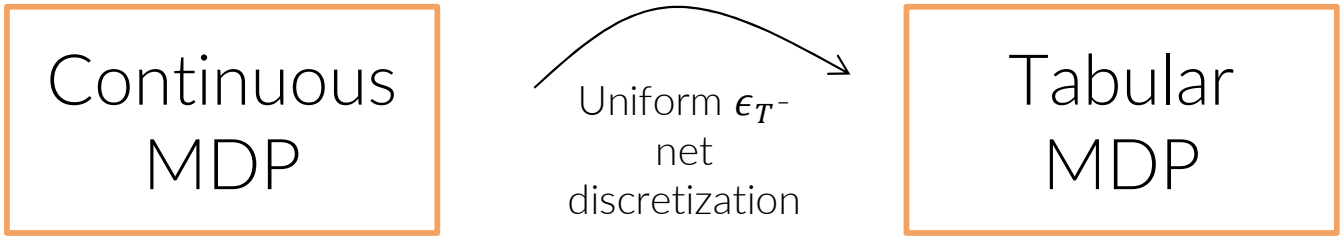
Uniform Discretization



1. (*Fixed Representation):* Representation is fixed, wastes memory storing suboptimal parts
2. (*Generate Estimates):* Tabular algorithms are easy to implement
3. (*Selection Rule):* Tabular algorithms are easy to implement

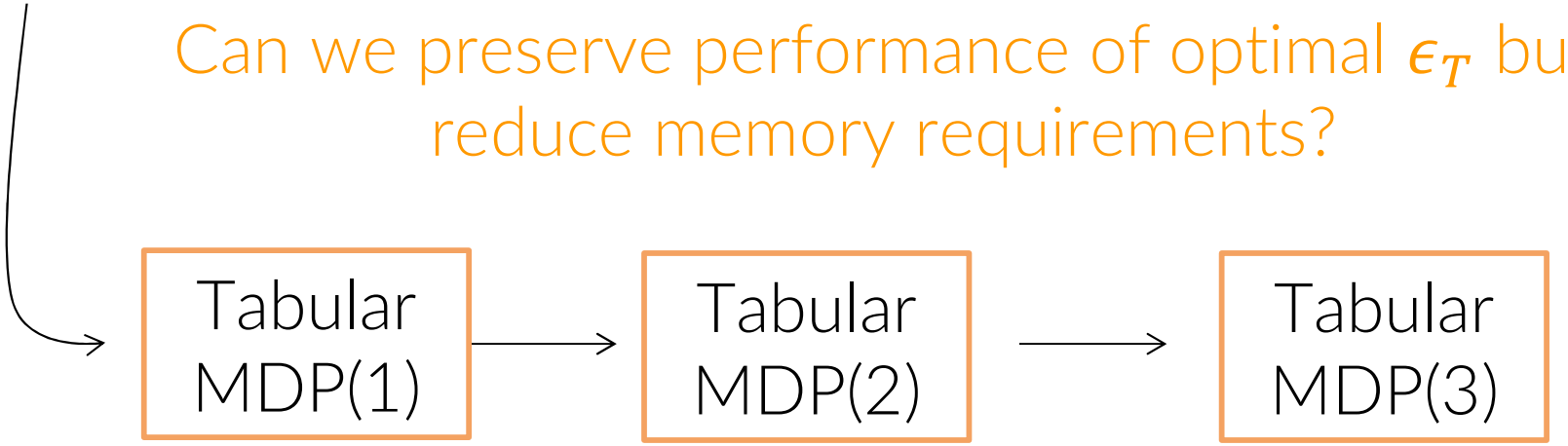
Adaptive Discretization

Naive Solution: Discretize the state-action space uniformly



Choose ϵ_T as a fn of time horizon to balance approximation error + cost of solving the tabular MDP

Can we preserve performance of optimal ϵ_T but reduce memory requirements?



Only refine discretization on an "as needed" basis.

Adaptive Discretization

Algorithms are based on:

1. *(Adaptive Partition)*: Maintaining an adaptive partition of the state-action space
2. (*Generate Estimates)*: Create *optimistic* estimates for Q_h^* over the partition
3. (*Selection Rule)*: Executing the optimal policy based on the estimates.
4. *(Refinement Rule)*: Decide whether to continue or to refine the partition

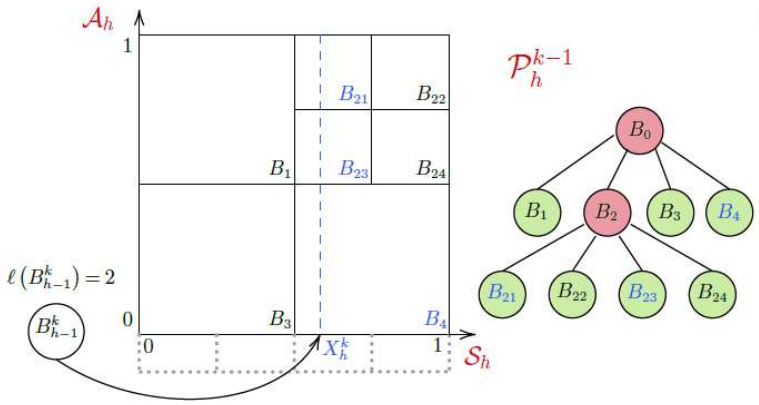


Figure 0 Illustrating the state-action partitioning scheme

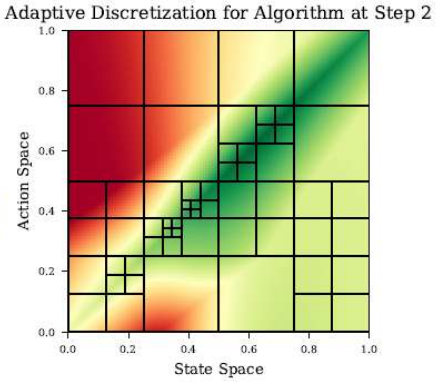
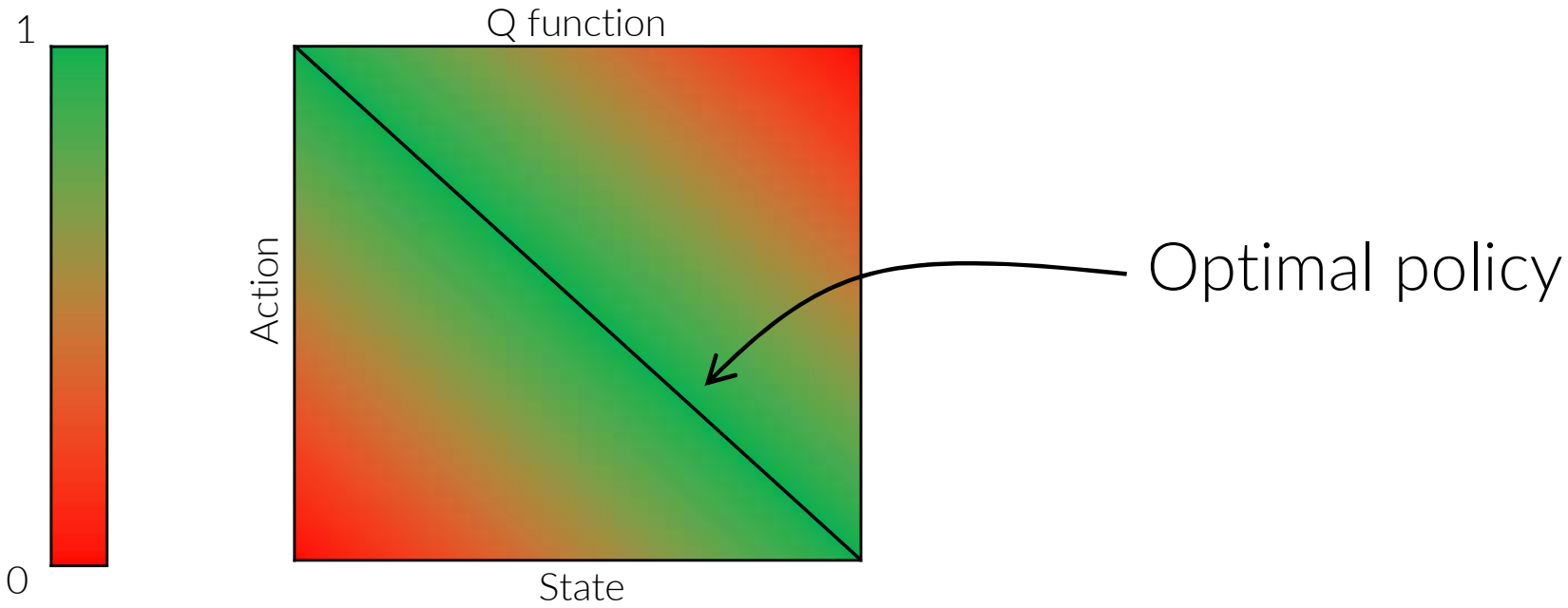


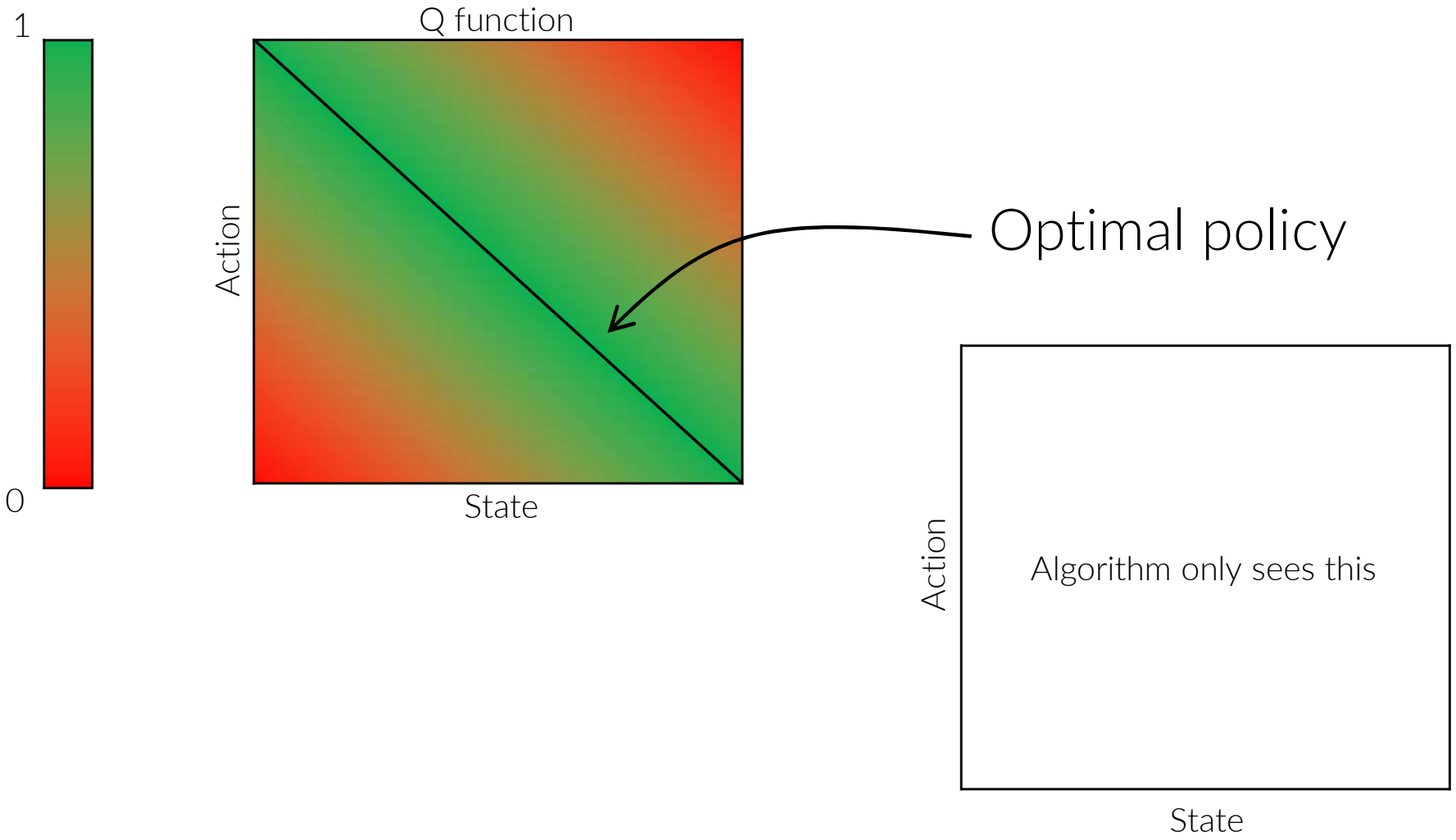
Figure 0 Partitioning in practice

Adaptive Discretization

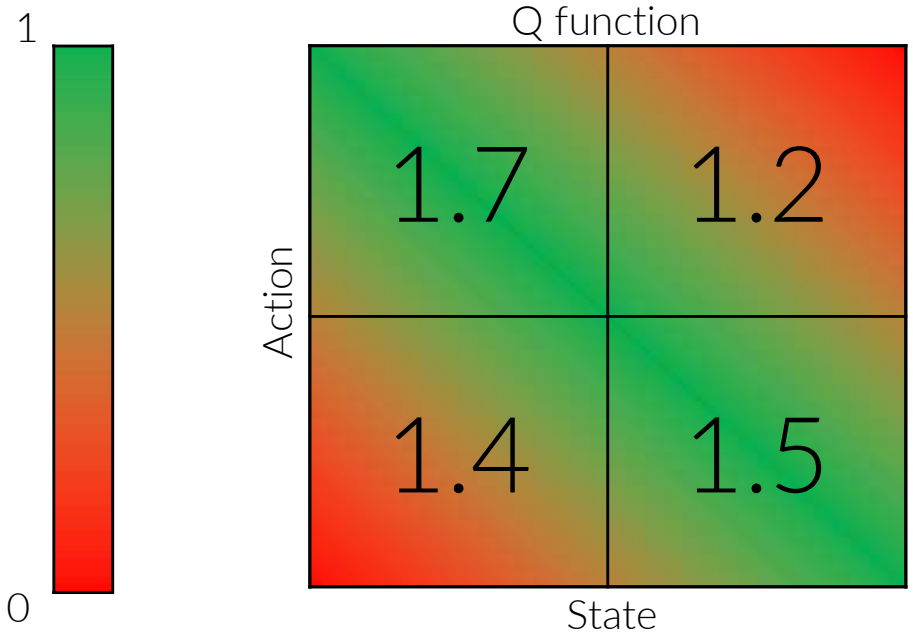


$$Q_h^*(s,a) = r_h(s,a) + \mathbb{E}[\max_{a' \in \mathcal{A}} Q_{h+1}^\star(S', a') \mid s, a]$$

Adaptive Discretization

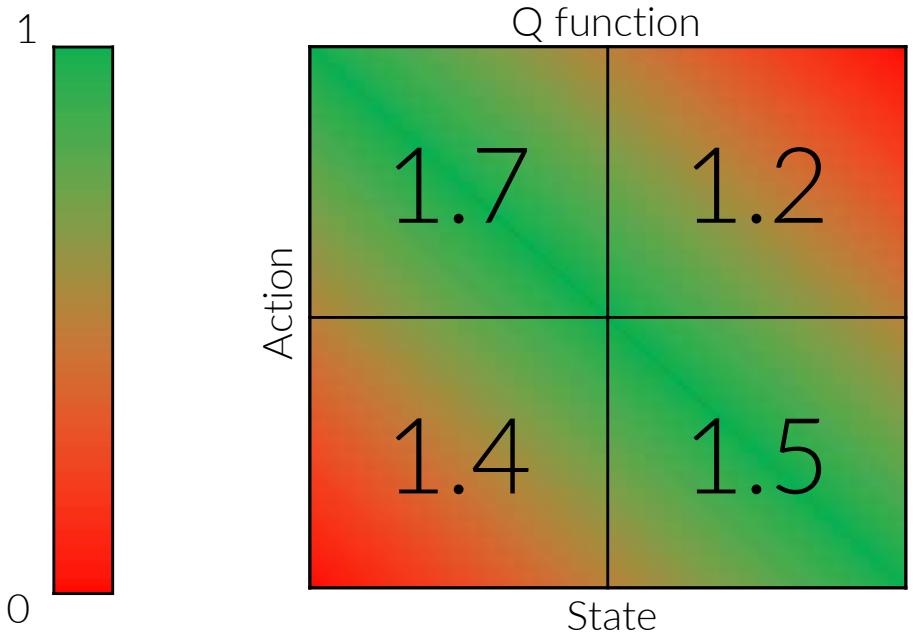


Adaptive Discretization



- *(Adaptive Partition):* Begin with coarse discretization of space

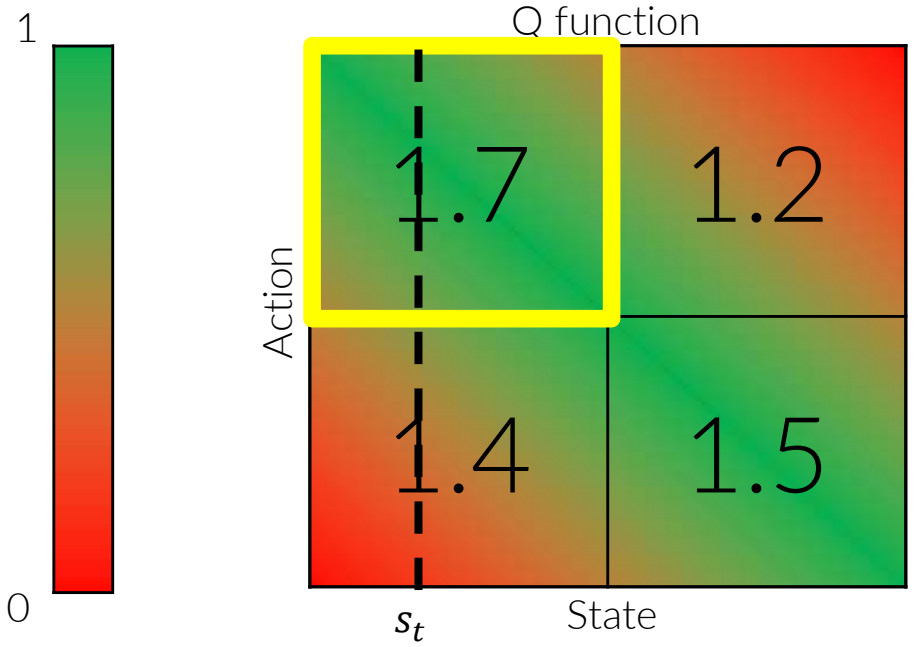
Adaptive Discretization



- *(Adaptive Partition):* Begin with coarse discretization of space
- *(Generate Estimates):* Maintain UCB estimates of Q function across regions

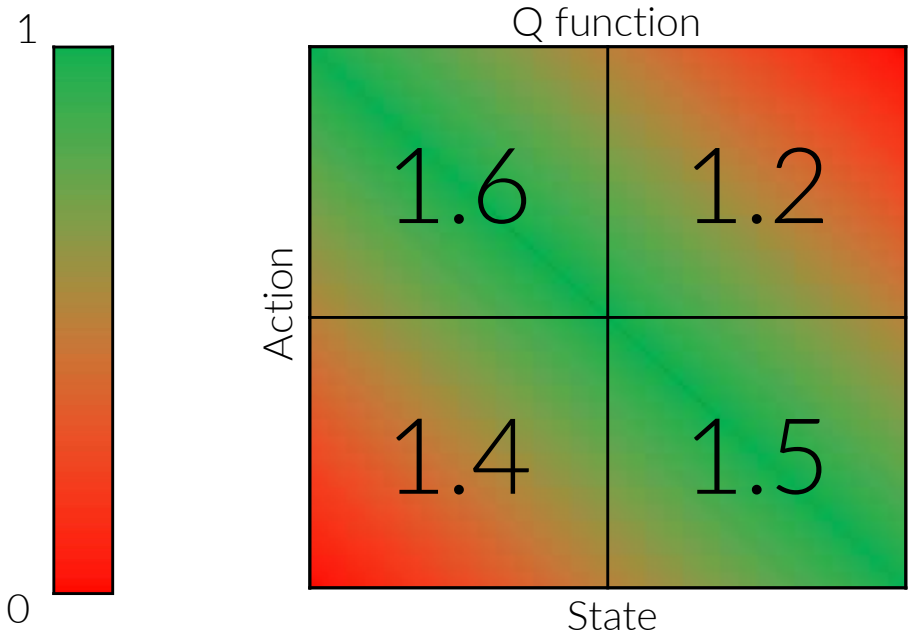
Mimic the tabular counterpart but aggregate estimates collected over a region

Adaptive Discretization



- *(Adaptive Partition):* Begin with coarse discretization of space
- *(Generate Estimates):* Maintain UCB estimates of Q function across regions
- *(Selection Rule):* Select "relevant" region that maximizes UCB for state

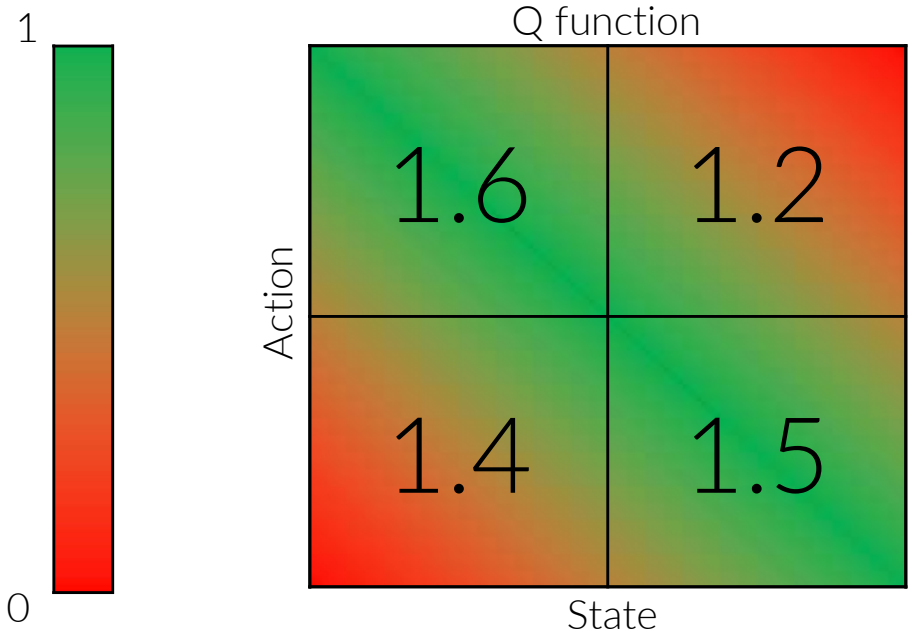
Adaptive Discretization



- *(Adaptive Partition):* Begin with coarse discretization of space
- *(Generate Estimates):* Maintain UCB estimates of Q function across regions
- *(Selection Rule):* Select "relevant" region that maximizes UCB for state
- *(Update Estimates):* Update estimates with newly collected data:

$$(s_h, a_h, s_{h+1}, r_h)$$

Adaptive Discretization



- *(Update Estimates)*: Update estimates with newly collected data:

$$(s_h, a_h, s_{h+1}, r_h)$$

Model-Free:

$$\overline{Q}_h(B_{sel}) = (1-\alpha_t)\overline{Q}_h(B_{sel}) + \alpha_t(r_h + \textsc{Conf}(B_{sel}) + \max_{x_{h+1} \in B} \overline{Q}_{h+1}(B))$$

Empirical Bellman Optimality Equation

Model-Based: More complicated…as we require estimates of:

Average Rewards

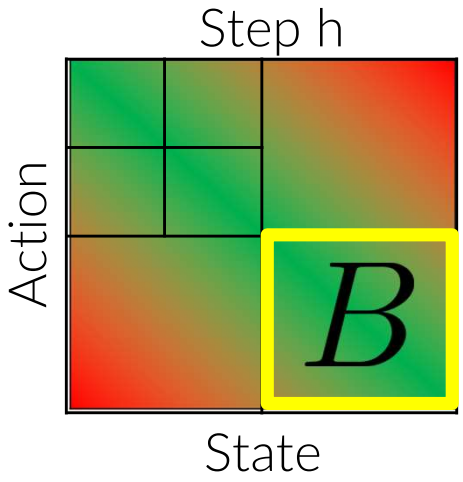
Easy – estimate as average reward of samples from region

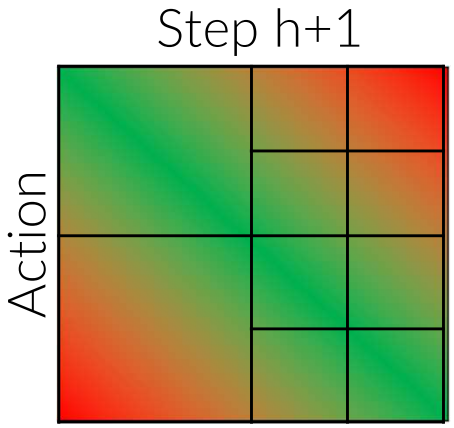
Transition Distribution

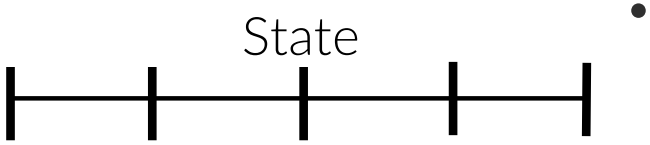
Tough – domain is region, range is state space

$$\overline{T}_h(\cdot \mid B)$$

Adaptive Discretization



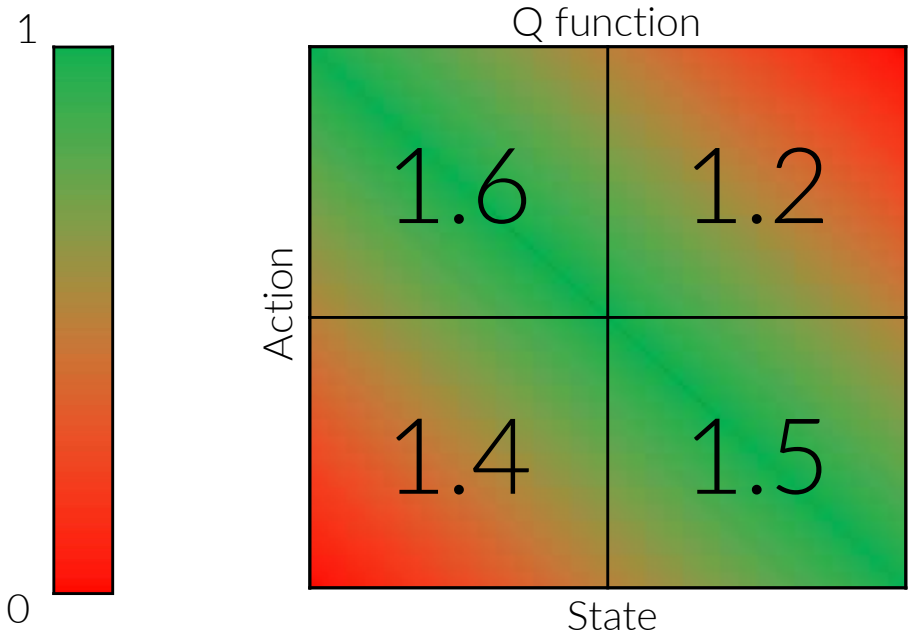




Uniform State Discretization

- Estimate transition function over a uniform-discretization of the state space up to the precision of a ball
- Ensures accuracy of the estimate is proportional to the diameter
- Limits storage complexity for the estimates

Adaptive Discretization



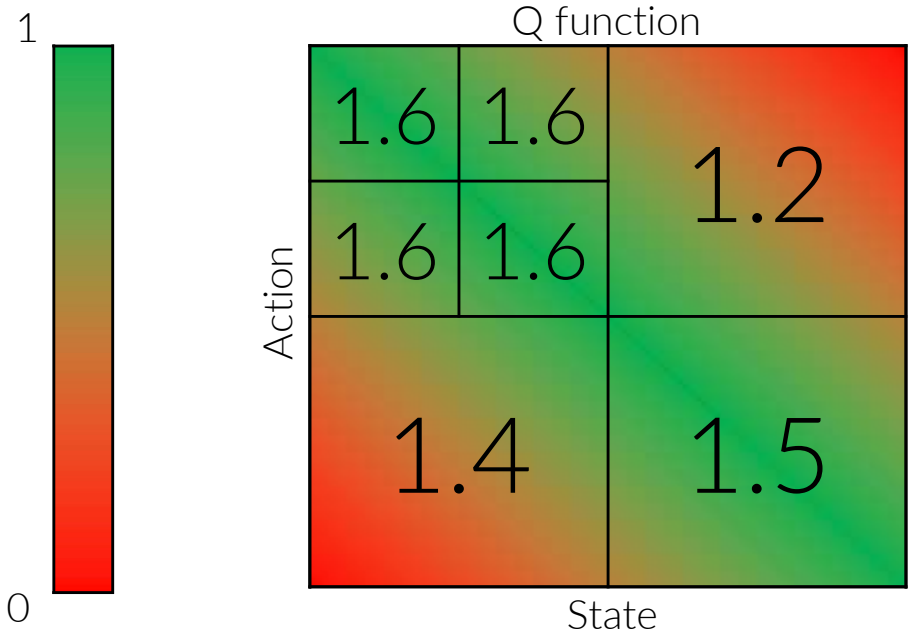
- *(Update Estimates)*: Update estimates with newly collected data:

$$(s_h, a_h, s_{h+1}, r_h)$$

Model-Based:

$$\overline{Q}_h(B) = \overline{r}_h(B) + \overline{\mathbb{E}}(\max_{X' \in B} \overline{Q}_{h+1}(B) \mid B) + \textsc{Conf}(B)$$

Adaptive Discretization



- *(Splitting Rule):* Sub-partition a region when uncertainty in estimate is smaller than bias, i.e.:

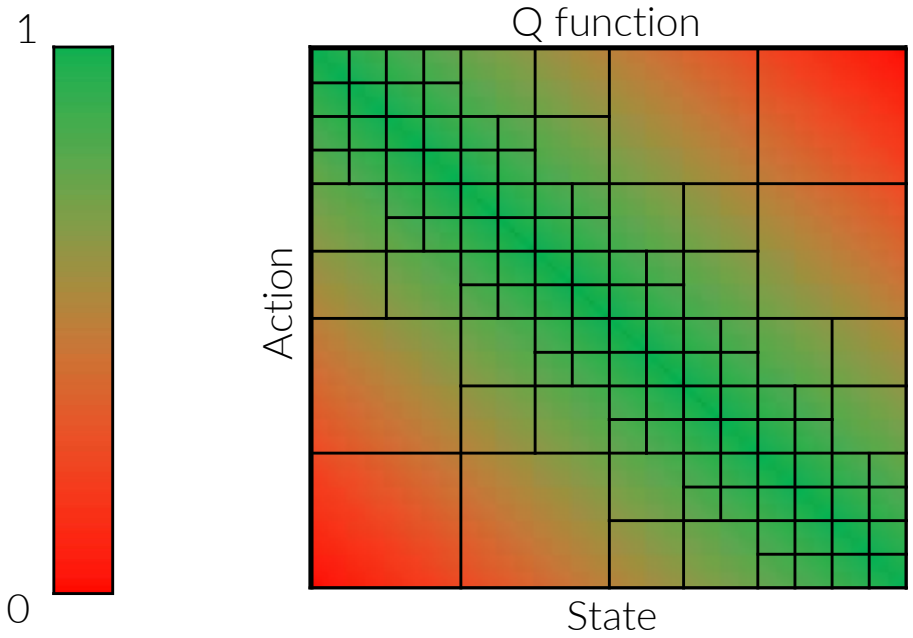
$$\textsc{Conf}_h^k(B) \leq \textsc{diam}(B)$$

Uncertainty/confidence in estimates depends on:

- Method used to construct estimates (i.e. model-free vs model-based)
- Concentration inequalities used

By Lipschitz property, bias is proportional to size of region

Adaptive Discretization



- Process repeats until end of the episodes
- Or stop refining discretization based on available compute and memory

Regret Bound

1. A region is subpartitioned when it leads to high rewards, or reduce downstream estimation errors

   Clipping estimates by performance, additional factor for down-stream effect

2. Constructing estimates of Q function is technical for model-based algorithm (domain of transition is state-action space, range is state space only)

   "Minimum" data required to store estimate of transition kernel

3. Performance of the algorithm is proportional to the size of the partition – but bounding size of partition requires dealing with errors across steps

   LP based arguments with Kraft's inequality

Regret Bound

Theorem: For any sequence of initial states, our algorithms achieve regret guarantees:

$$R(K) \leq \begin{cases} \textsc{adaQL}: & H^{5/2} K^{\frac{z+1}{z+2}} \\ \textsc{adaMB}: & H K^{\frac{z+d_S-1}{z+d_S}} \quad d_S > 2 \\ \textsc{adaMB}: & H K^{\frac{z+d_S+1}{z+d_S+2}} \quad d_S \leq 2 \end{cases}$$

d is the covering dimension of the state-action space, d_S is the covering dimension of the state space, and $z \leq d$ is the 'zooming dimension'.

- Optimal dependence on the zooming dimension for the model-free algorithm
- Contrasting story to the tabular case / folklore, where model-based algorithms have better regret and empirical performance than model-free
- Can also provide explicit bounds on computation and storage requirements

Structures

Real applications have limitations on operation behavior of the algorithm:

- Low memory
- Low computation requirements
- Low sample complexity

} Trifecta for RL in OR

Adaptive Discretization algorithms provide concrete bounds across all three dimensions

Zooming Dimension

Definition: For any state action pair and step, the stage-dependent suboptimality gap is

$$\textsc{Gap}_h(x,a) = V_h^\star(x) - Q_h^\star(x,a)$$

The near-optimal set for a specific problem and radius is defined as

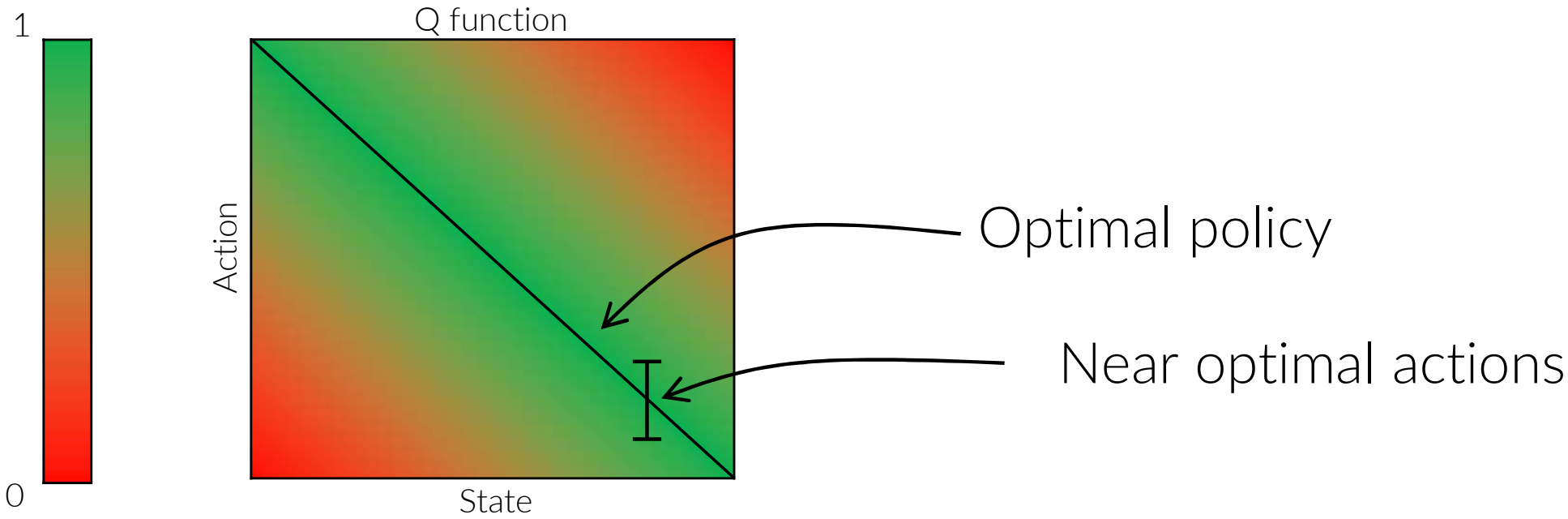
$$Z_h^r = \{(x,a) \in \mathcal{X} \times \mathcal{A} \mid \textsc{Gap}_h(x,a) \leq 2(H+1)r\}$$

The zooming dimension $\boldsymbol{z_h}$ is the covering dimension of this set.

$$z_h = \min\{d > 0 : N_r(Z_h^r) \leq c_h r^{-d} \, \forall r > 0\}$$

Leads to 'instance-dependent' optimality
Only *minimal* improvements (think linear)

Zooming Dimension



Example where covering dimension is 2, but zooming dimension is 1

Proof Sketch

1. A region is subpartitioned when it leads to high rewards, or reduce downstream estimation errors

   Clipping estimates by performance, additional factor for down-stream effect

2. Constructing estimates of Q function is technical for model-based algorithm (domain of transition is state-action space, range is state space only)

   "Minimum" data required to store estimate of transition kernel

3. Performance of the algorithm is proportional to the size of the partition – but bounding size of partition requires dealing with errors across steps

   LP based arguments with Kraft's inequality

Concentration

Algorithms maintain estimates of reward (both) and transitions (only AdaMB). Use standard concentration inequalities to show that:

$$|\overline{r}_h^k(B) - r_h(x,a)| \leq \textsc{Rucb}_h^k(B) + L_r \textsc{Diam}(B)$$

$$d_W(\overline{T}_h^k(\cdot \mid B) - T_h(\cdot \mid x,a)) \leq \textsc{Tucb}_h^k(B) + L_T \textsc{Diam}(B)$$

Bias in region for aggregation, RUCB / TUCB for standard bonus terms

Optimism

Use concentration to establish *optimism*, typical property in online algorithms. Recall:

$$\overline{Q}_h(B) = \overline{r}_h(B) + \overline{\mathbb{E}}(\max_{X' \in B} \overline{Q}_{h+1}(B) \mid B) + \textsc{Conf}(B)$$

Can establish that with high probability:

$$0 \leq \overline{Q}_h^k(B) - Q_h^\star(x,a) \leq \textsc{Conf}(B) + \textsc{Bias}(B) + f_{h+1}^k$$

Current step mis-estimation

Downstream mis-estimation

Clipping

Clipping argument first used for tabular. Clip current step misestimation by gap of region, account for clip in downstream errors

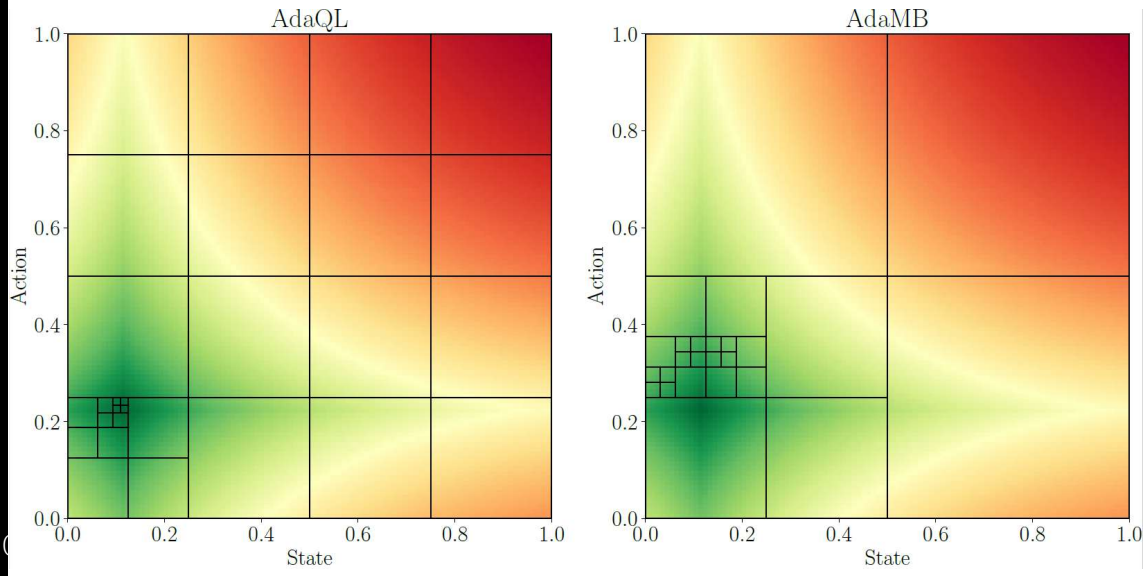
$$\textsc{Gap}_h(B) = \min_{(x,a)\in B} V_h^\star(x) - Q_h^\star(x,a)$$

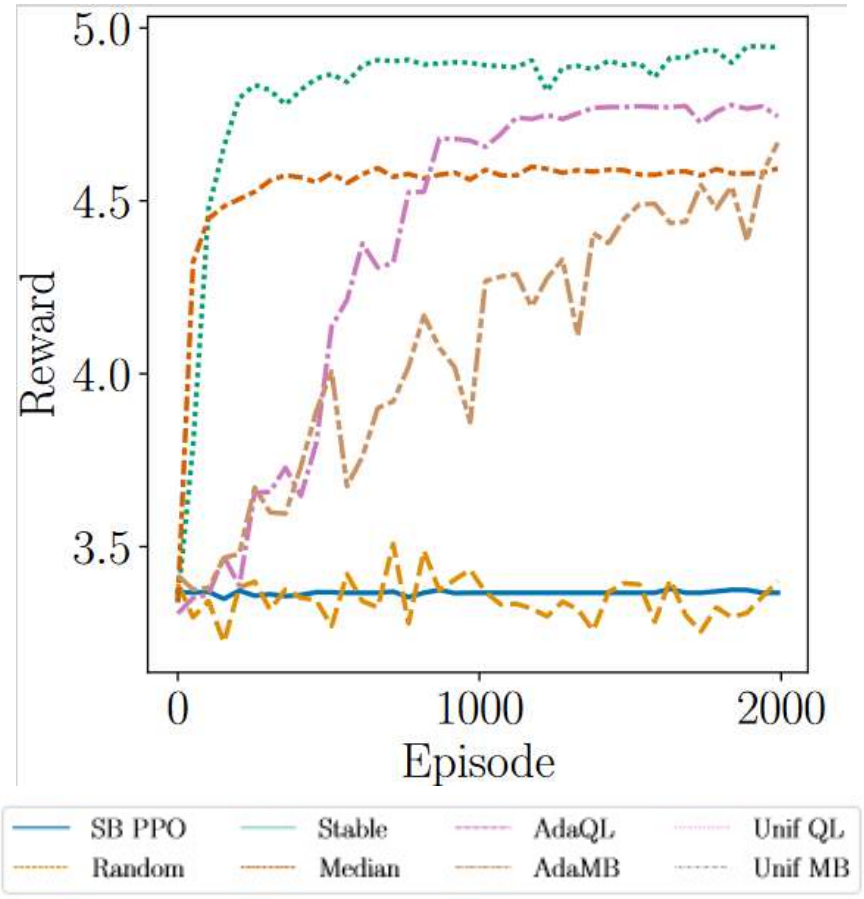
Current step mis-estimation

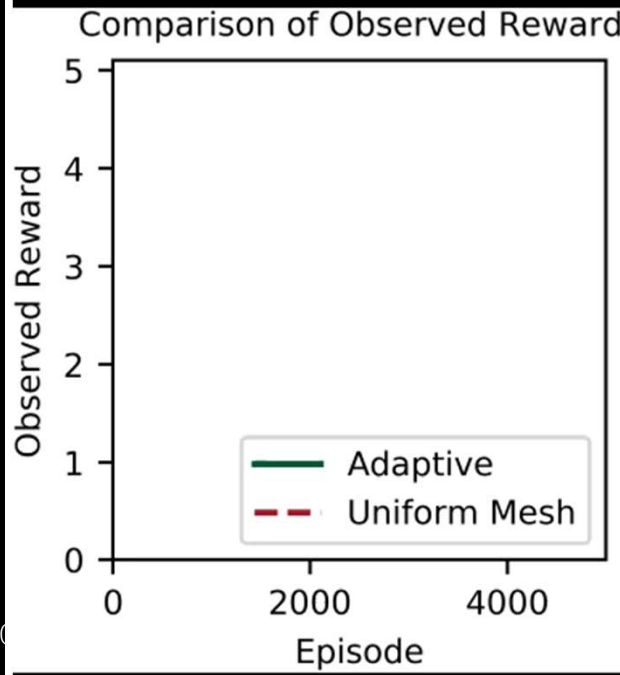
$$\overline{Q}_h^k(B) - Q_h^\star(x,a) \le \textsc{Clip}\left[\textsc{Conf}(B) + \textsc{Bias}(B) \mid \frac{\textsc{Gap}_h(B)}{H+1}\right] + \left(1 + \frac{1}{H+1}\right) f_{h+1}^k$$

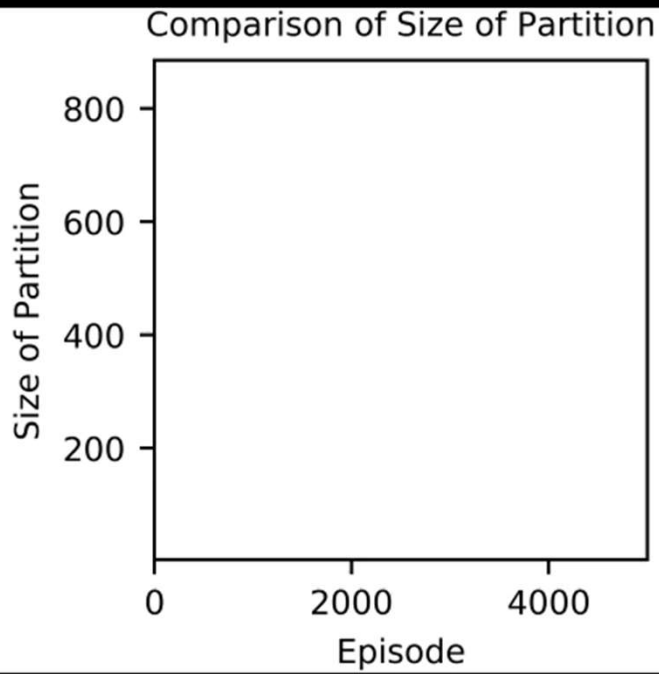
Downstream mis-estimation

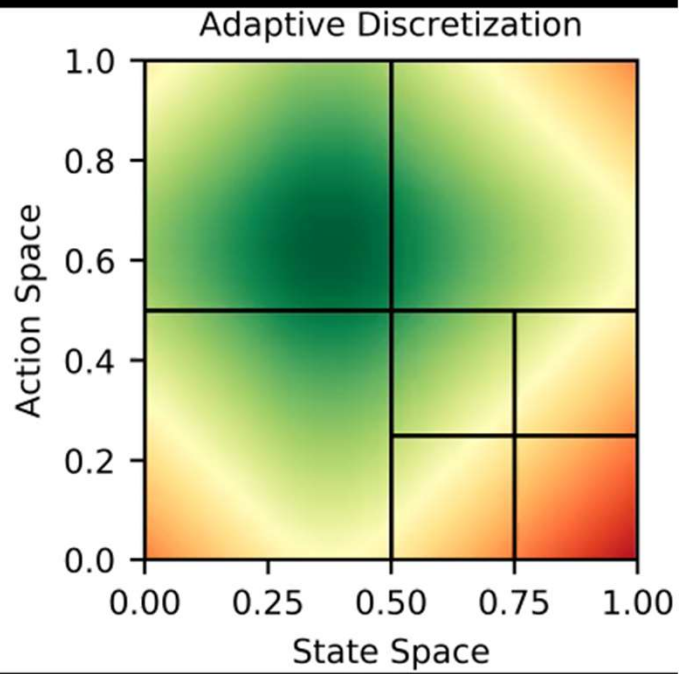
Numerical Results






Comparison of Observed Rewards
Observed Reward
Episode
Adaptive
Uniform Mesh
0
1
2
3
4
5
0
2000
4000


Comparison of Size of Partition
Size of Partition
Episode
200
400
600
800
0
2000
4000


Adaptive Discretization
Action Space
State Space
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00

Open Questions

- Discrepancy in performance between model-based and model-free
- “Optimal” space and time complexities
- Incorporating hardware techniques for practical applications in small-compute regimes

References

[Sinclair2019] Sean R. Sinclair, Siddhartha Banerjee, Christina Lee Yu. "Adaptive Discretization for Episodic Reinforcement Learning in Metric Spaces." *POMACS*, 2019.

[Sinclair2020] Sean R. Sinclair, Tianyu Wang, Gauri Jain, Siddhartha Banerjee, Christina Lee Yu. "Adaptive Discretization for Model-Based Reinforcement Learning." *NeurIPS*, 2020.

[Sinclair2022] Sean R. Sinclair, Siddhartha Banerjee, Christina Lee Yu. "Adaptive Discretization in Online Reinforcement Learning." *Under Review*, *arXiv*. 2022

[Domingues2021] Omar Darwiche Domingues, Pierre Menard, Matteo Pirotta, Emilie Kaufmann, Michal Valko. "Kernel-Based Reinforcement Learning: A Finite-Time Analysis." *ICML*, 2021.

[Song2019] Zhao Song, Wen Sun. "Efficient Model-Free Reinforcement Learning in Metric Spaces.". *arXiv*. 2019.

References

[Liu2019] Joshua Comden et al. "Online Optimization in Cloud Resource Provisioning: Predictions, Regret, and Algorithms." *POMACS*, 2019.

[Ipek2008] Engin Ipek, Onur Mutlu, Jose Martinez, Rich Caruana. "Self Optimizing Memory Controllers: A Reinforcement Learning Approach." *ACM SIGARCH*, 2008.

[Berral2010] Josep Berral, Ricard Gavalda, Jordi Torres. "Adaptive Scheduling on Power-Aware Managed Data-Centers Using Machine Learning." *IEEE Workshop on Grid Computing*. 2010.