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*

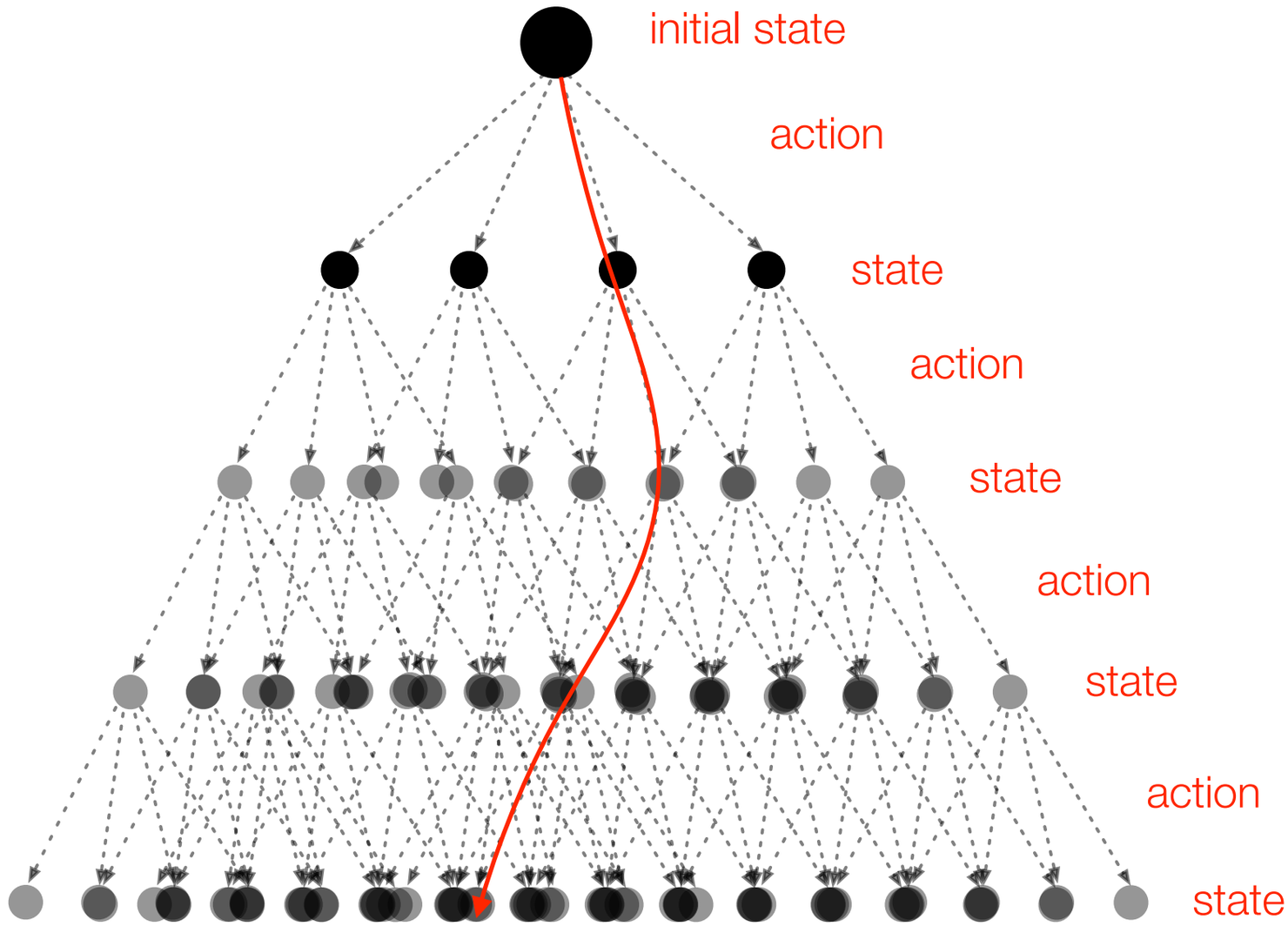
July 11, 2018
ICML 2018, Stockholm, Sweden

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

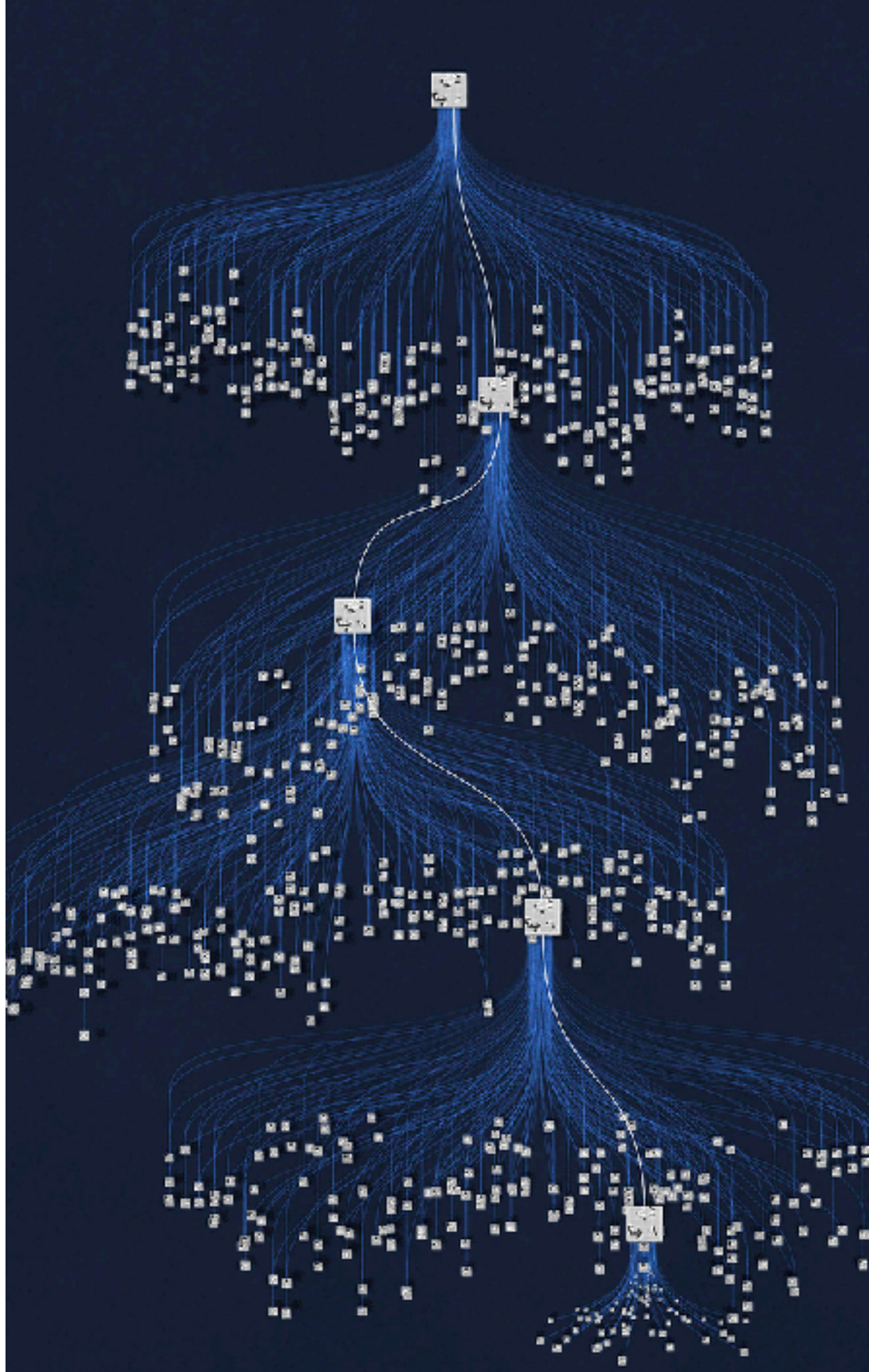
"state, action, new state… up to a finite horizon"



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.

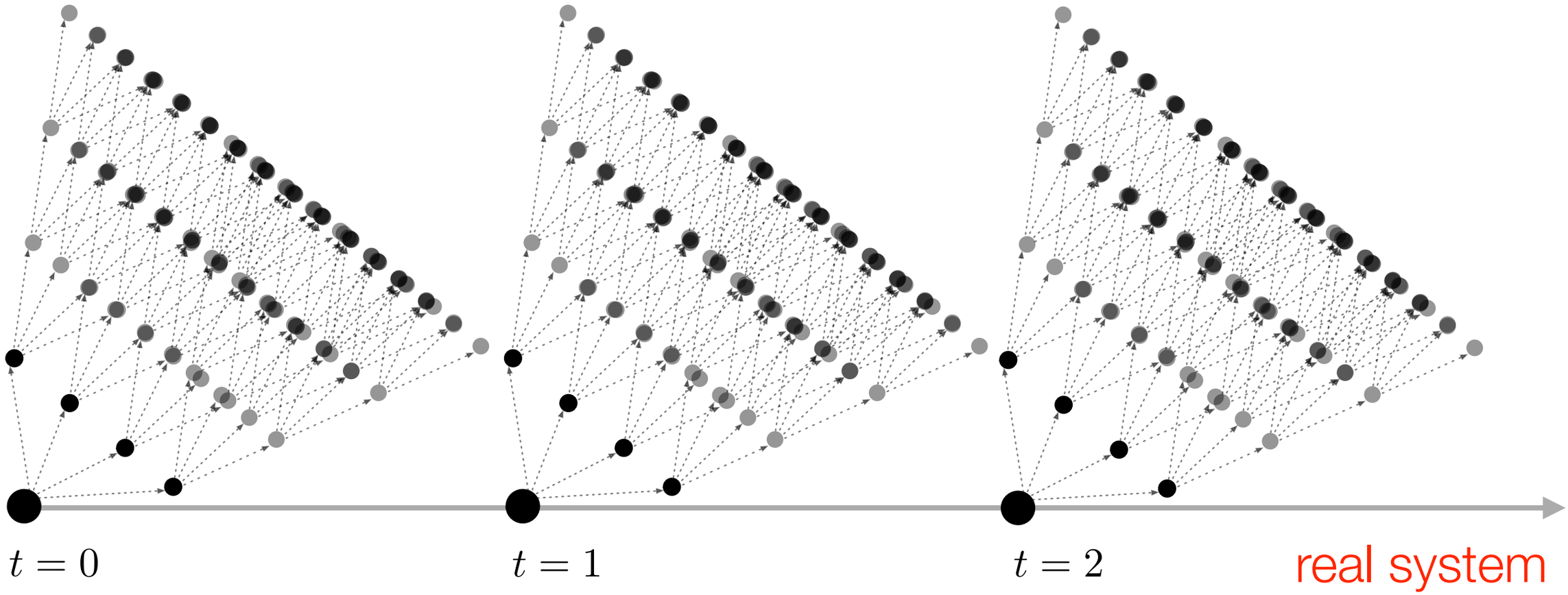


Source: *https://www.alphagomovie.com/gallery*

Decision Trees *in Real Time*

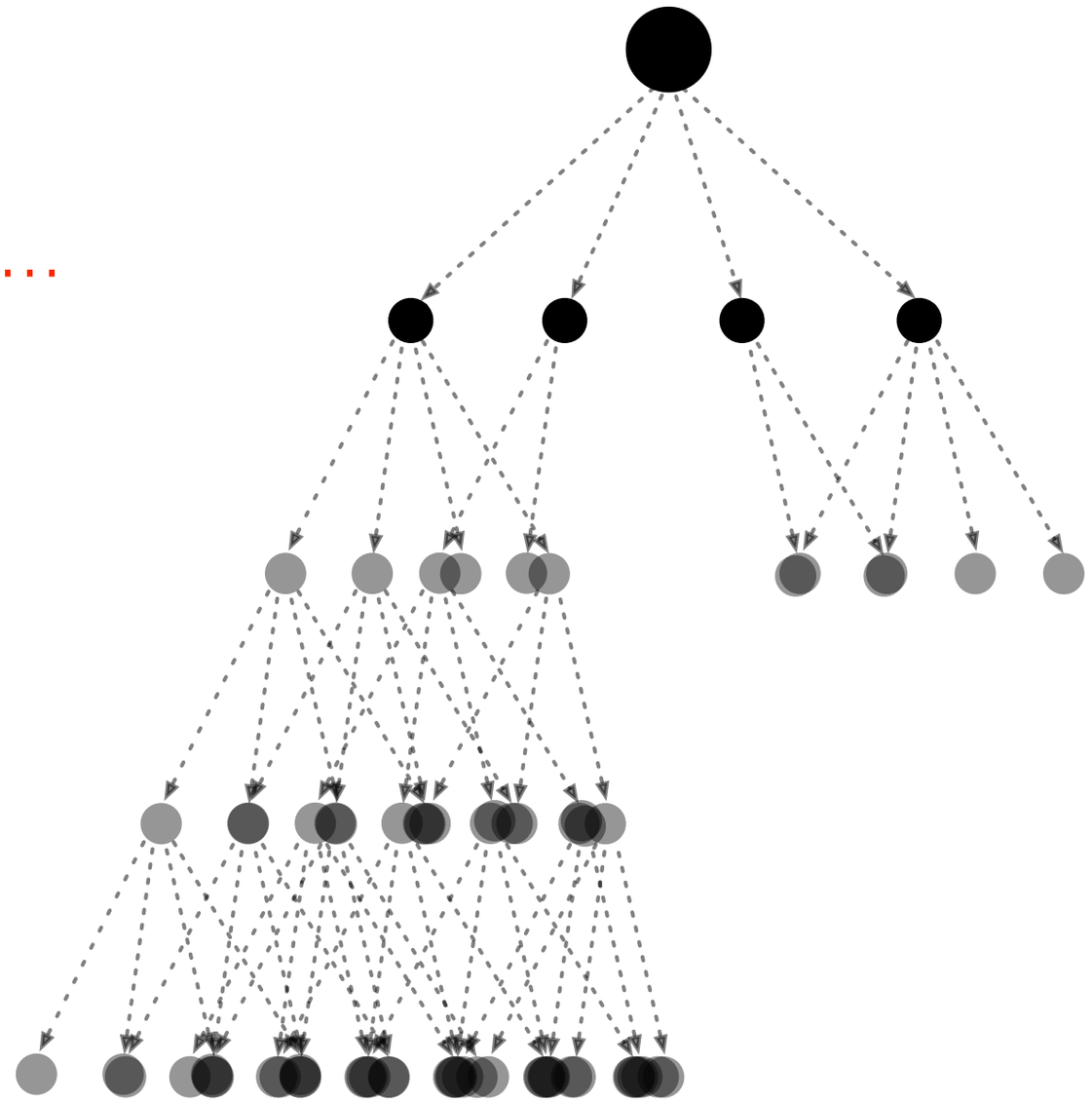
"looking ahead from the current node in a rolling manner"

"simulated futures"



Asymmetric Decision Trees

Under a good default policy…



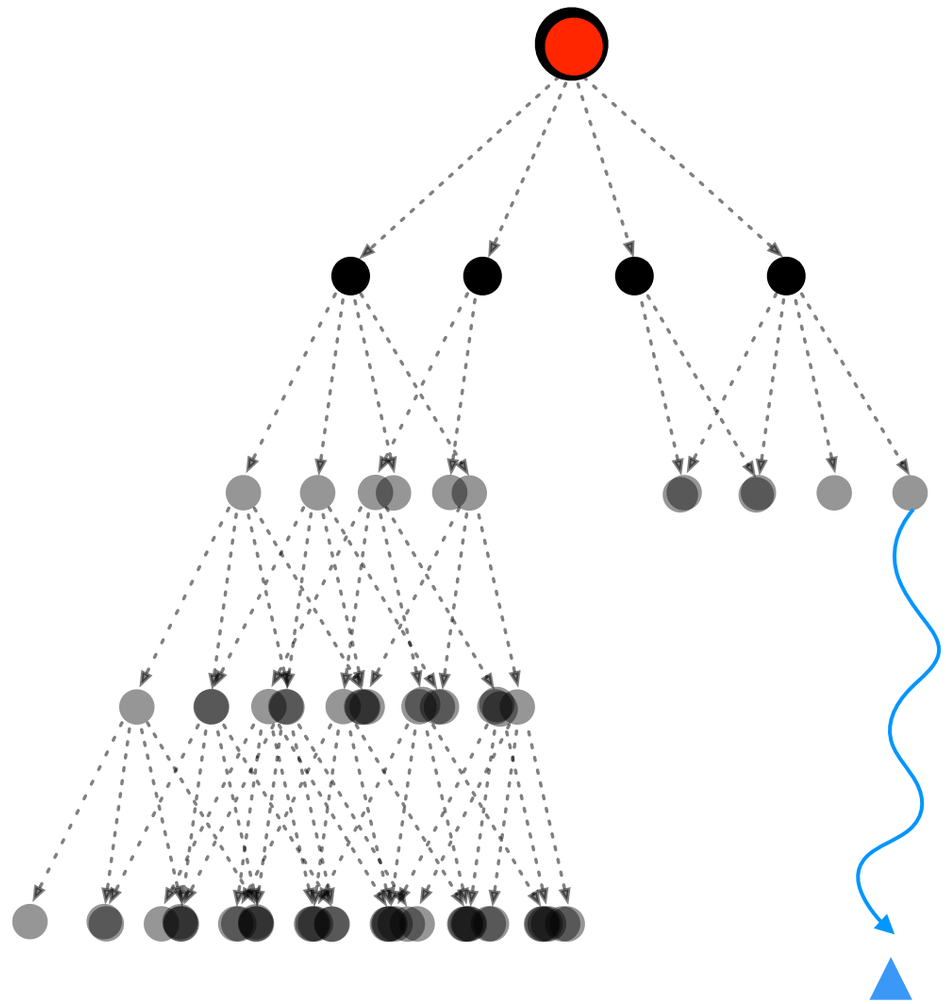
Main Steps of MCTS

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



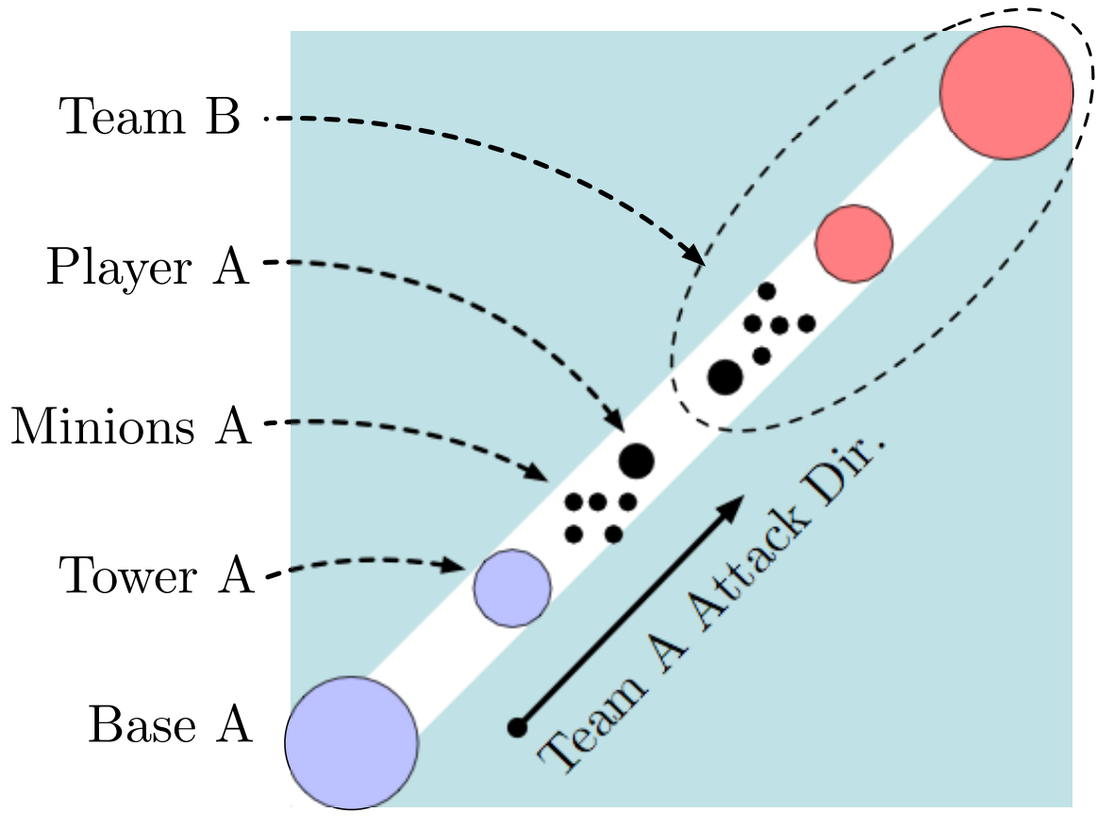
Problem Setting

- 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)

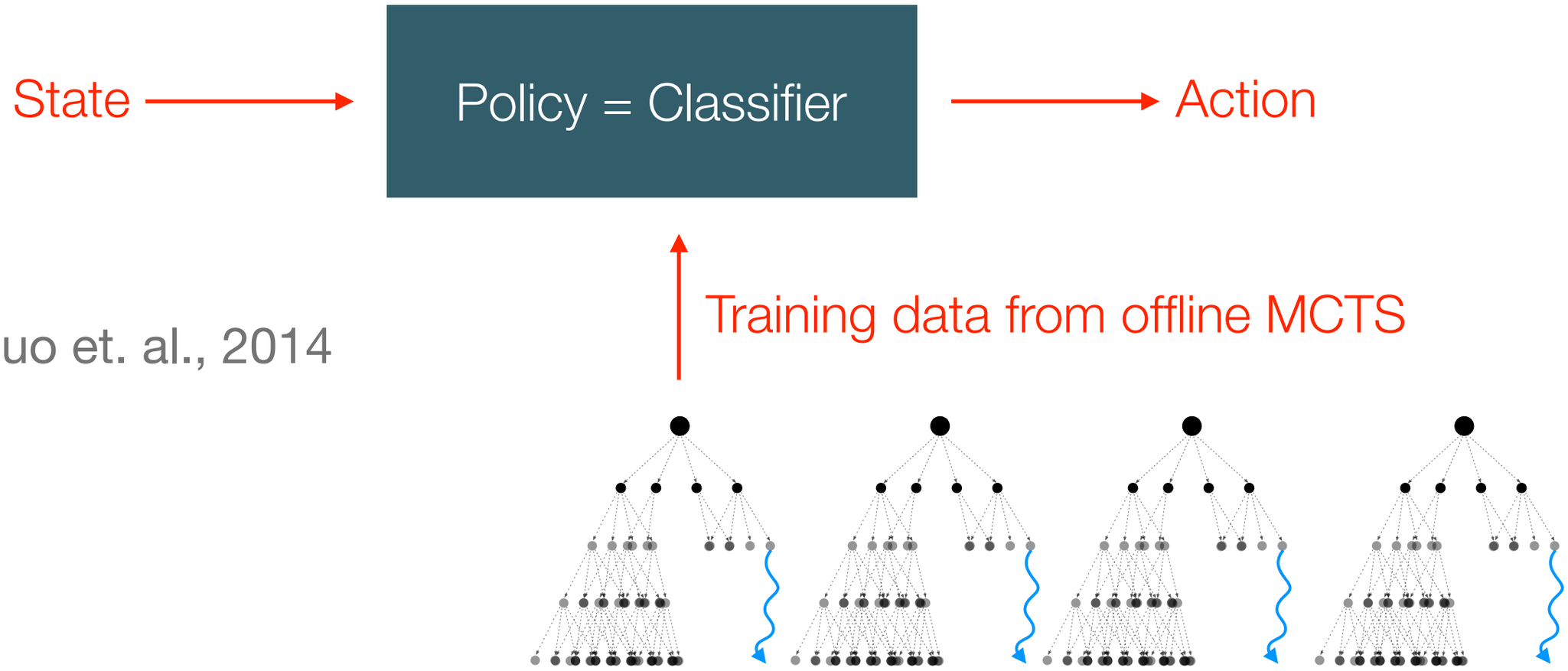
- Optimality in long/infinite horizon problems cannot be guaranteed unless leaf node evaluation is optimal.

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.

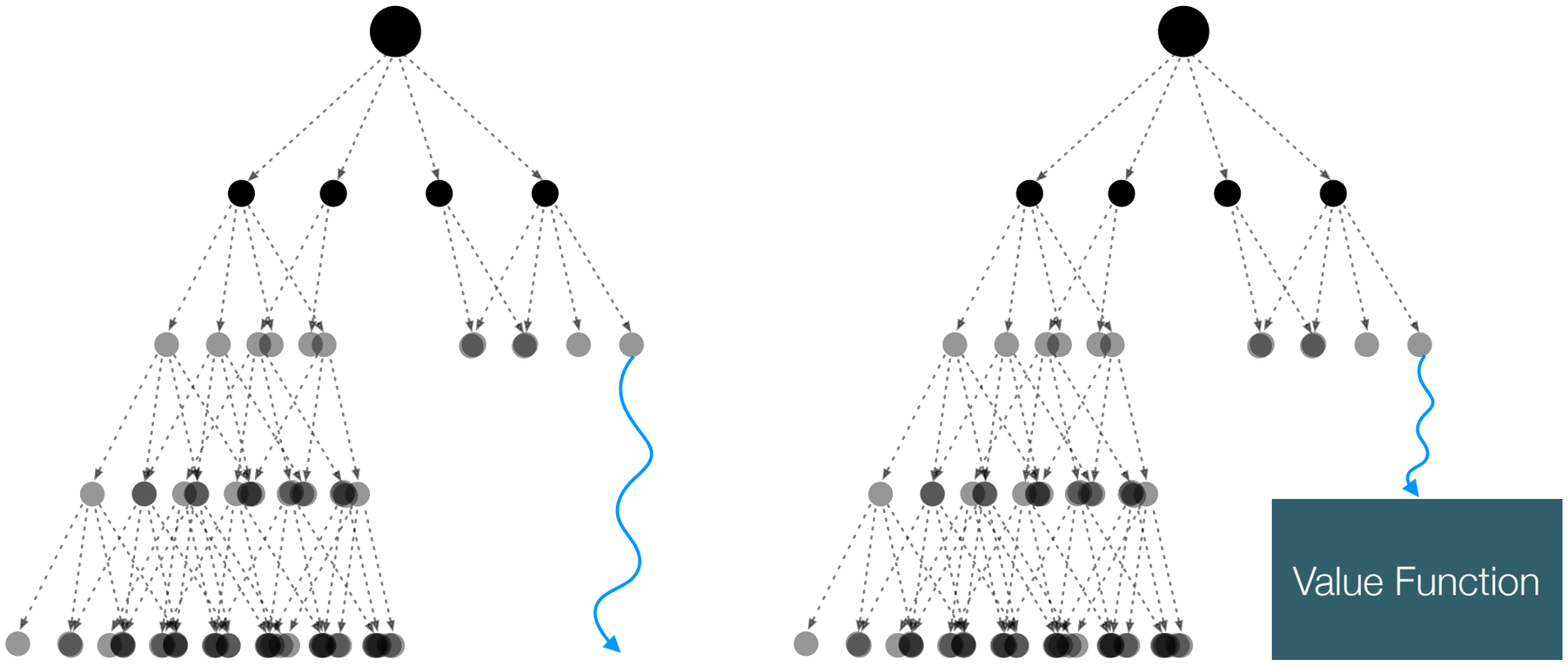


cf. Guo et. al., 2014

Terminate Rollout with Value Function

Challenge 2: Long/Infinite Horizons

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



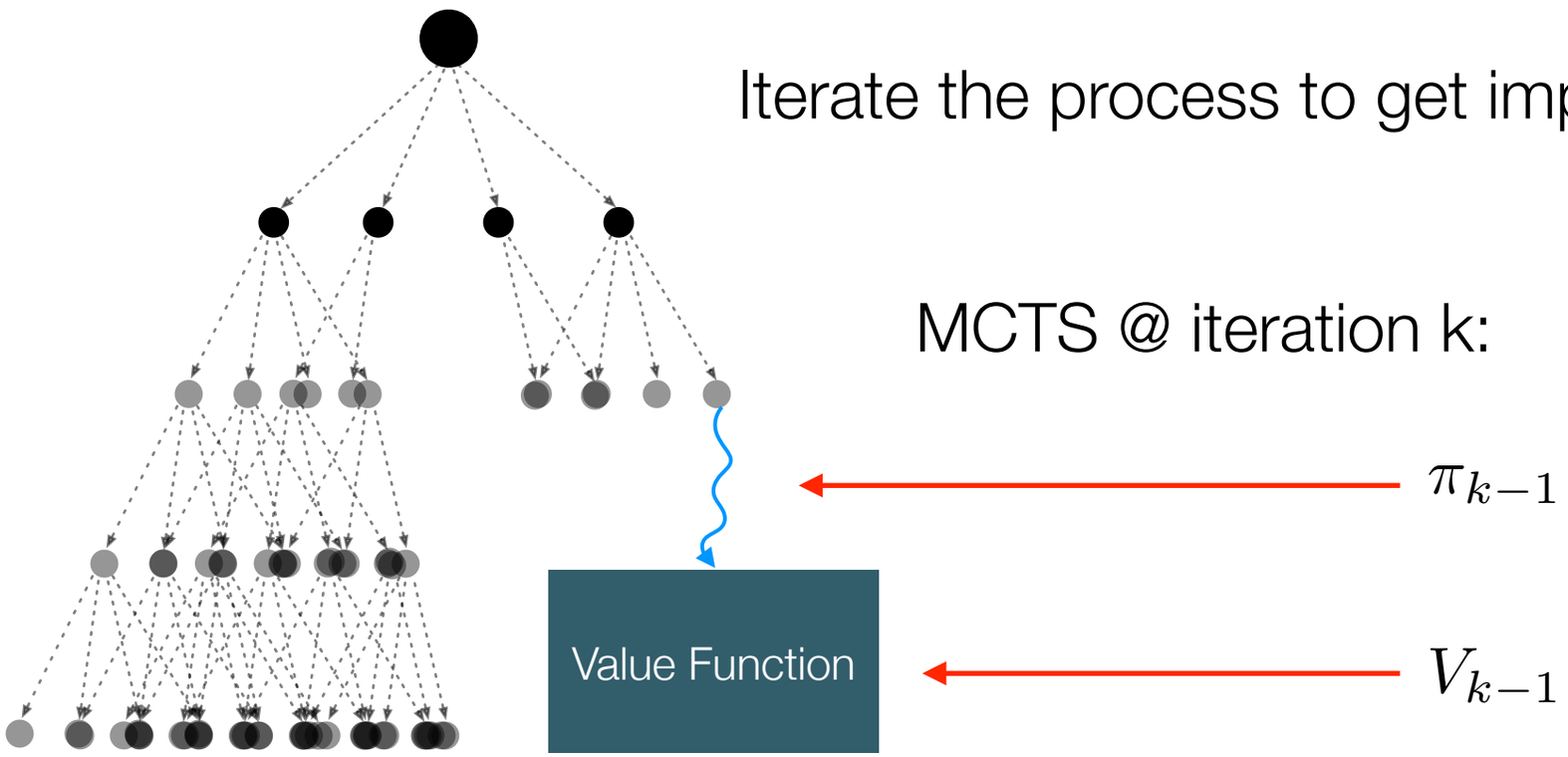
stop after some rollout horizon h; intuition: error in value function gets discounted by γ^h

Iterate to Improve

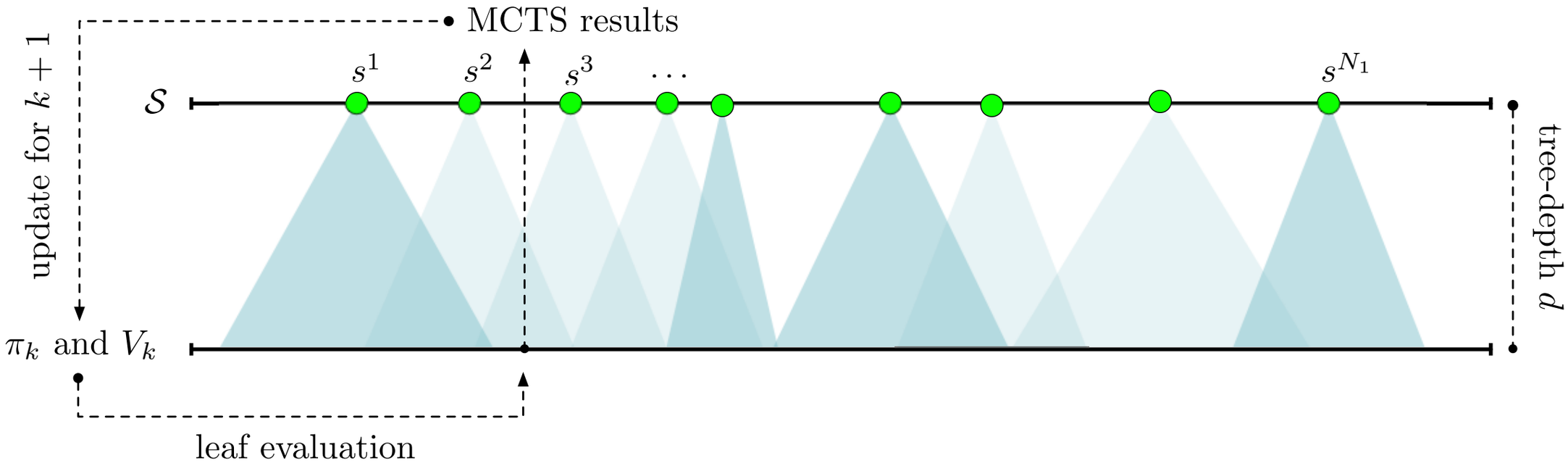
Challenge 3: Sub-optimality of leaf node evaluation (rollout)

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

Iterate the process to get improving policies.



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 ρ_0 and N_1 i.i.d. states $\mathcal{S}_{1,k}$ from ρ_1.

2. Compute a sample average $\hat{Y}_k(s)$ of M_0 independent roll-outs of π_k for each $s \in \mathcal{S}_{0,k}$. See Assumption 1.

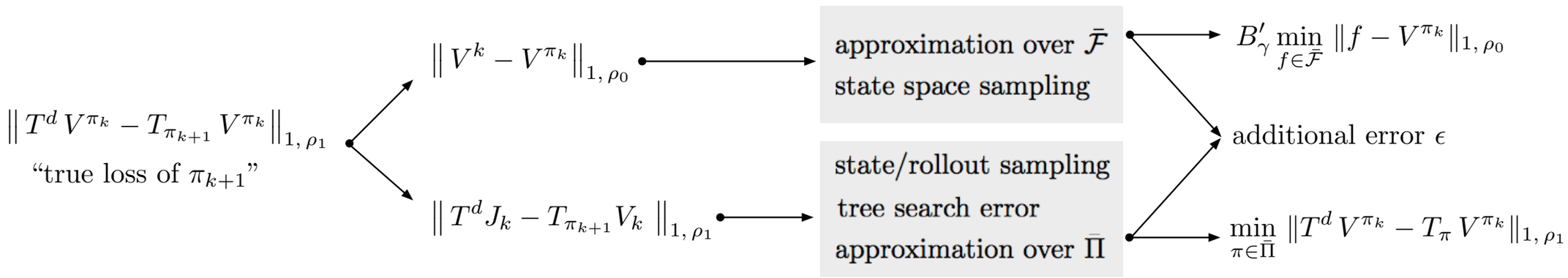
3. Use `Regress` on the set $\{\hat{Y}_k(s) : s \in \mathcal{S}_{0,k}\}$ to obtain a value function $V_k \in \bar{\mathcal{F}}$. See Assumption 1.

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 $p(\cdot \mid s, a)$. 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 δ), it holds that

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

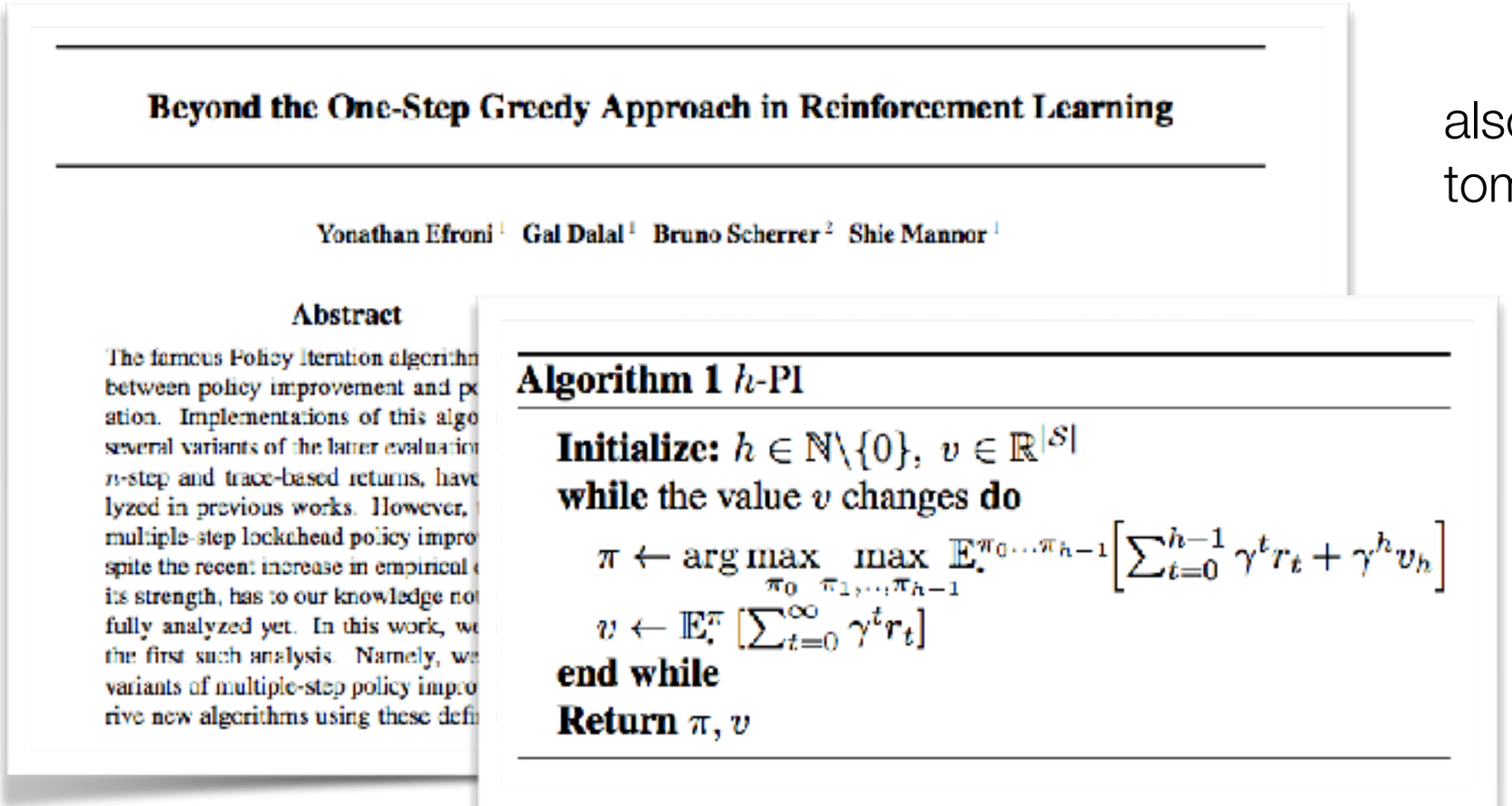
with probability at least $1 - \delta$.

approximation error

cf. Kaufmann & Koolen, 2017; Lazaric et al., 2016; Munos & Szepesvári, 2008.

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 [1] Gal Dalal [1] Bruno Scherrer [2] Shie Mannor [1]

Abstract

The famous Policy Iteration algorithm
between policy improvement and po
ation. Implementations of this algo
several variants of the latter evaluation
n-step and trace-based returns, have
lyzed in previous works. However,
multiple-step lookahead policy impro
spite the recent increase in empirical
its strength, has to our knowledge not
fully analyzed yet. In this work, we
the first such analysis. Namely, we
variants of multiple-step policy impro
rive new algorithms using these defi

Algorithm 1 h-PI

Initialize: $h \in \mathbb{N}\backslash\{0\},\ v \in \mathbb{R}^{|S|}$

while the value v changes **do**

$$\pi \leftarrow \arg\max_{\pi_0} \max_{\pi_1,\ldots,\pi_{h-1}} \mathbb{E}_{\cdot}^{\pi_0\ldots\pi_{h-1}}\left[\sum_{t=0}^{h-1}\gamma^t r_t + \gamma^h v_h\right]$$

$$v \leftarrow \mathbb{E}_{\cdot}^{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$$

end while

Return π, v

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

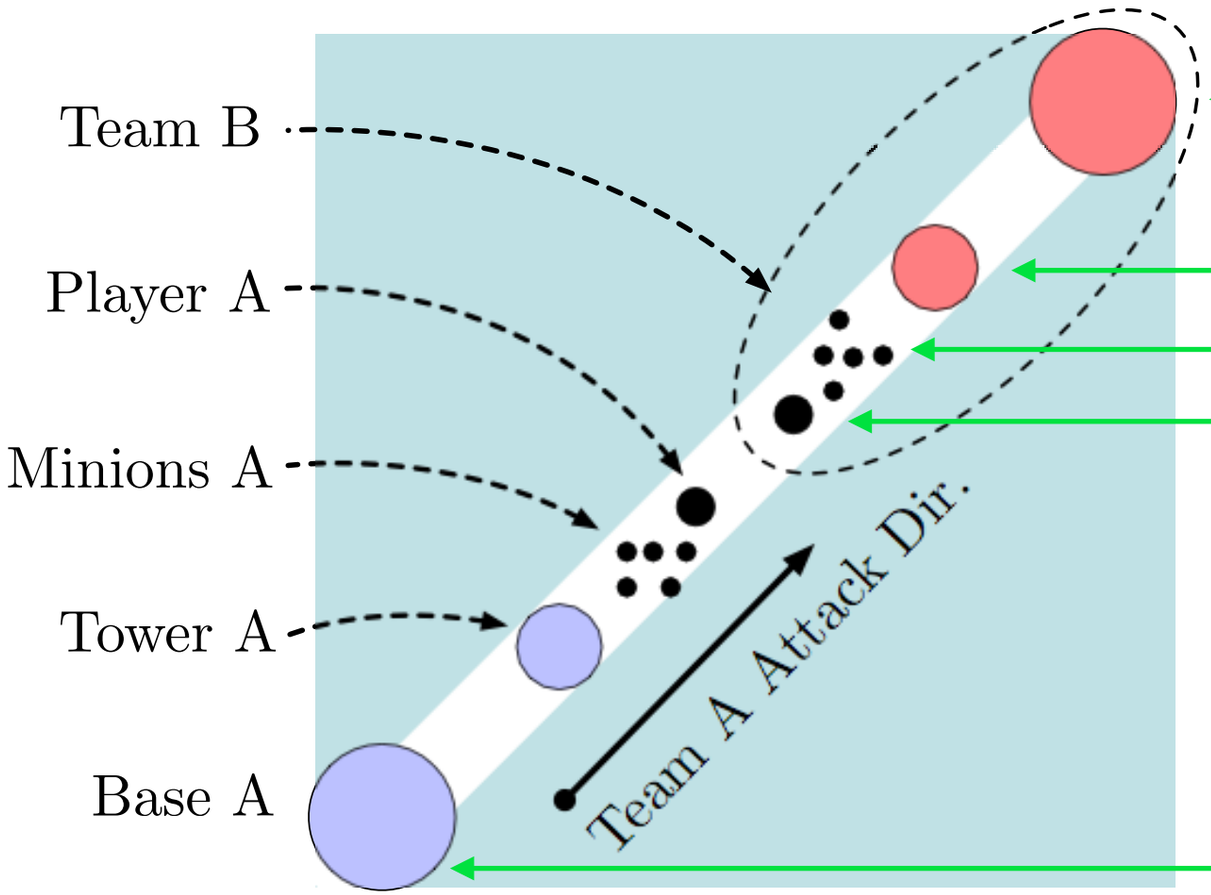
Test Case: Video Game (MOBA Game)

- 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



Big reward (destroy enemy base & win)

Medium reward (defeat guard tower)

Very small reward (defeat enemy minions)

Small reward (defeat enemy hero)

Penalties when you/your structures are defeated, etc.

Small reward (recharge health at base)

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)

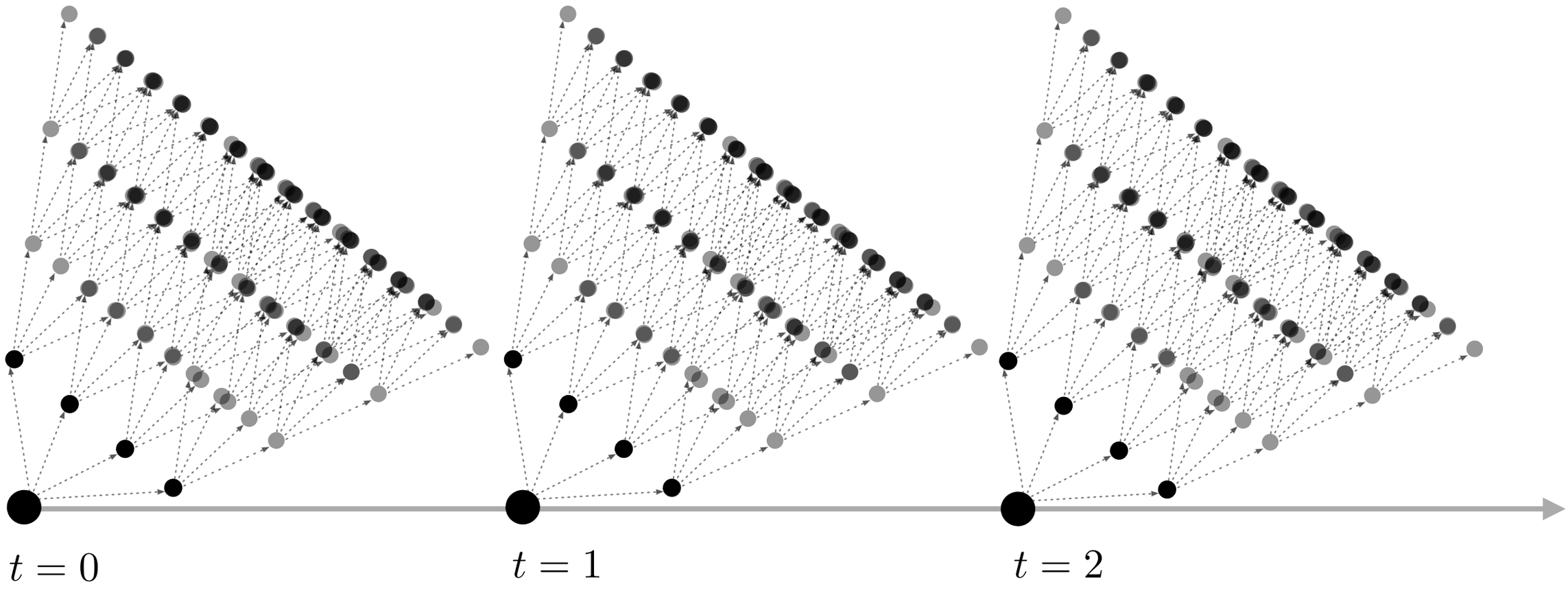
- No tree search; no value function.

How does FBTS differ? Primarily:

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

Practical Implementation

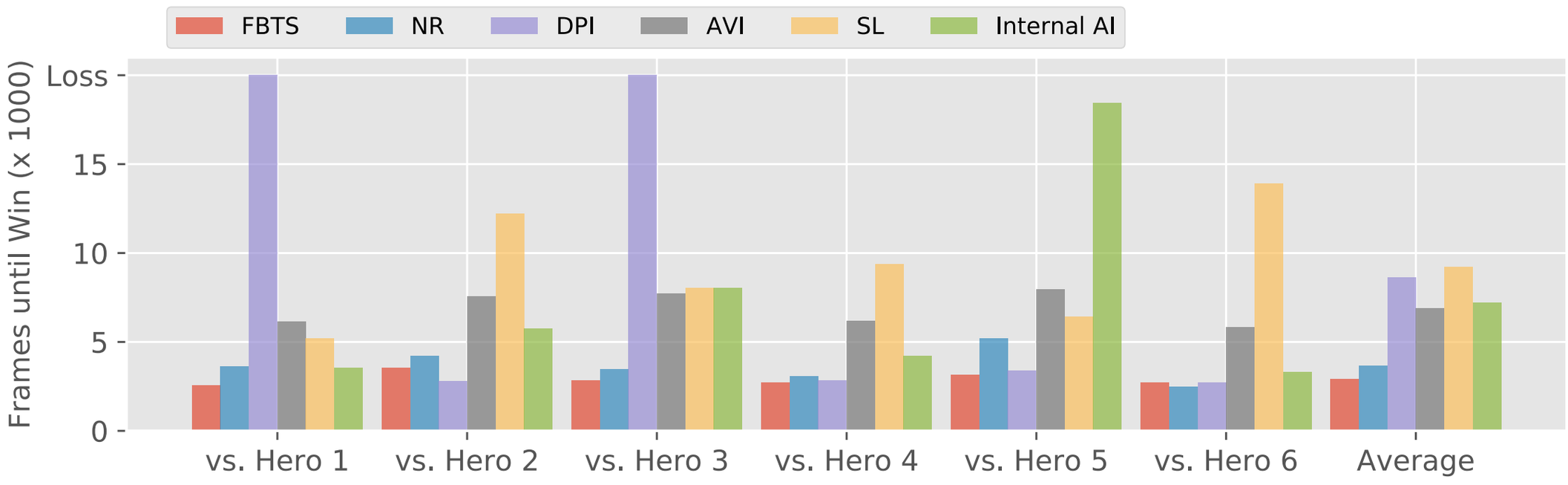
Main Challenge: Sampling batch of i.i.d. states is difficult in the game simulator. Instead, we implemented a version where we played full games using MCTS while injecting some noise into the actual implemented actions.



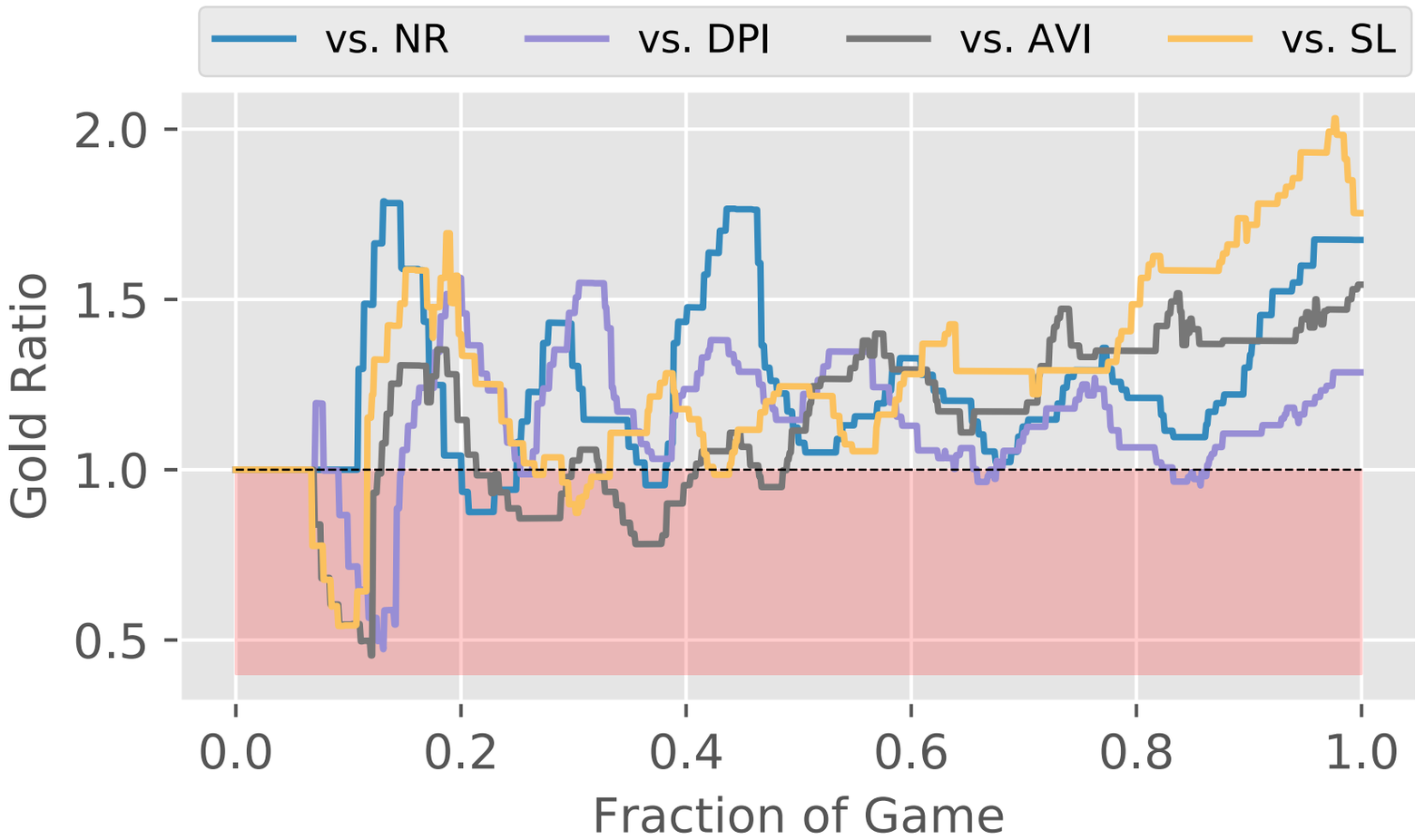
+ some exploration

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!