

# Feedback-based Tree Search for Reinforcement Learning

---

Daniel R. Jiang, *University of Pittsburgh*

Emmanuel Ekwedike, *Tencent AI Lab & Princeton University*

Han Liu, *Tencent AI Lab & Northwestern University*

**July 11, 2018**

**ICML 2018, Stockholm, Sweden**

# Outline

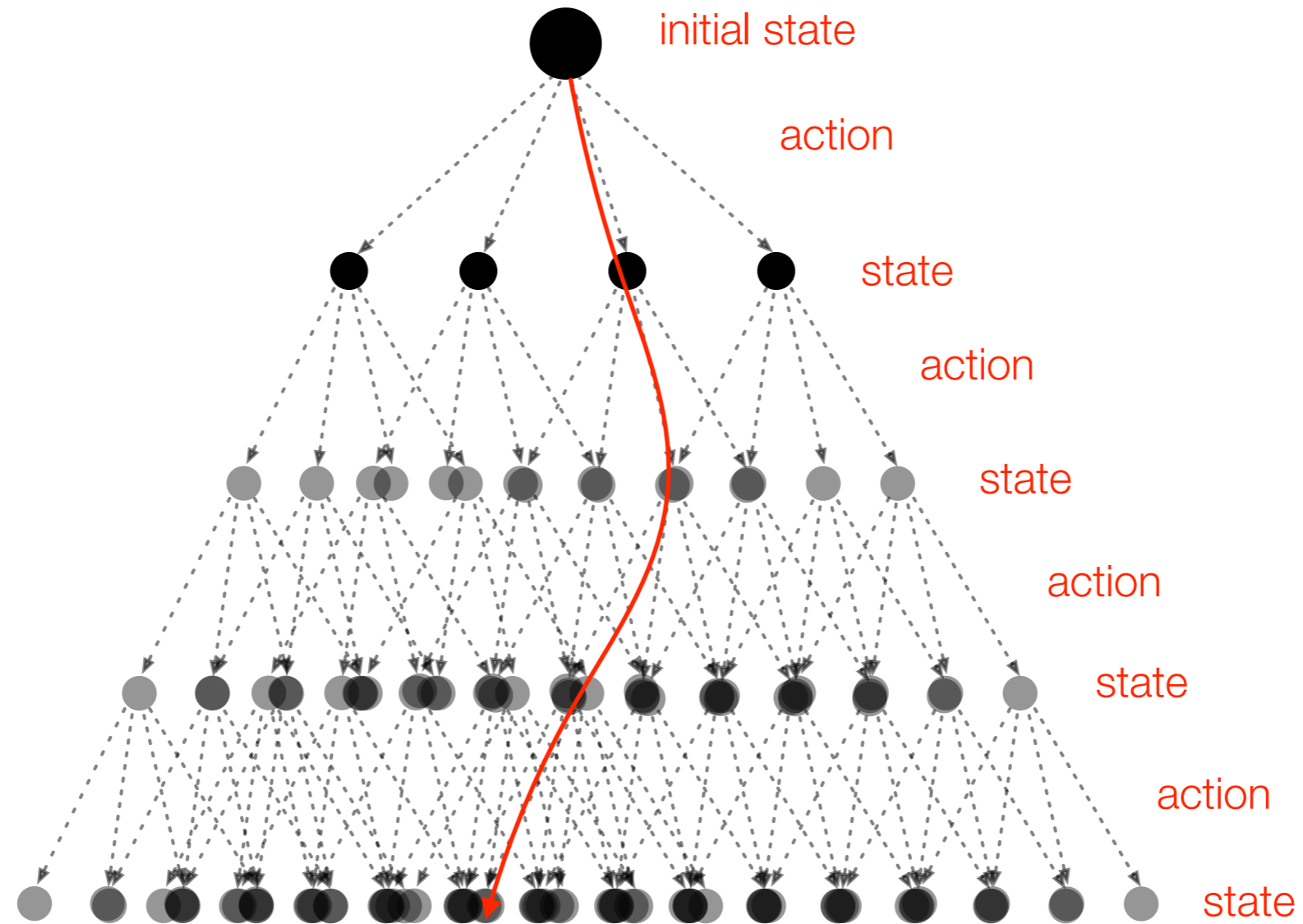
---

1. Introduction to MCTS
2. Feedback-based Tree Search
  - Inspired by recent successes of mixing RL and MCTS
3. Theoretical Analysis
4. Empirical Results on MOBA Game

# Core Component of MCTS: Decision Trees

“state, action, new state... up to a finite horizon”

---



# Monte-Carlo Tree Search

“iterative decision trees in real-time”

- MCTS is a technique for solving sequential problems (e.g., MDPs or two-player games).
- Given an initial state, it **iteratively** builds the decision tree.
- Focuses on **important states and actions**, (i.e., ones that an optimal policy might visit).
- A heuristic called the **default policy** provides Monte-Carlo estimates of downstream values to guide the tree expansion process.
- The hope is that the optimal action at the root node recommended by the **partial tree** is nearly optimal.

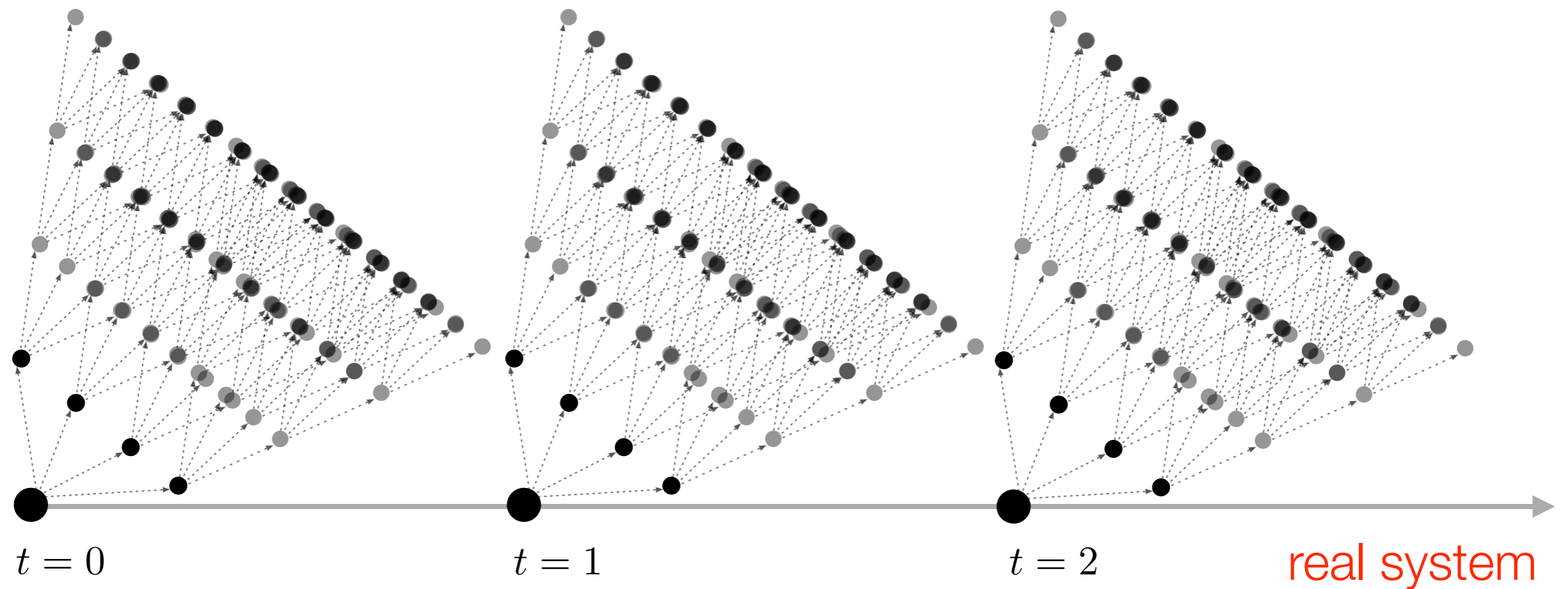


# Decision Trees *in Real Time*

“looking ahead from the current node in a rolling manner”

---

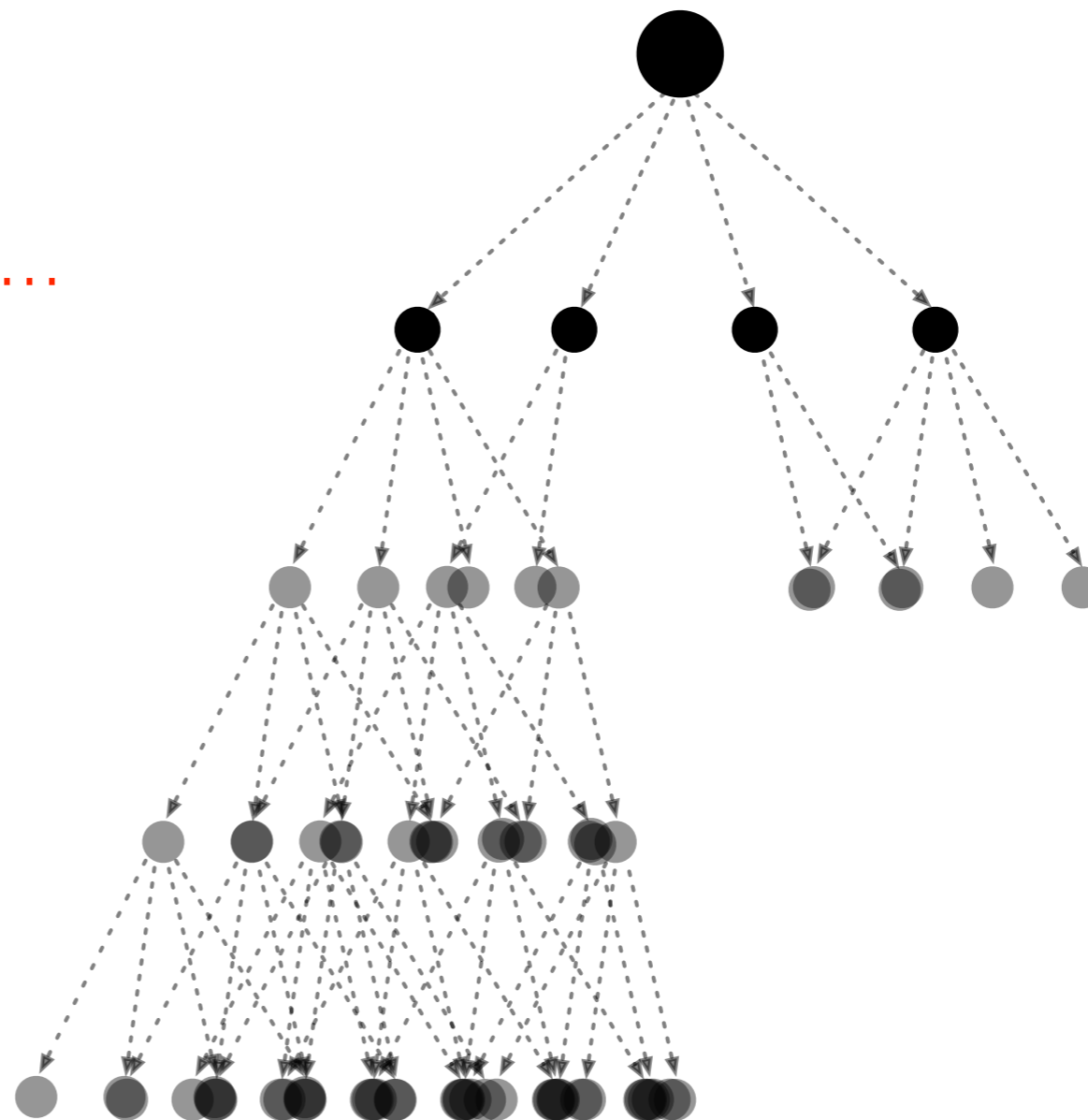
“simulated futures”



# Asymmetric Decision Trees

---

Under a good default policy...



# Main Steps of MCTS

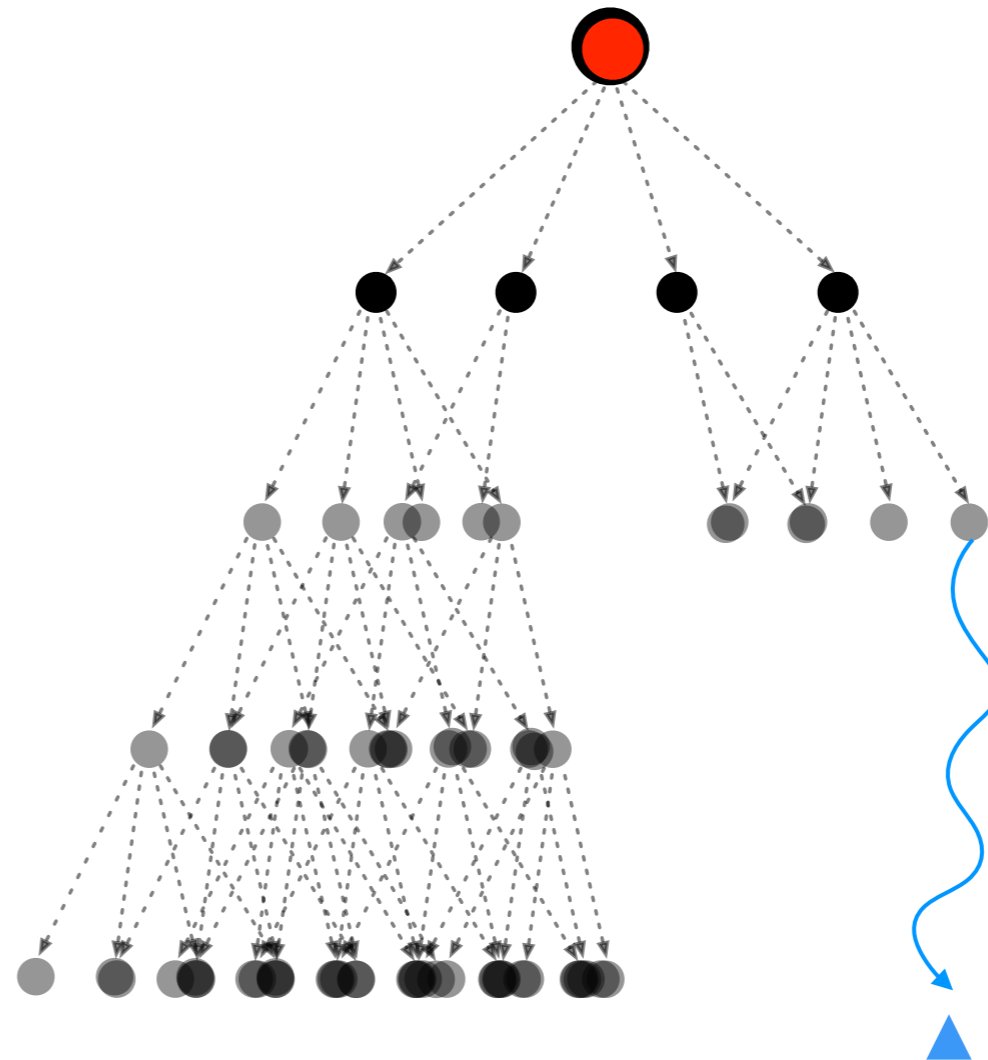
---

Step 1:  
Selection & Expansion

Step 2:  
Simulate a Rollout Policy

Step 3:  
Update the Tree

“make your way to a leaf node”  
balance exploration/exploitation



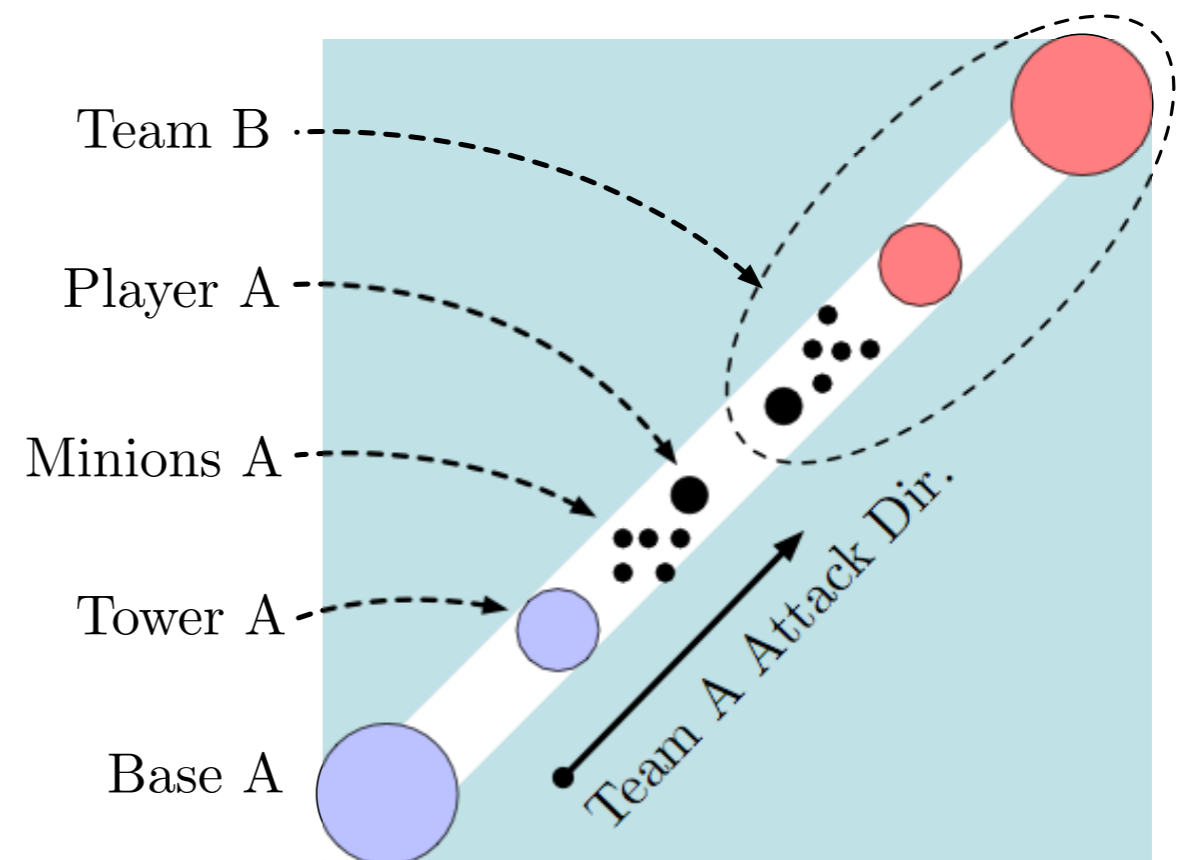
# Problem Setting

- Infinite horizon MDP.

$$V^\pi(s) = \mathbf{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_t(s_t)) \right],$$

$$V^*(s) = \sup_{\pi \in \Pi} V^\pi(s)$$

- Continuous state space.
- Finite action space.
- **Real-time** decisions required.
  - e.g., several decisions per second.



Example Application: Video Game



# Challenges Faced by MCTS

---

## Challenge 1: True Real-time Decision Making

- The “thinking time” required of MCTS is often too long for some practical applications.
- In our MOBA application, each run of MCTS takes 10-15 seconds in order for it to consistently win the game (~30x slower than real time).

## Challenge 2: Long/Infinite Horizons

- Simulation of rollout policy in **complex & long-horizon** environments can be computationally costly (several seconds to simulate a whole game).

## Challenge 3: Sub-optimality of leaf node evaluation (rollout)

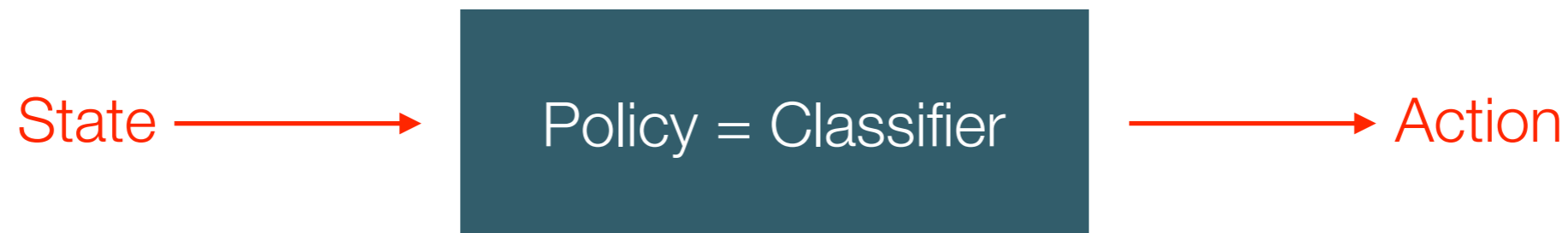
- Optimality in long/infinite horizon problems cannot be guaranteed unless leaf node evaluation is optimal.

**Most RL methods require hours/days of offline training. We can allow MCTS to do the same rather than taking the standard rolling approach.**

# Offline Training

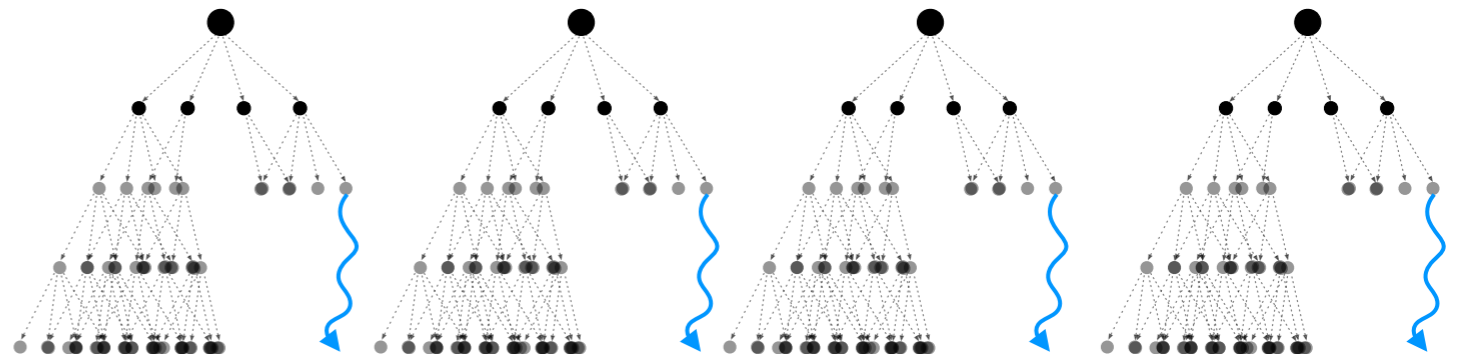
## Challenge 1: Real-time Decision Making

- The “thinking time” required of MCTS is often too long for some practical applications.
- In our MOBA application, each run of MCTS takes 10-15 seconds.



cf. Guo et. al., 2014

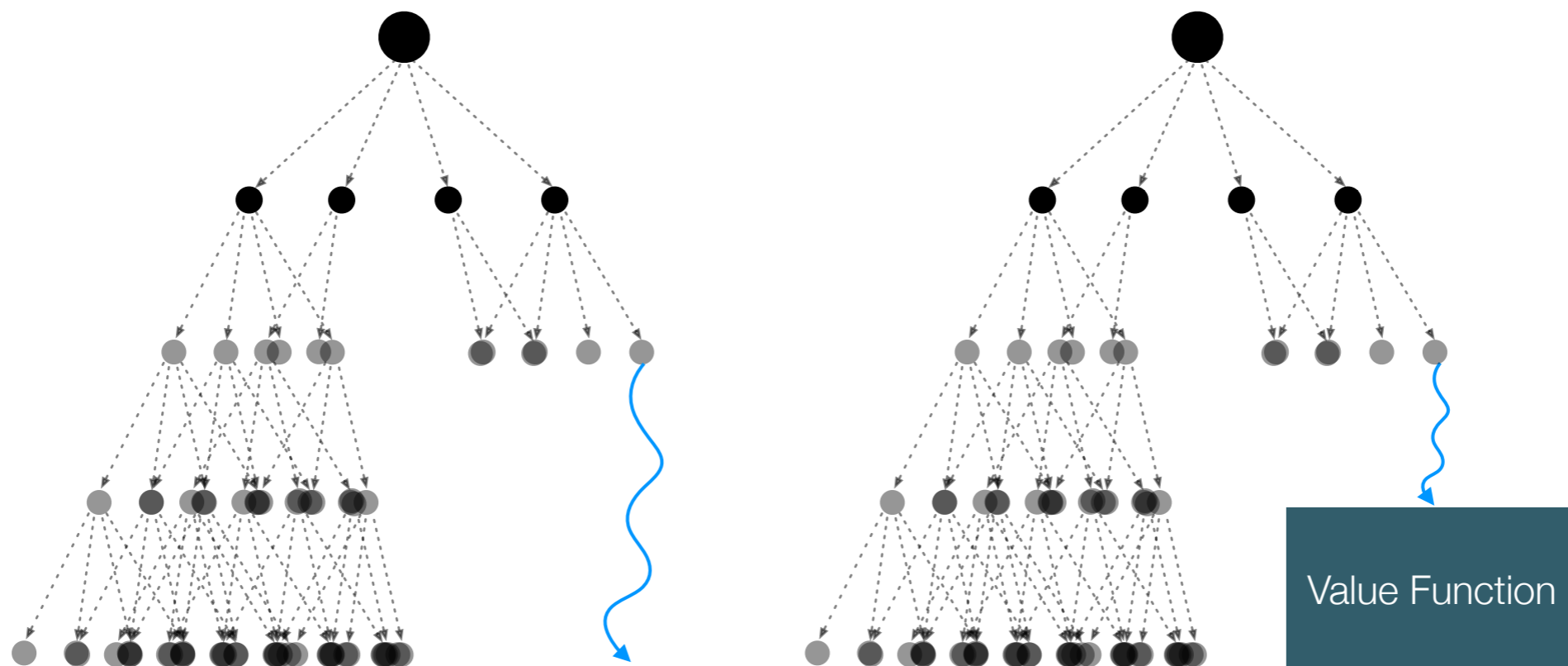
↑ Training data from offline MCTS



# Terminate Rollout with Value Function

## Challenge 2: Long/Infinite Horizons

- Simulation of rollout policy in **complex & long-horizon** environments can be computationally costly.



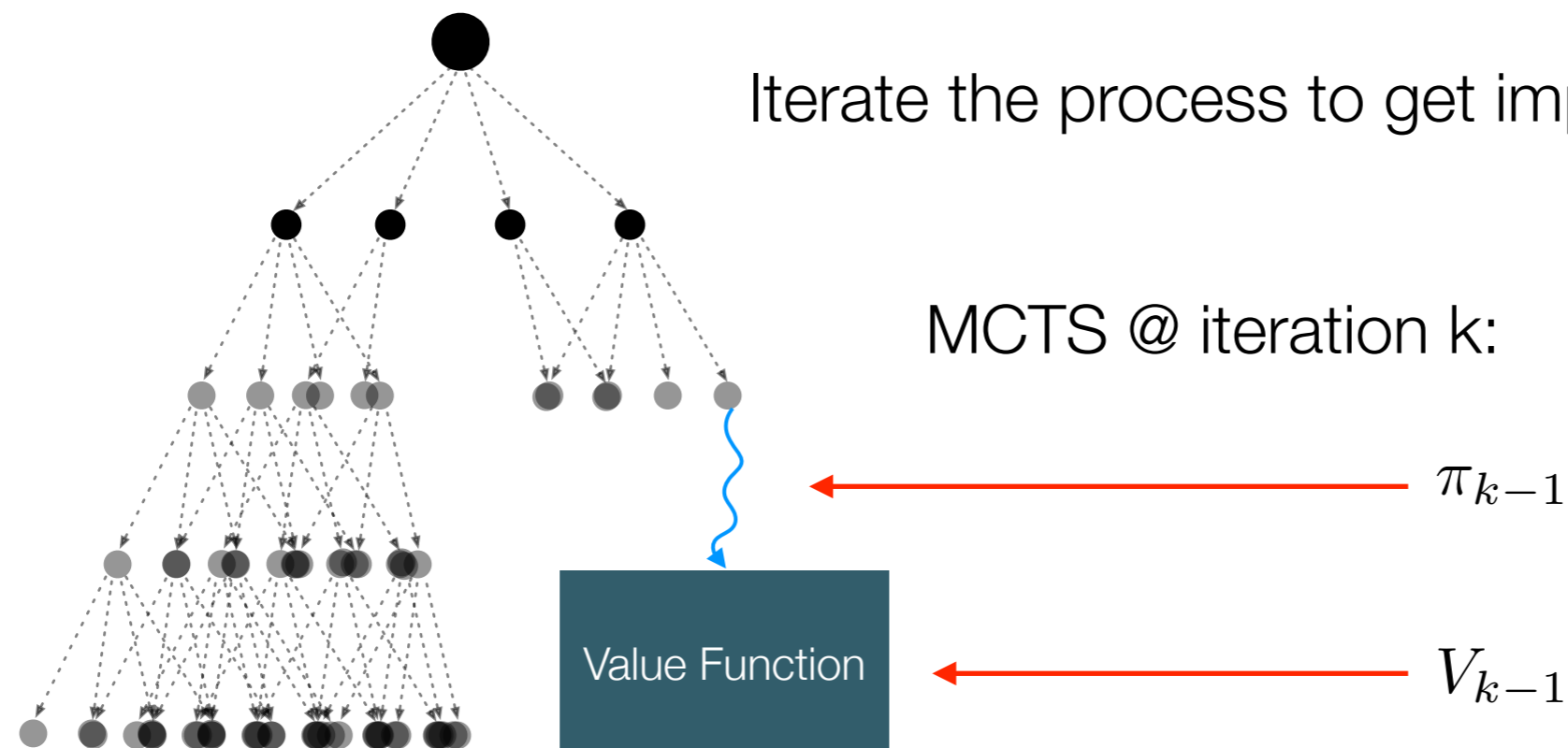
stop after some rollout horizon  $h$ ; intuition: error in value function gets discounted by  $\gamma^h$

# Iterate to Improve

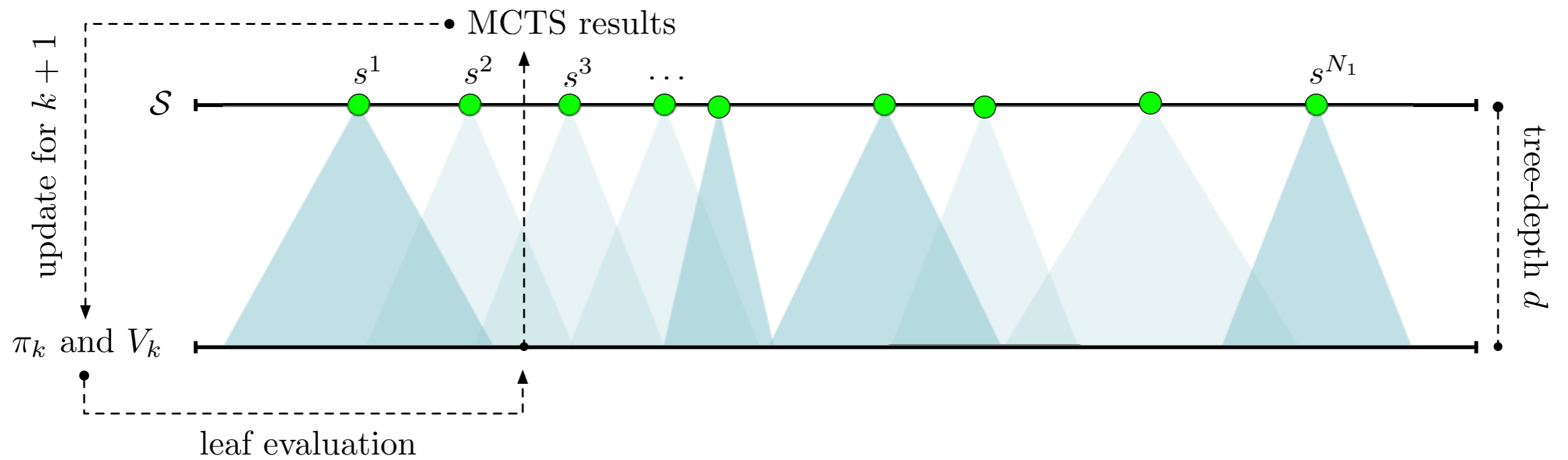
---

## Challenge 3: Sub-optimality of leaf node evaluation (rollout)

- Optimality in long/infinite horizon problems cannot be guaranteed unless leaf node evaluation is optimal



# Feedback-based (d-stage) Tree Search



# Feedback-based Tree Search

---

**Sample** states (i.i.d.) at which  
to estimate value



**Regression** to approximate  
value function



**Sample** states (i.i.d.) at which  
to run MCTS

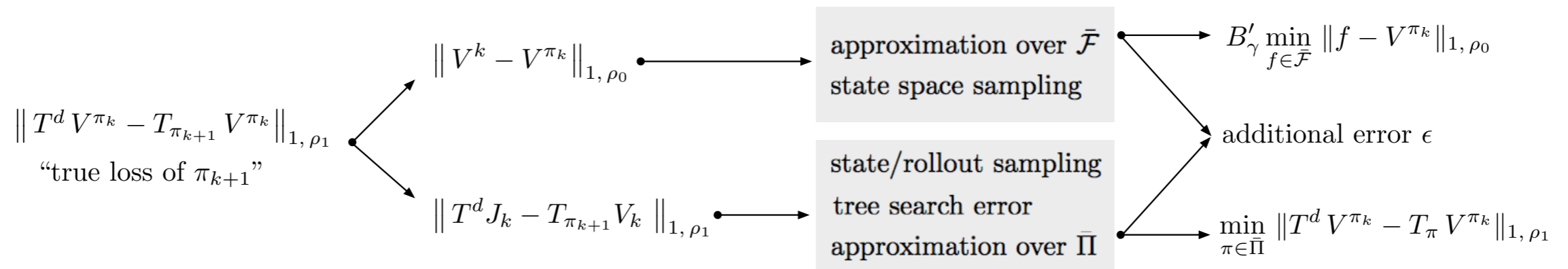


**Classification** to approximate  
policy function



1. Sample a set of  $N_0$  i.i.d. states  $\mathcal{S}_{0,k}$  from  $\rho_0$  and  $N_1$  i.i.d. states  $\mathcal{S}_{1,k}$  from  $\rho_1$ .
2. Compute a sample average  $\hat{Y}_k(s)$  of  $M_0$  independent roll-outs of  $\pi_k$  for each  $s \in \mathcal{S}_{0,k}$ . See Assumption 1.
3. Use **Regress** on the set  $\{\hat{Y}_k(s) : s \in \mathcal{S}_{0,k}\}$  to obtain a value function  $V_k \in \bar{\mathcal{F}}$ . See Assumption 1.
4. From each  $s \in \mathcal{S}_{1,k}$ , run MCTS with parameters  $M_0$ ,  $d$ , and evaluator **LeafEval**. Return estimated value of each  $s$ , denoted  $\hat{U}_k(s)$ . See Assumption 3.
5. For each  $s \in \mathcal{S}_{1,k}$  and  $a \in \mathcal{A}$ , create estimate  $\hat{Q}_k(s, a) \approx (T_a V_k)(s)$  by averaging  $L_1$  transitions from  $p(\cdot | s, a)$ . See Assumption 4.
6. Use **Classify** to solve a cost-sensitive classification problem and obtain  $\pi_{k+1} \in \bar{\Pi}$ . Costs are defined using  $\{\hat{U}_k(s) : s \in \mathcal{S}_k\}$  and  $\{\hat{Q}_k(s, \pi_{k+1}(s)) : s \in \mathcal{S}_{1,k}\}$ . See Assumption 4. Increment  $k$  and return to Step 1.

# Main Result



**Theorem.** For sufficiently large sample sizes (depending on  $\delta$ ), it holds that

$$\|V^* - V^{\pi_K}\|_{1, \nu} \leq B_\gamma [B'_\gamma \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi})] + \gamma^{Kd} \|V^* - V^{\pi_0}\|_\infty + \epsilon,$$

with probability at least  $1 - \delta$ .

  
 approximation error

# Related to Multistage PI

Under some small modifications to our classification step, FBTS is **similar in spirit to a multistage version of policy iteration**, which can be shown to converge (Bertsekas & Tsitsiklis, 1996).

## Beyond the One-Step Greedy Approach in Reinforcement Learning

Yonathan Efroni<sup>1</sup> Gal Dalal<sup>1</sup> Bruno Scherrer<sup>2</sup> Shie Mannor<sup>1</sup>

### Abstract

The famous Policy Iteration algorithm balances between policy improvement and policy evaluation. Implementations of this algorithm using several variants of the latter evaluation step, such as  $n$ -step and trace-based returns, have been analyzed in previous works. However, the multiple-step lookahead policy improvement step, despite the recent increase in empirical evidence of its strength, has to our knowledge not been fully analyzed yet. In this work, we provide the first such analysis. Namely, we analyze several variants of multiple-step policy improvement and propose new algorithms using these definitions.

### Algorithm 1 $h$ -PI

```
Initialize:  $h \in \mathbb{N} \setminus \{0\}, v \in \mathbb{R}^{|S|}$   
while the value  $v$  changes do  
   $\pi \leftarrow \arg \max_{\pi_0, \pi_1, \dots, \pi_{h-1}} \max_{\pi_h} \mathbb{E}_{\pi_0, \dots, \pi_{h-1}} \left[ \sum_{t=0}^{h-1} \gamma^t r_t + \gamma^h v_h \right]$   
   $v \leftarrow \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$   
end while  
Return  $\pi, v$ 
```

also @ ICML 2018  
tomorrow 4:20-4:40 @ A1



# Test Case: Video Game (MOBA Game)

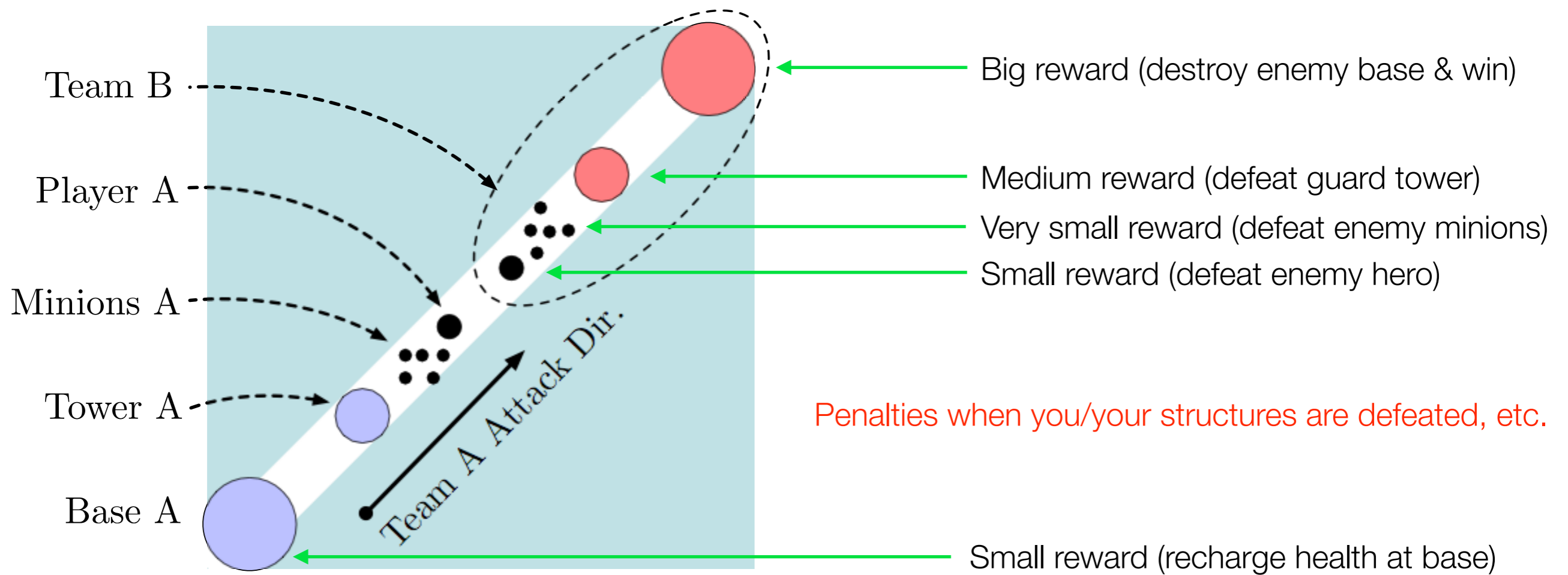
- King of Glory, a MOBA game by Tencent.
- Goal is to defeat the enemy crystal, which is guarded by a tower.
- State space is **41-dimensional**.
- There are **22 actions**, including move (discretized directions), attack, special attacks, and heal.
- Several actions per second.
- Each hero is assisted by computer controlled “minions.”
- **One call to MCTS takes 10-15 seconds.**



$$V^\pi(s) = \mathbf{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_t(s_t)) \right],$$

$$V^*(s) = \sup_{\pi \in \Pi} V^\pi(s)$$

# Reward Specifications



# Competing Batch Algorithms

---

## Algorithm 1: FBTS with No Rollouts

- Directly using a value function at the leaf nodes of the tree search.

## Algorithm 2: Approximate Value Iteration (Munos & Szepesvári, 2008)

- Or “fitted value iteration.” No policy evaluations.

## Algorithm 3: Direct Policy Iteration (Lazaric et al., 2016)

- No tree search; no value function.

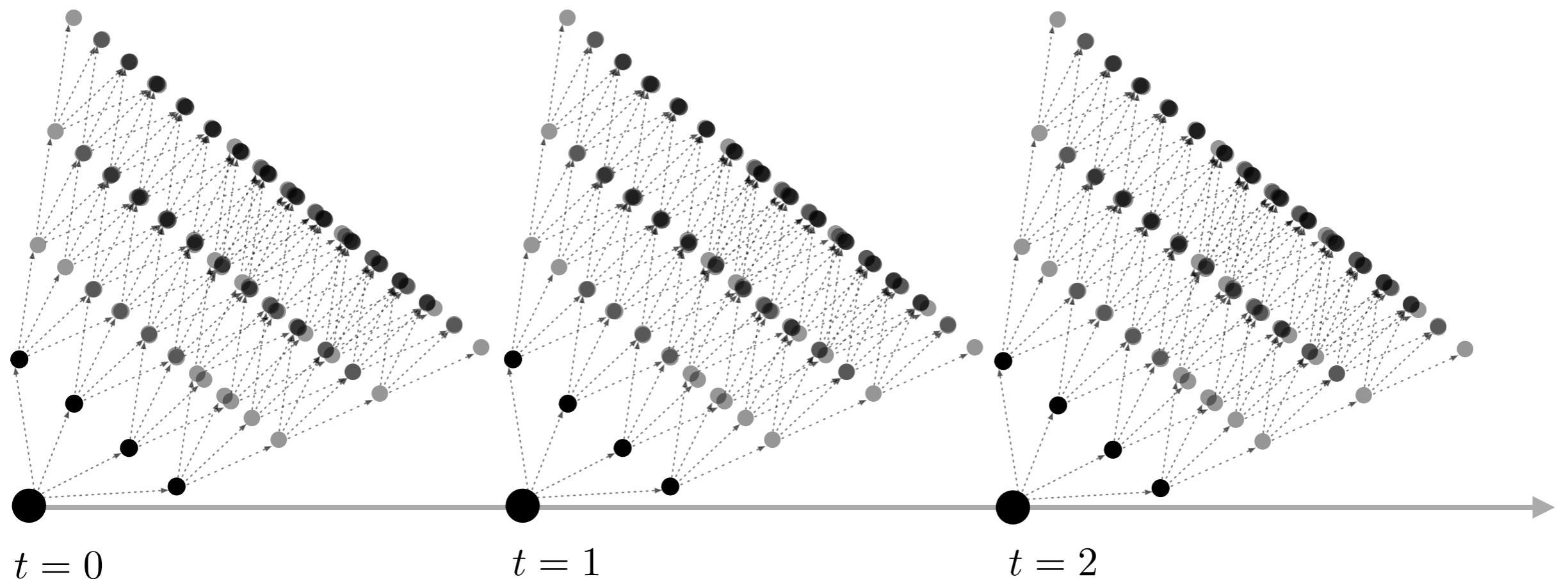
## How does FBTS differ? Primarily:

- “multi-step” nature of MCTS,
- partial rollout with value function evaluation.

# Practical Implementation

---

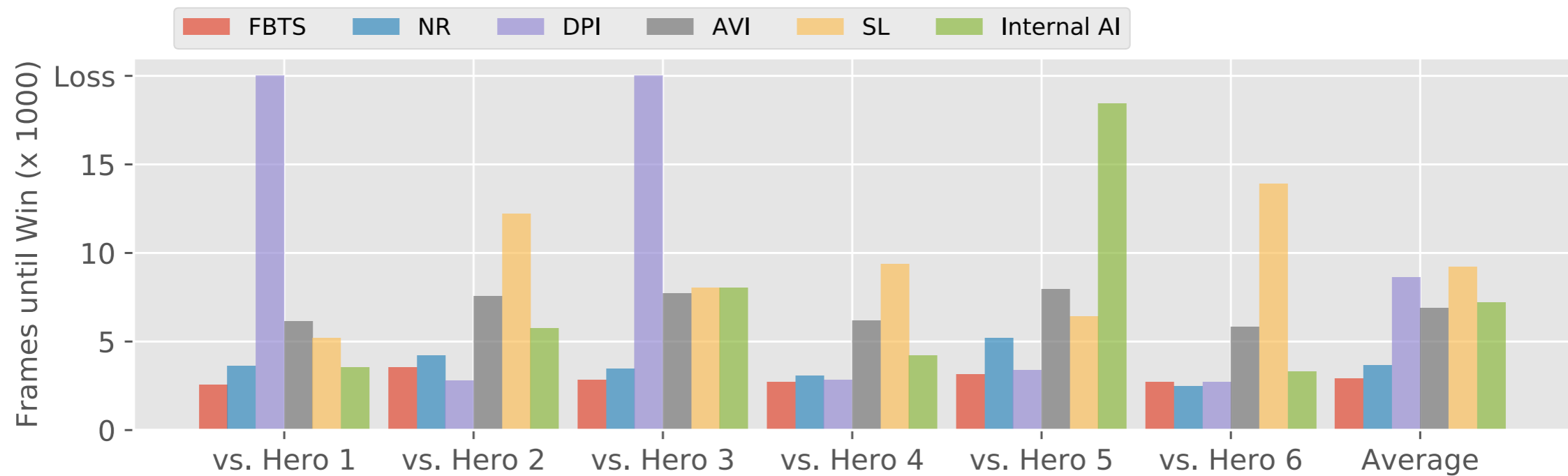
**Main Challenge:** Sampling batch of i.i.d. states is difficult in the game simulator. Instead, we implemented a version where we played full games using MCTS while injecting some noise into the actual implemented actions.



+ some exploration

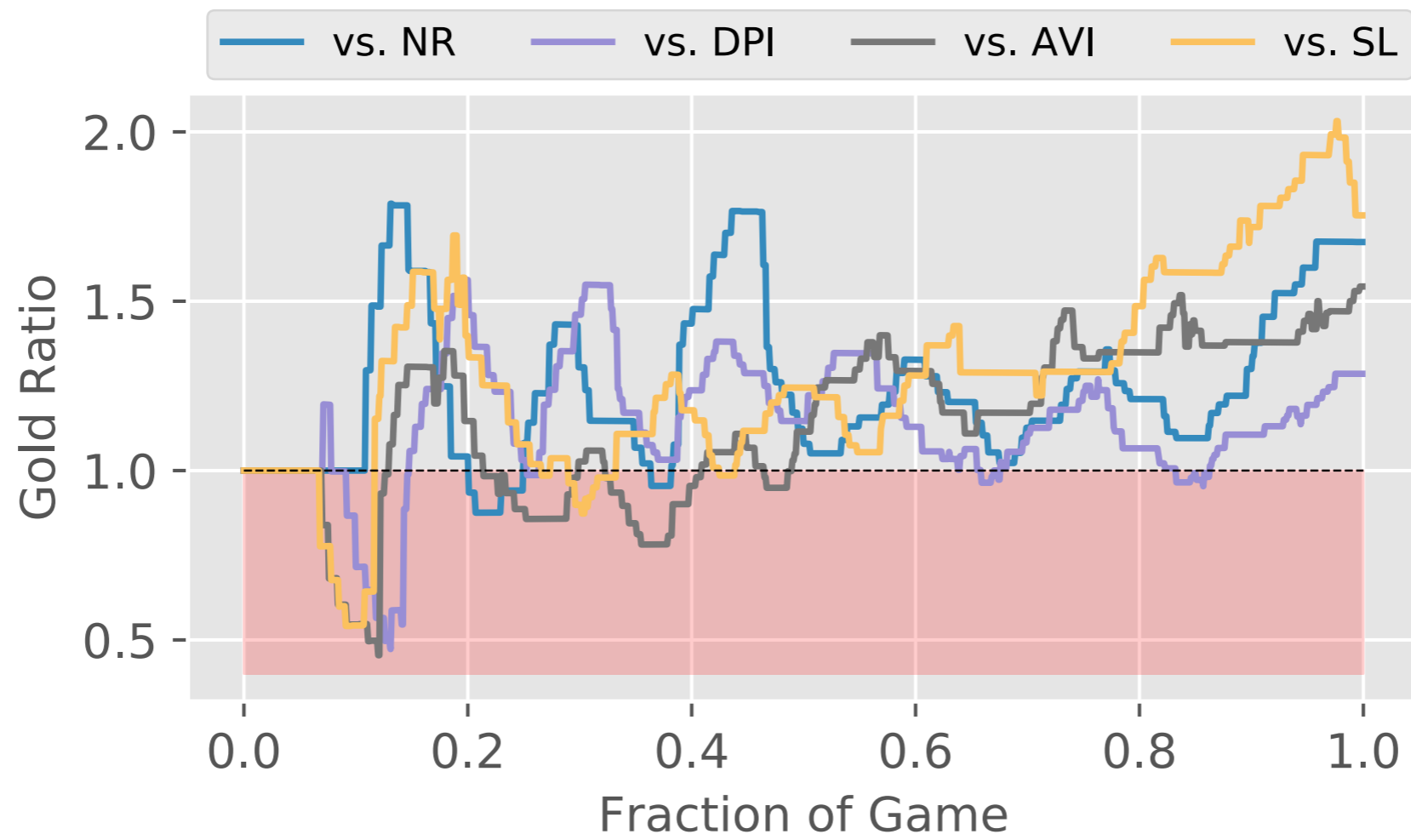
# Common Opponents (Internal AI)

For all algorithms, initial policy is almost completely **random**: move (w.p. 0.5), directional attack (w.p. 0.2), special skill (w.p. 0.3)



# Head-to-Head vs. Competing Algorithms

---





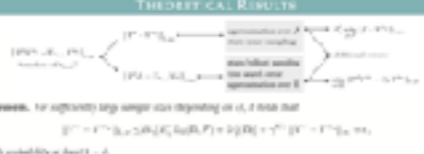


# Thank you!

## Visit poster #162 tonight for more details!

### FEEDBACK-BASED TREE SEARCH FOR REINFORCEMENT LEARNING

Daniel Jiang, Ruosong Zhuo and He Li



<p><b>ABSTRACT</b></p> <p>We propose a reinforcement learning (RL) technique that iteratively applies MCTS on batches of small, finite-horizon versions of the original reinforcement learning decision process. We provide the first sample complexity bounds for a tree-walk-based RL algorithm. In addition, we show that a deep neural network implementation of this technique can create a competitive player on the popular multi-player video table game Go (GoFA game like GoFA).</p>	<p><b>FEEDBACK LOOP</b></p> 	<p><b>THEORETICAL RESULTS</b></p>  <p><b>Theorem.</b> For arbitrary step size <math>\eta</math> depending on <math>\epsilon</math>, it holds that</p> $E[V^* - V^T] \leq \epsilon + O(\eta \log(1/\epsilon)) + O(\eta^2)$ <p>with probability at least <math>1 - \delta</math>.</p>
<p><b>ABSTRACT</b></p> <p><b>GoFA Phase.</b> Given a policy <math>\pi</math>, we independently bootstrap using self-play progress and <math>\pi</math> values sampled from a distribution <math>\mu</math>.</p> <p><b>GoFA Phase.</b> On every iteration <math>k</math>, we sample a batch <math>\mathcal{B}_k</math> of <math>N</math> states from a distribution <math>\mu</math>. For each state <math>s</math> in <math>\mathcal{B}_k</math>, we evaluate the current policy <math>\pi</math>, and store the value <math>V(s)</math>. We then use a neural network to estimate the value of each state <math>s</math>.</p> <p><b>GoFA Phase.</b> We then use the neural network to estimate the value of each state <math>s</math>.</p>	<p><b>TEST CASE: GOFA GAME</b></p>  <ul style="list-style-type: none"> <li>• Key of GoFA: a GoFA game by Tencent, similar to League of Legends or Dota.</li> <li>• Goal is to defeat the enemy crystal, which is guarded by towers.</li> <li>• There are 10 actions including move, fire, attack, and special attacks.</li> </ul>	<p><b>EMPIRICAL RESULTS</b></p> 
<p><b>REFERENCES</b></p> <p>[1] M. Browne, E. Herberich, M. Johansen, and D. Schaeffer. Analyzing GoFA: a new game for Monte Carlo tree search. <i>Journal of Machine Learning Research</i>, 17:1-13, 2016.</p> <p>[2] M. Browne and G. Schaeffer. Reinforcement learning in GoFA: a new game for Monte Carlo tree search. <i>Journal of Machine Learning Research</i>, 17:1-13, 2016.</p>	<p><b>ASSUMPTIONS</b></p> <p><b>Assumption 1 (Episodic Substochastic).</b></p> <p><b>Assumption 2 (Self-play Substochastic).</b></p> <p><b>Assumption 3 (GoFA Substochastic).</b></p> <p><b>Assumption 4 (GoFA Substochastic).</b></p>	<p><b>CONCLUSION &amp; FUTURE WORK</b></p> <p>We provide a sample complexity study on the feedback-based tree search, an RL algorithm based on iteratively solving finite-horizon subproblems using MCTS. The implementation of the algorithm in the GoFA game provided an encouraging result against several state-of-the-art algorithms. In future work, we will continue to improve the algorithm by incorporating more advanced RL techniques, and we will study the generalization of the algorithm to other GoFA-like games.</p>