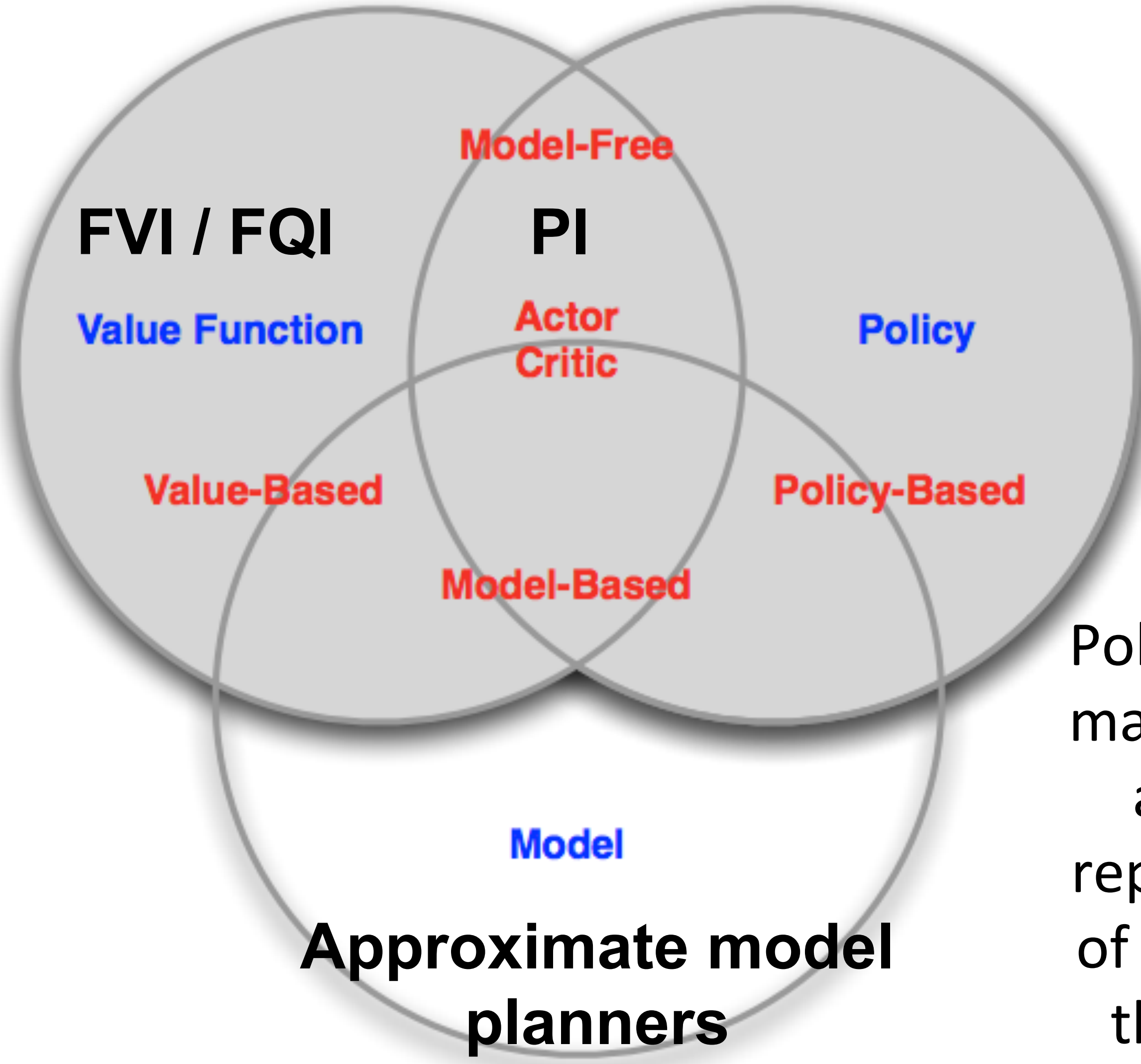
Approximate Models for Batch RL

Emma Brunskill

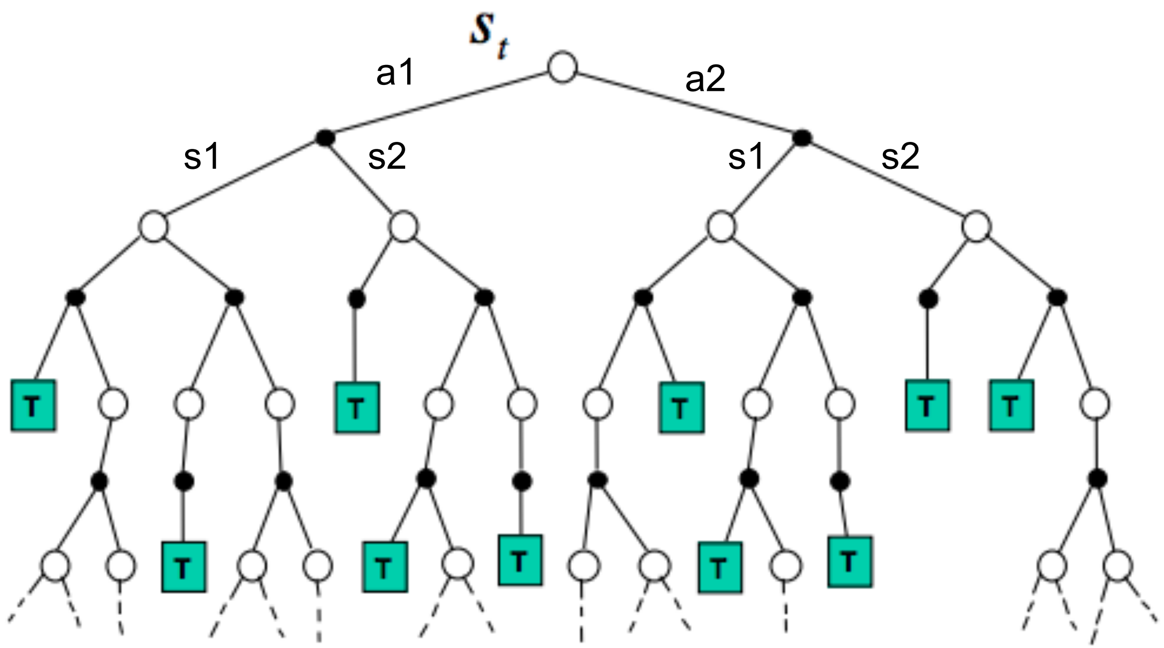


Policy Iteration maintains both an explicit representation of a policy and the value of that policy

Image from David Silver

Forward Search w/Generative Model

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead

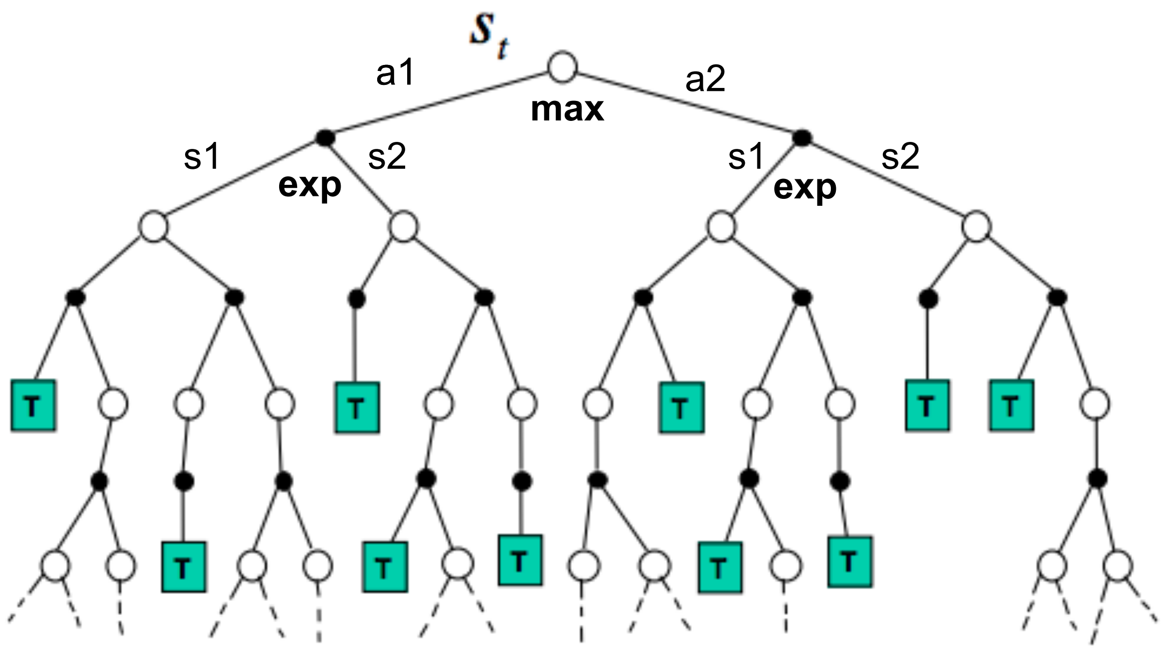


- No need to solve whole MDP, just sub-MDP starting from now

Slide modified from David Silver

Exact/Exhaustive Forward Search

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead

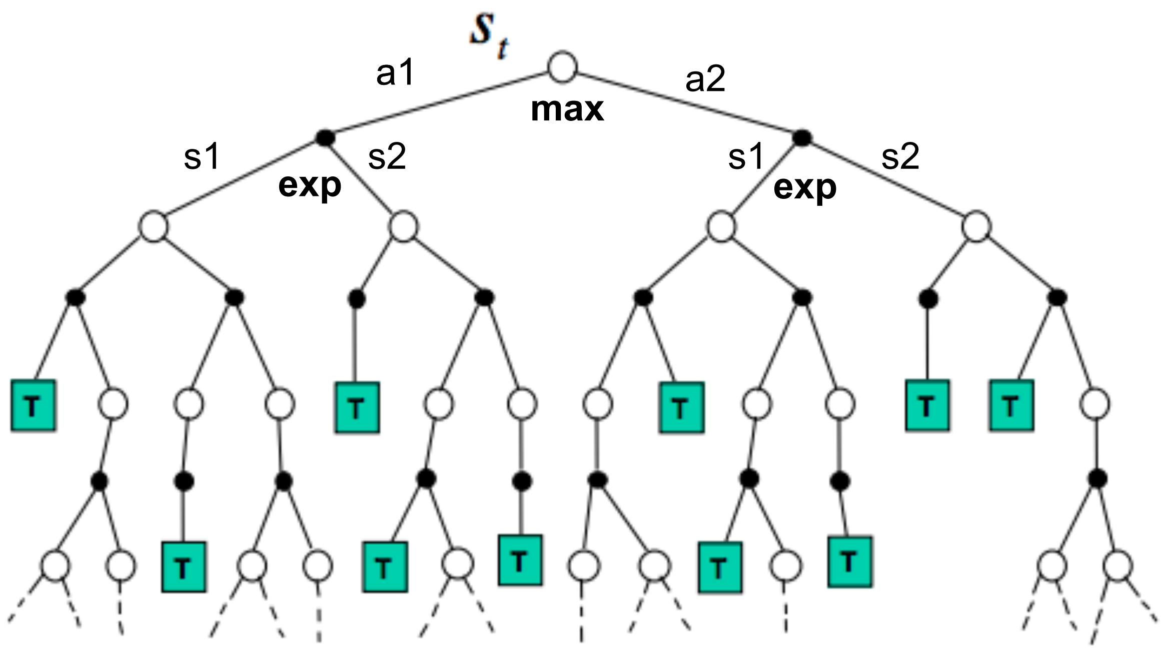


- No need to solve whole MDP, just sub-MDP starting from now

Slide modified from David Silver

How many nodes in a H-depth tree (as a function of state space |S| and action space |A|)?

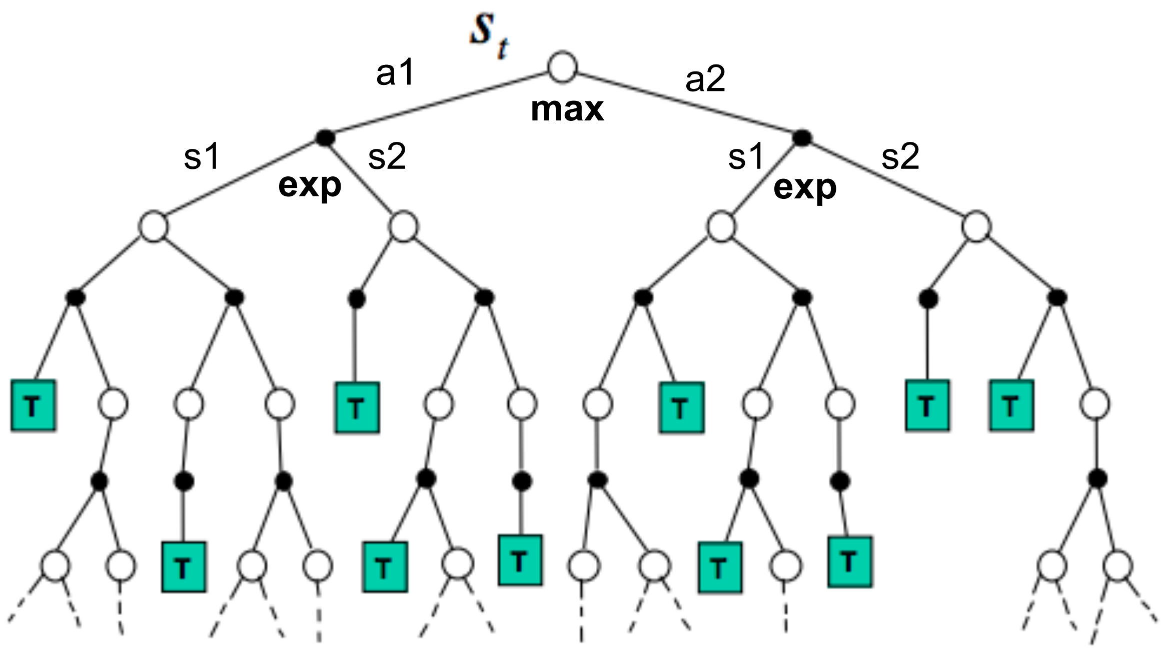
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- No need to solve whole MDP, just sub-MDP starting from now

How many nodes in a H-depth tree (as a function of state space |S| and action space |A|)? **(|S||A|)^H**

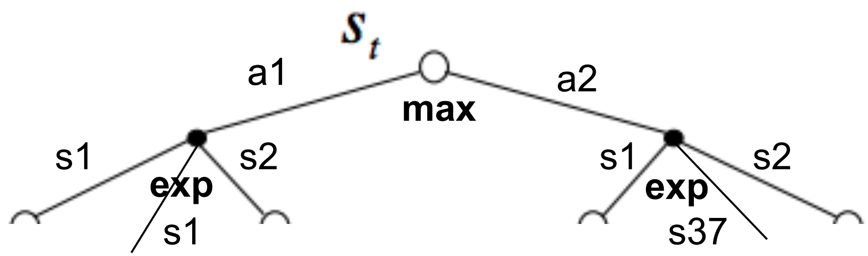
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- No need to solve whole MDP, just sub-MDP starting from now

Sparse Sampling: Don't Enumerate All Next States, Instead Sample Next States s' ~ P(s'|s,a)

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



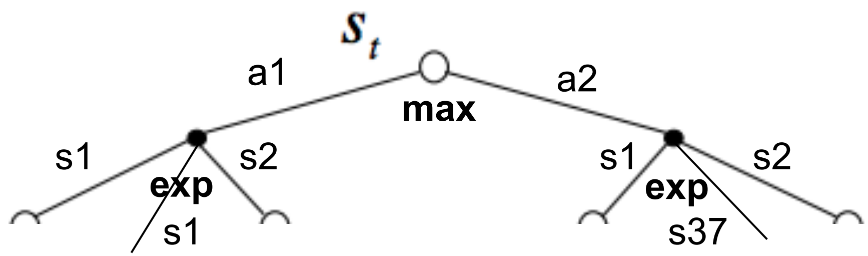
Sample n next states, s_i ~ P(s'|s,a)

Compute (1/n) Sum_i V(s_i)

Converges to expected future reward: $\text{Sum}_{s'}$ p(s'|s,a)V(s')

Sparse Sampling: # nodes if sample n states at each action node? Independent of |S|! $O(n|A|)^H$

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



Sample n next states, $s_i \sim P(s'|s,a)$

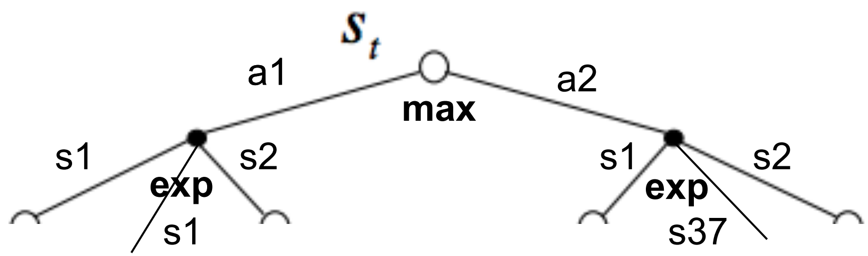
Compute $(1/n)\ \mathrm{Sum}_i\ V(s\_i)$

Converges to expected future reward: $\mathrm{Sum}_{s'}\ p(s'|s,a)V(s')$

Slide modified from David Silver

Sparse Sampling: # nodes if sample n states at each action node? Independent of |S|! $O(n|A|)^H$

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead

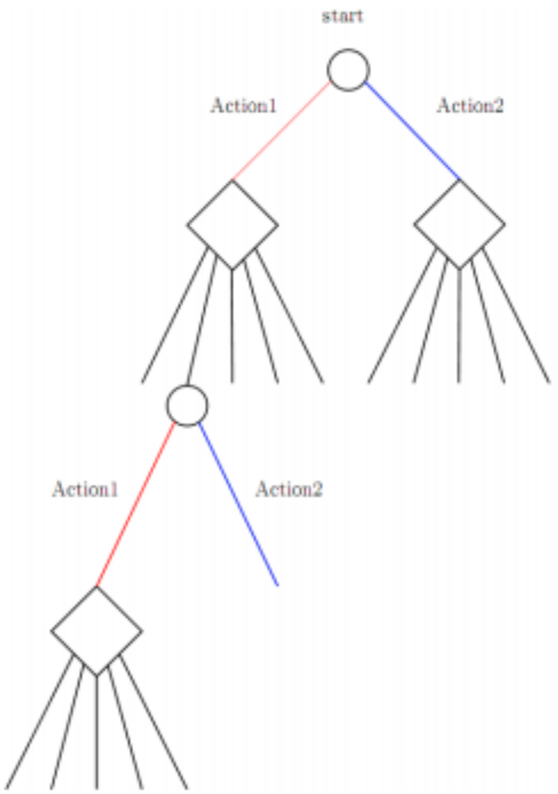


Upside: Can choose n to achieve bounds on the accuracy of the value function at the root state, independent of state space size

Downside: Still exponential in horizon, n still large for good bounds

Limitation of Sparse Sampling

- Sparse sampling wastes time on bad parts of tree
  - Devotes equal resources to each state encountered in the tree
  - Would like to focus on most promising parts of tree
- But how to control exploration of new parts of tree vs. exploiting promising parts?



Monte Carlo Tree Search

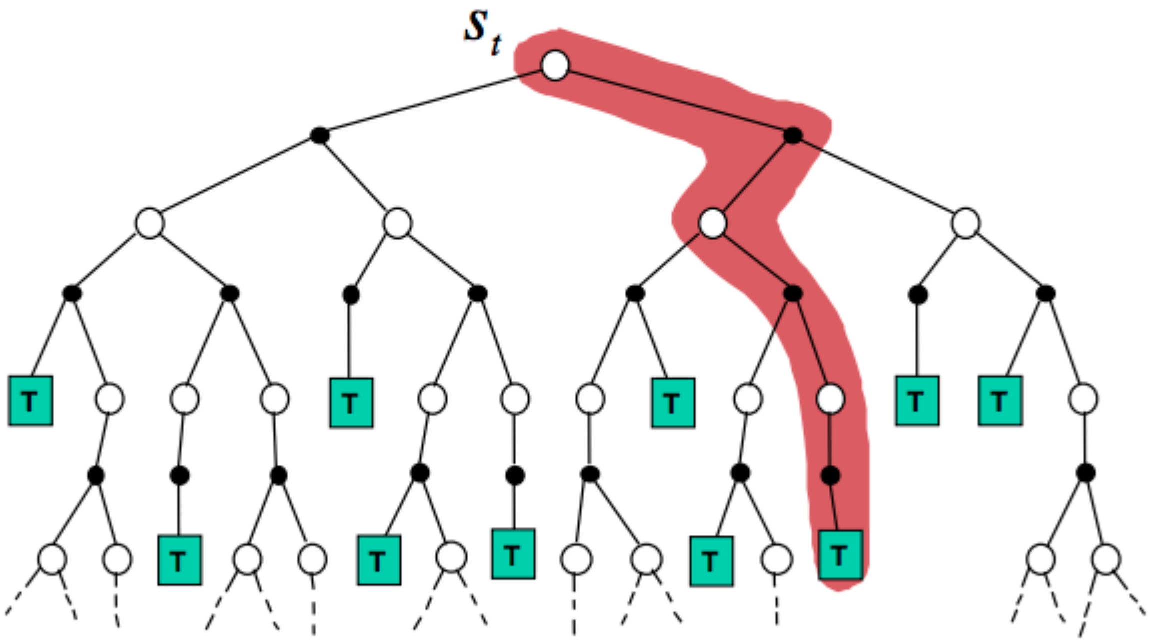
Combine ideas of sparse sampling with an adaptive method for focusing on more promising parts of the ree

Here “more promising” means the actions that are seem likely to yield higher long term reward

Uses the idea of simulation search

Simulation Based Search

- Forward search paradigm using sample-based planning
- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes



Simulation based Search

- Simulate episodes of experience from now with the model

$$\{s_t^k, A_t^k, R_{t+1}^k, \ldots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$$

- Apply model-free RL to simulated episodes
  - Monte-Carlo control $\rightarrow$ Monte-Carlo search
  - Sarsa $\rightarrow$ TD search

Simple Monte Carlo Search

- Given a model $\mathcal{M}_\nu$ and a rollout policy π
- For each action $a \in \mathcal{A}$
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, ..., S^k_T\}^K_{k=1} \sim \mathcal{M}_\nu, \pi$$

  - Evaluate actions by mean return (Monte-Carlo evaluation)

$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a)$$

- Select current (real) action with maximum value

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s_t, a)$$

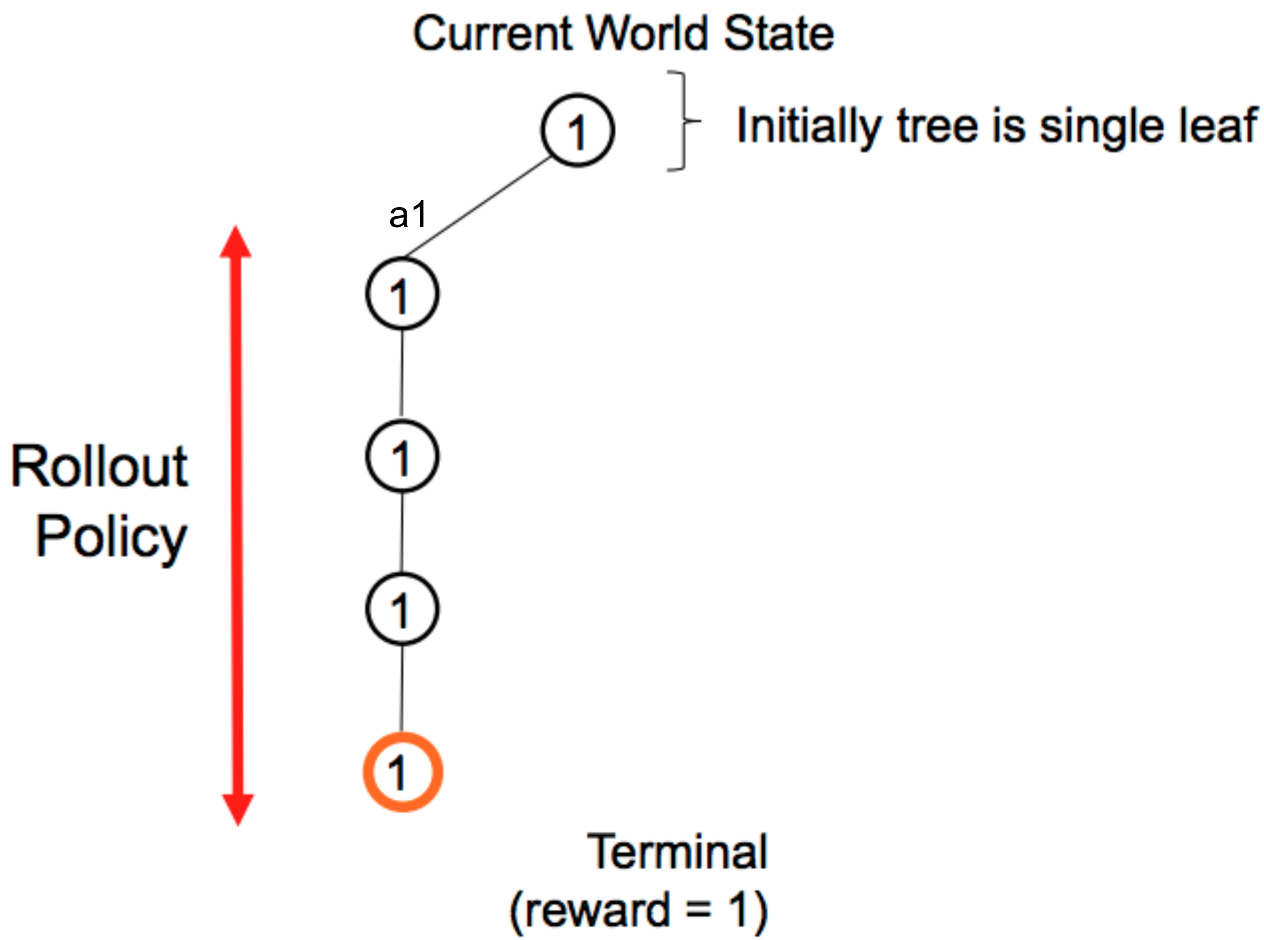
greedy improvement with respect to fixed rollout policy

Slide modified from David Silver

Upper Confidence Tree (UCT)
[Kocsis & Szepesvari, 2006]

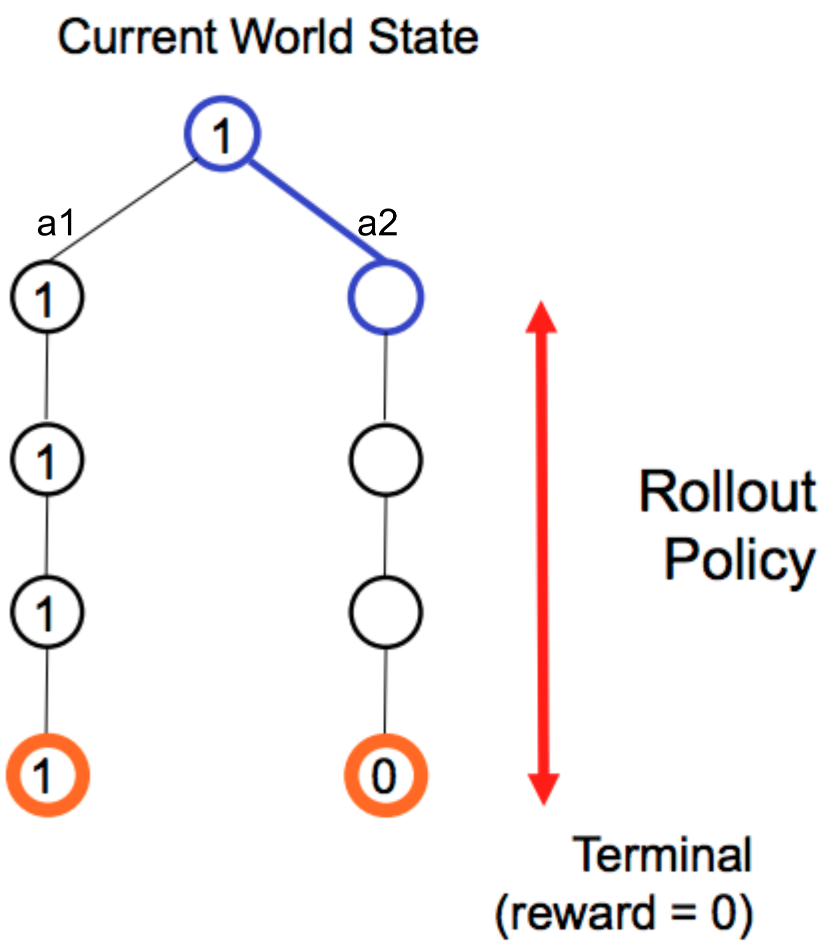
- Combine forward search and simulation search
- Instance of Monte-Carlo Tree Search
  - Repeated Monte Carlo simulation of rollout policy
  - Rollouts add one or more nodes to search tree
- UCT
  - Uses optimism under uncertainty idea
  - Some nice theoretical properties
  - Much better realtime performance than sparse sampling

At a leaf node perform a random rollout



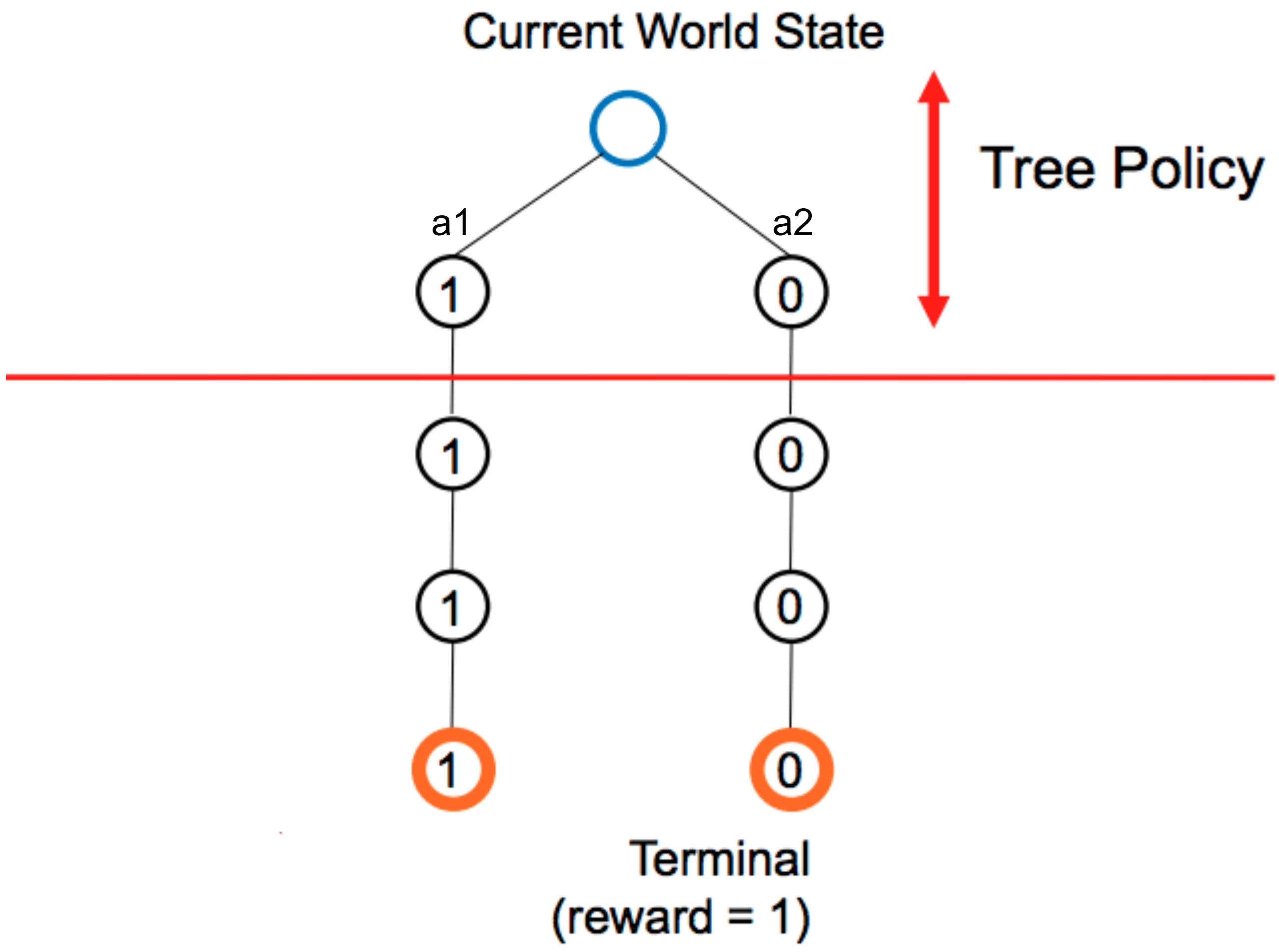
Slide modified from Alan Fern

Must select each action at a node at least once



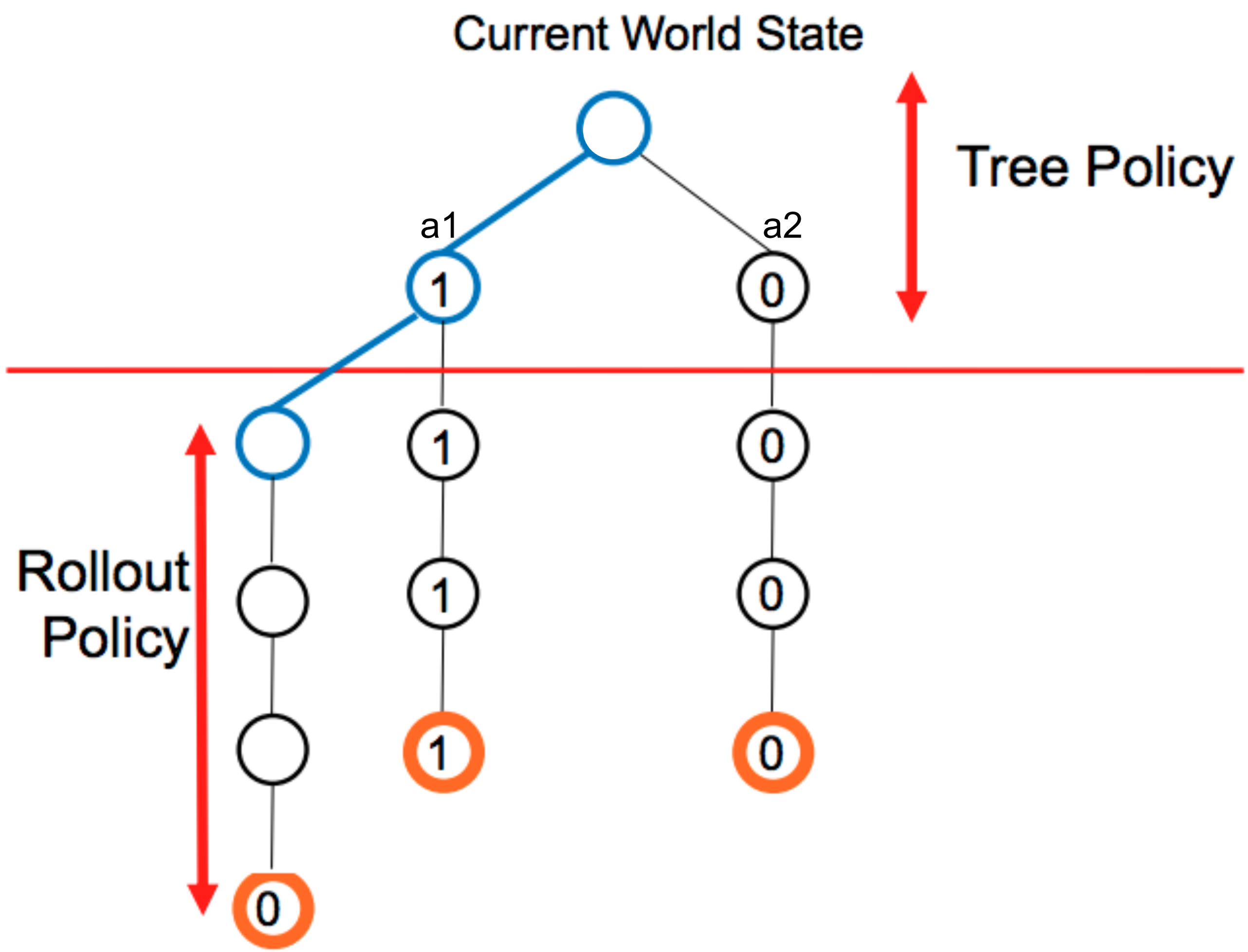
F

When all node actions tried once, select action according to tree policy



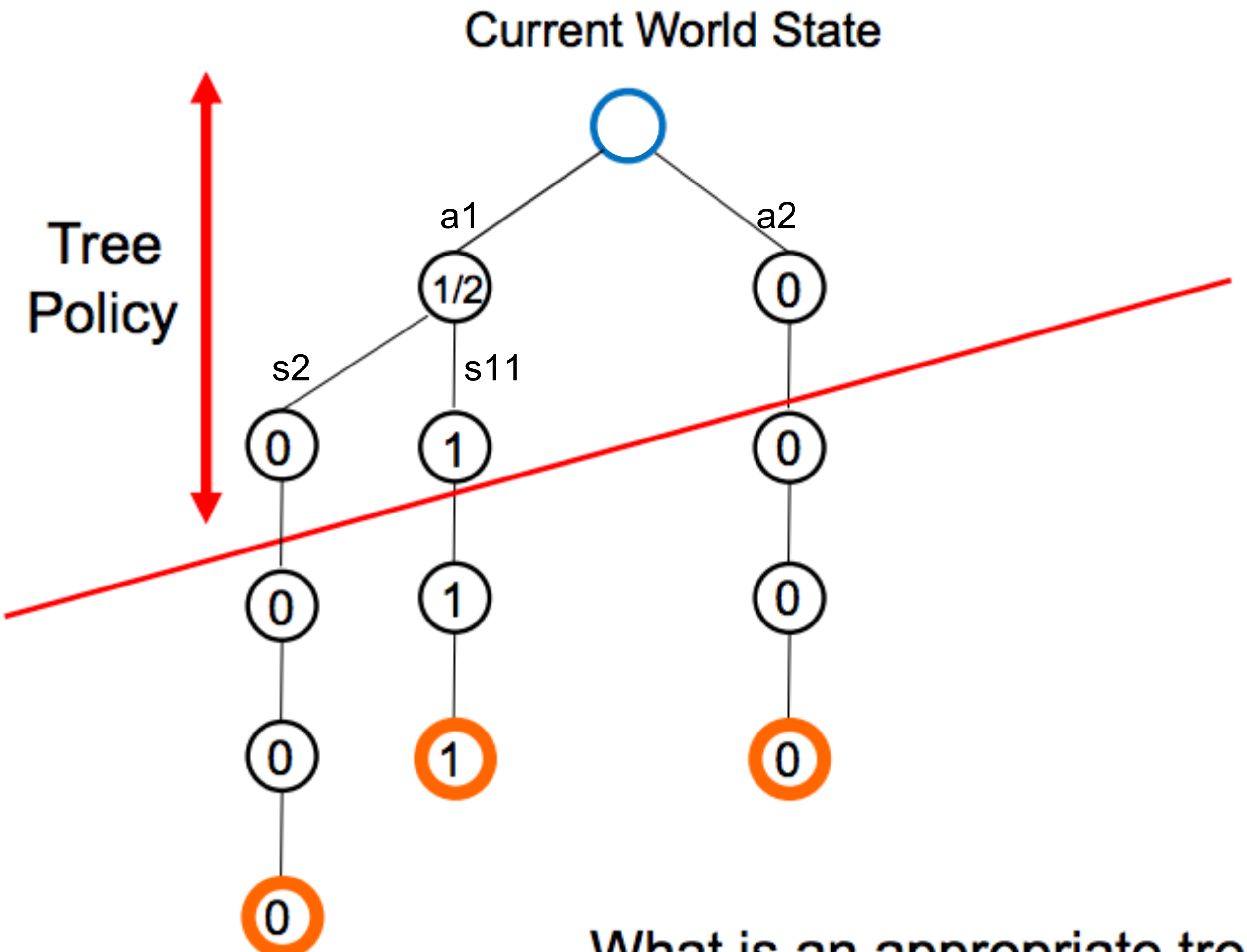
chercheurs.lille.inria.fr/~munos/papers/files/AAAI2013_slides.pdf

ns tried once, select action according to tree policy



Slide modified from Alan Fern

When all node actions tried once, select action according to tree policy



What is an appropriate tree policy?
Rollout policy?

UCT Algorithm [Kocsis & Szepesvari, 2006]

- Basic UCT uses random rollout policy

- Tree policy is based on UCB: (Upper Confidence Bound)
  - Q(s,a) : average reward received in current trajectories after taking action a in state s
  - n(s,a) : number of times action a taken in s
  - n(s) : number of times state s encountered

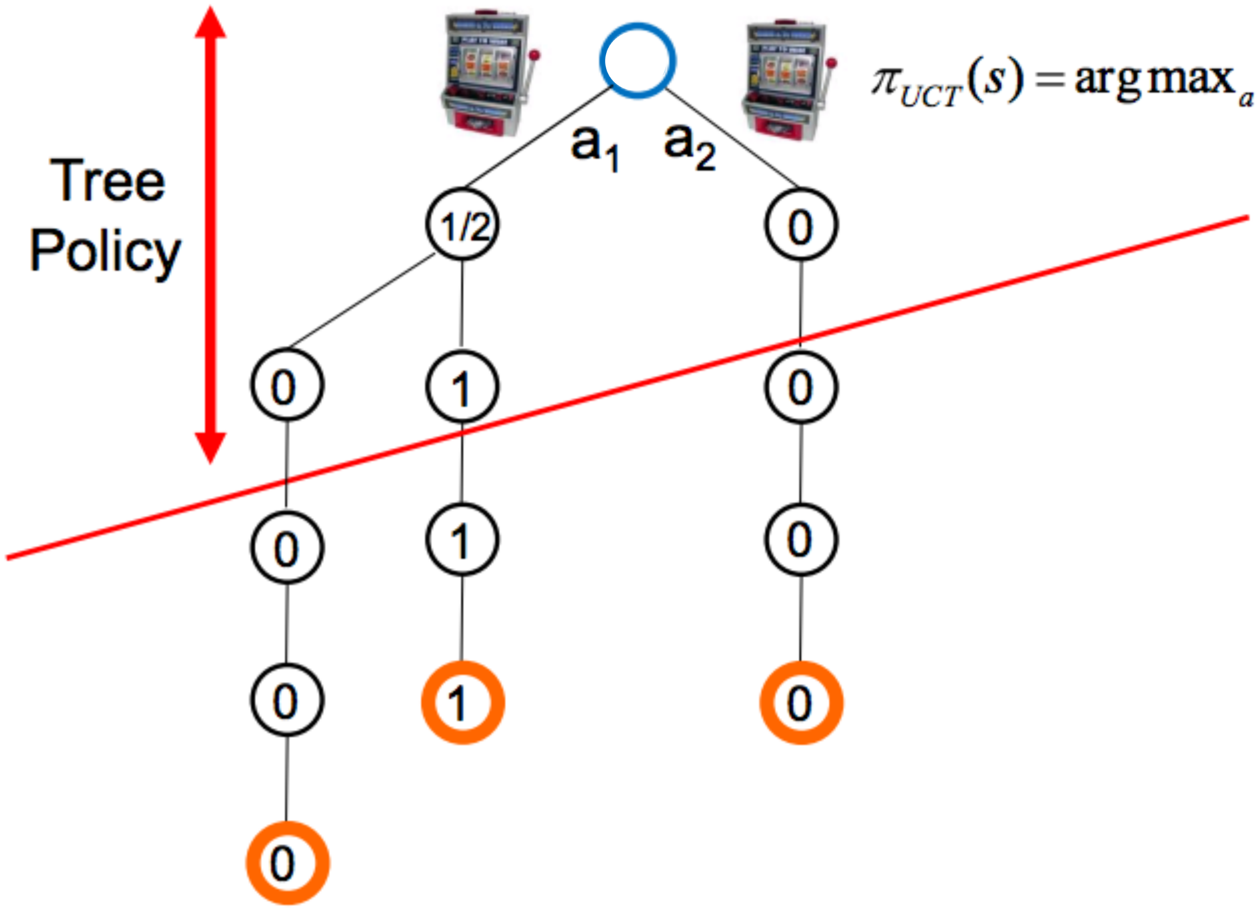
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Theoretical constant that must be selected empirically in practice

chercheurs.lille.inria.fr/~munos/papers/files/AAAI2013_slides.pdf

tried once, select action according to tree policy

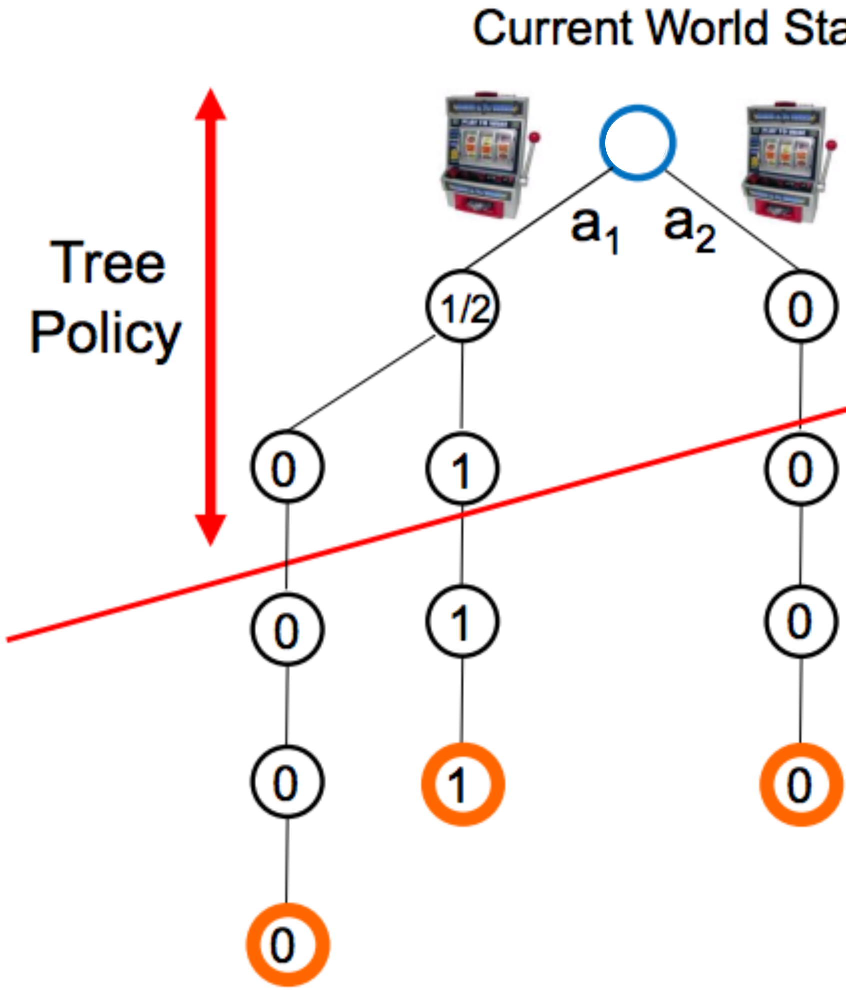
Current World State



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

chercheurs.lille.inria.fr/~munos/papers/files/ AAAI2013_slides.pdf

tried once, select action according to tree policy



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

- Requires us to have a simulator/ generative model
- Each pass down the tree, follow tree policy until reach a state leaf where not all actions have been tried.
- Then need to simulate starting from that state leaf the result of taking another action

Guarantees on UCT

[Kocsis and Szepesvári, 2006]

- In a tree with finite depth, all leaves will be eventually explored an infinite number of times, thus by backward induction, UCT is consistent and the regret is $O(\log n)$.
- However, the constant can be so bad that there is not finite-time guarantee for any reasonable n.

Computer Go

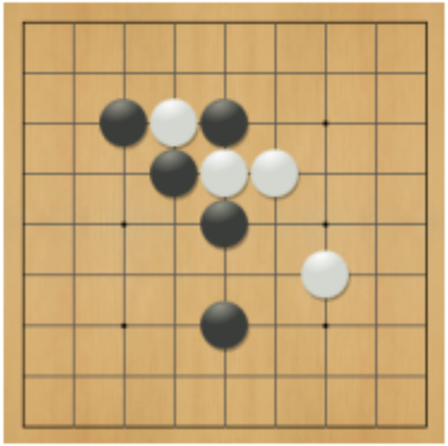
Previous game tree approaches faired poorly

- *2005*: Computer Go is impossible!
- *2006*: UCT invented and applied to 9x9 Go *(Kocsis, Szepesvari; Gelly et al.)*
- *2007*: Human master level achieved at 9x9 Go *(Gelly, Silver; Coulom)*
- *2008*: Human grandmaster level achieved at 9x9 Go *(Teytaud et al.)*

Slide modified from slides from Alan Fern & David Silver

Rules of Go

- Usually played on 19x19, also 13x13 or 9x9 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game



Slide modified from David Silver

Position Evaluation in Go

- How good is a position s?
- Reward function (undiscounted):

$$R_t = 0 \text{ for all non-terminal steps } t < T$$

$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

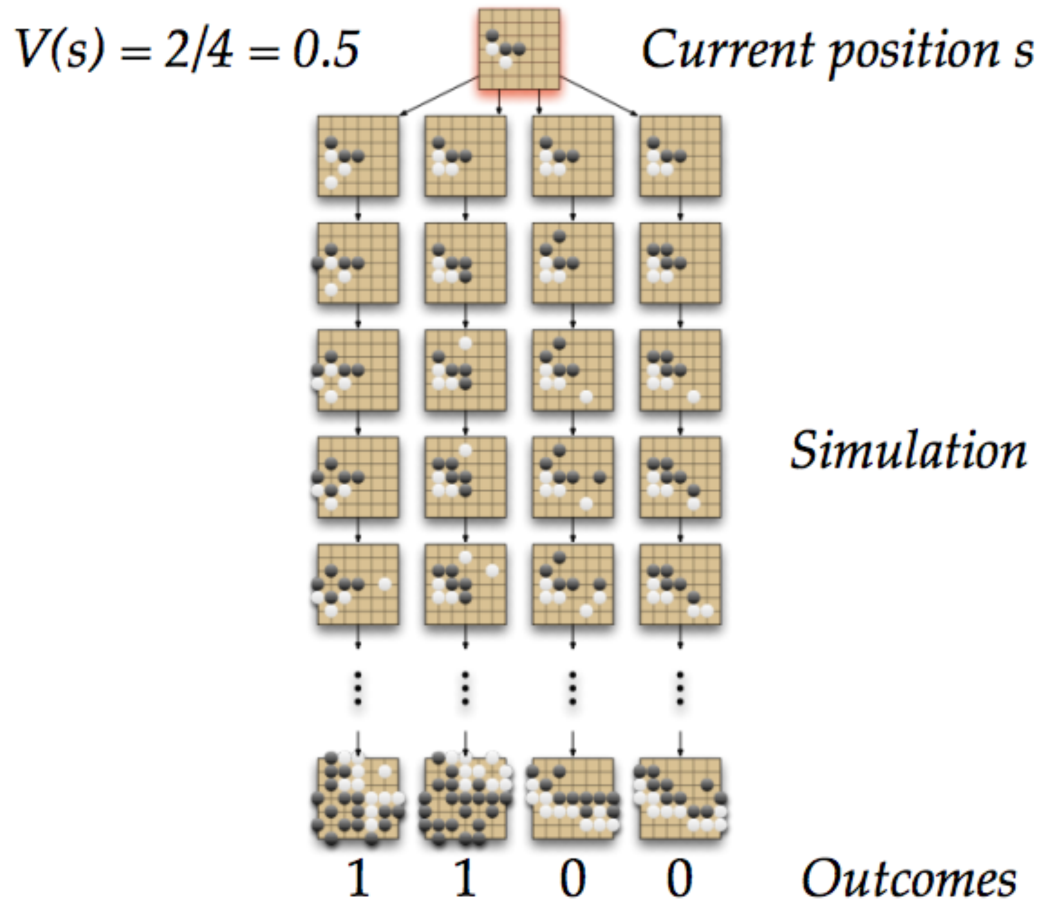
$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid S = s \right] = \mathbb{P} \left[\text{Black wins} \mid S = s \right]$$

$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

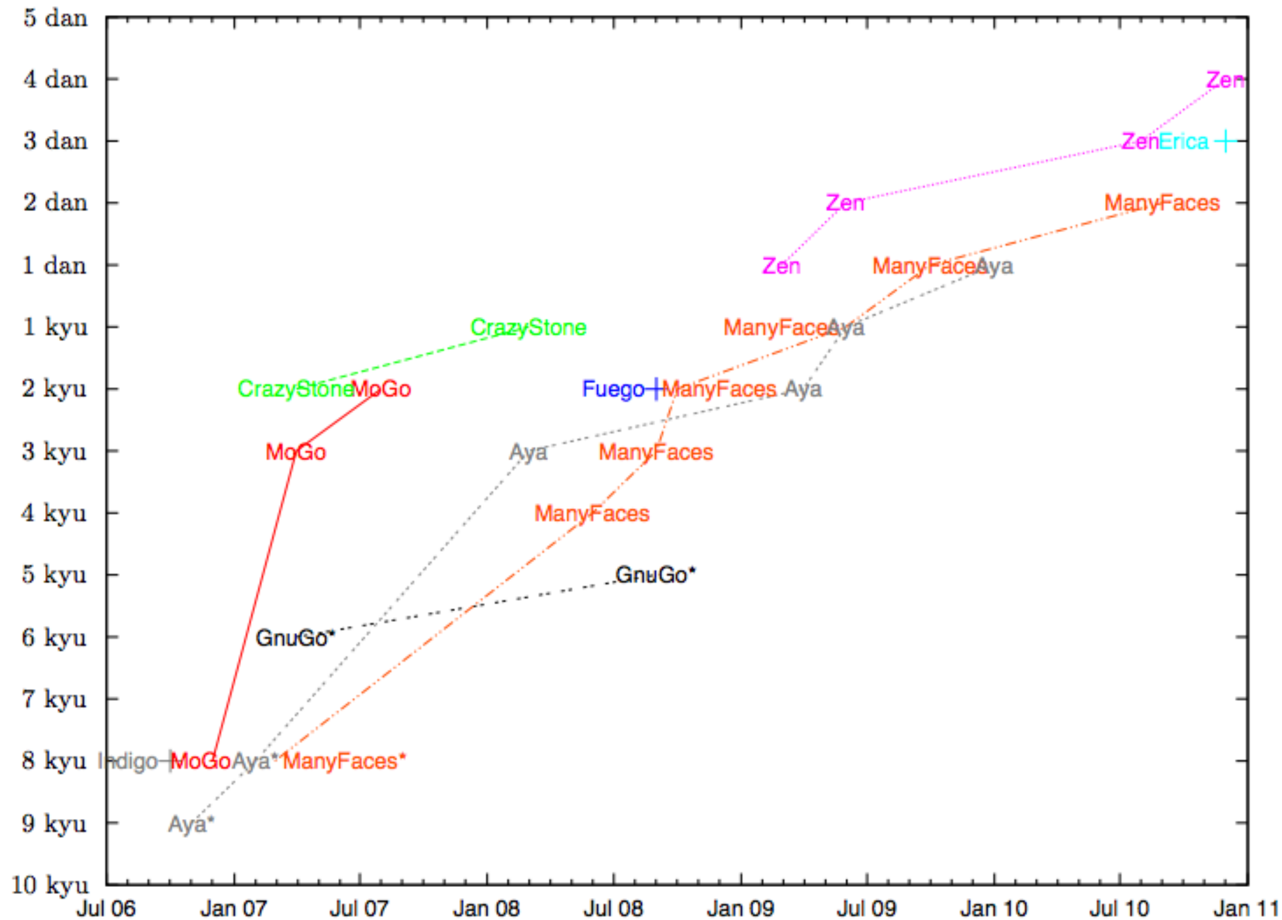
Slide modified from David Silver

Monte Carlo Evaluation in Go:

Planning problem, just a very very hard one



Enormous Progress. MCTS Huge Impact



Slide modified from David Silver

Going Back to Batch RL...

- Use supervised learning method to compute model
- Use learned model with MCTS planning
  - Note: error in model will impact error in estimated values!
- Computes an action for current state, take action, then redo planning for next state

Autonomous Driving using Texplore (Hester and Stone 2013)

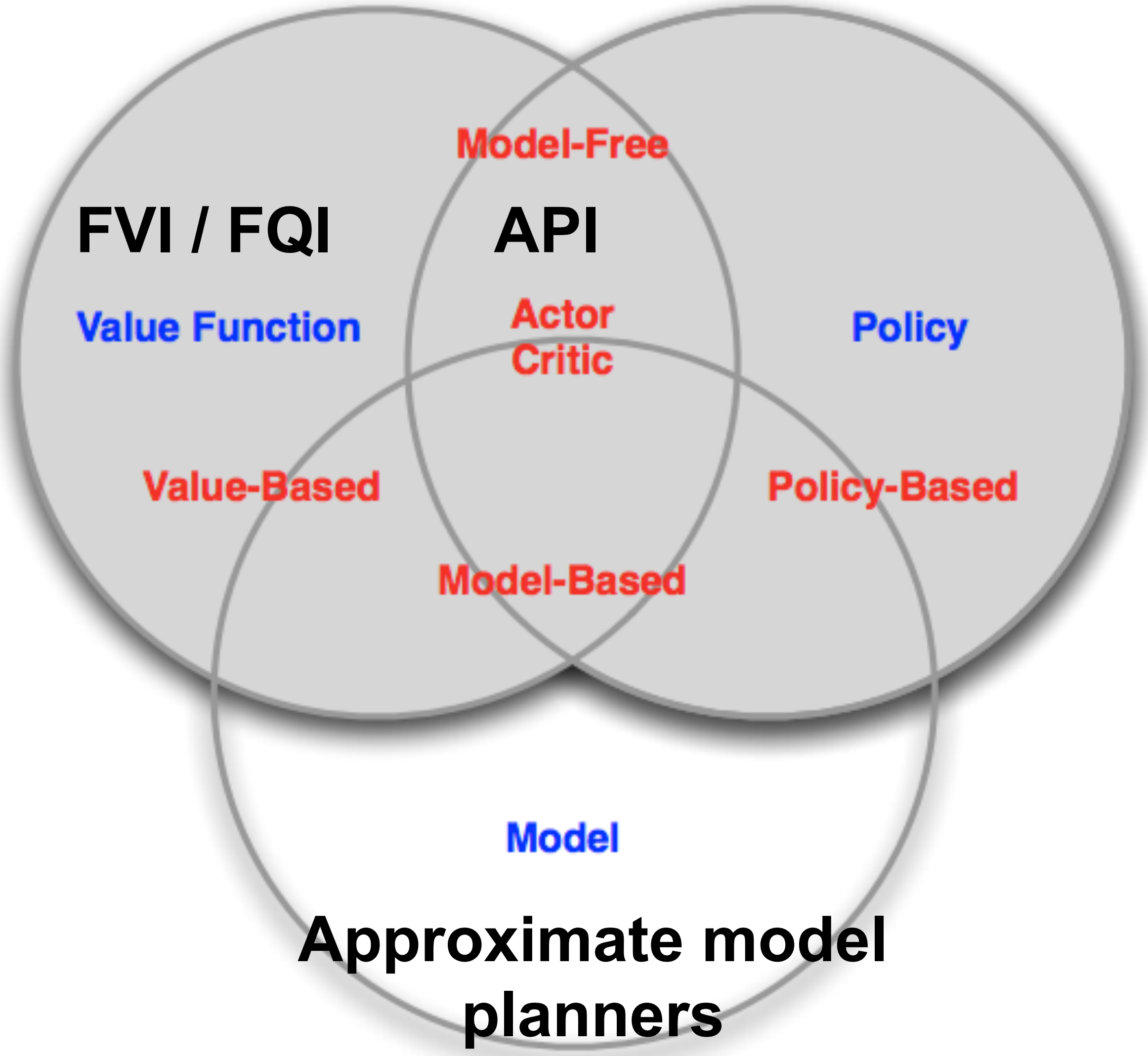


Image from David Silver