A quick guide to Markov Decision Processes and Reinforcement Learning

Oleg Szehr
Dalle Molle Institute for Artificial Intelligence (IDSIA), Switzerland
oleg.szehr@idsia.ch

May 12, 2022

Hedging of Financial Derivative contracts

Derivative contract: Legal agreement with two counterparties. One pays now, the other in the future, depending on the random behavior of an underlying asset.

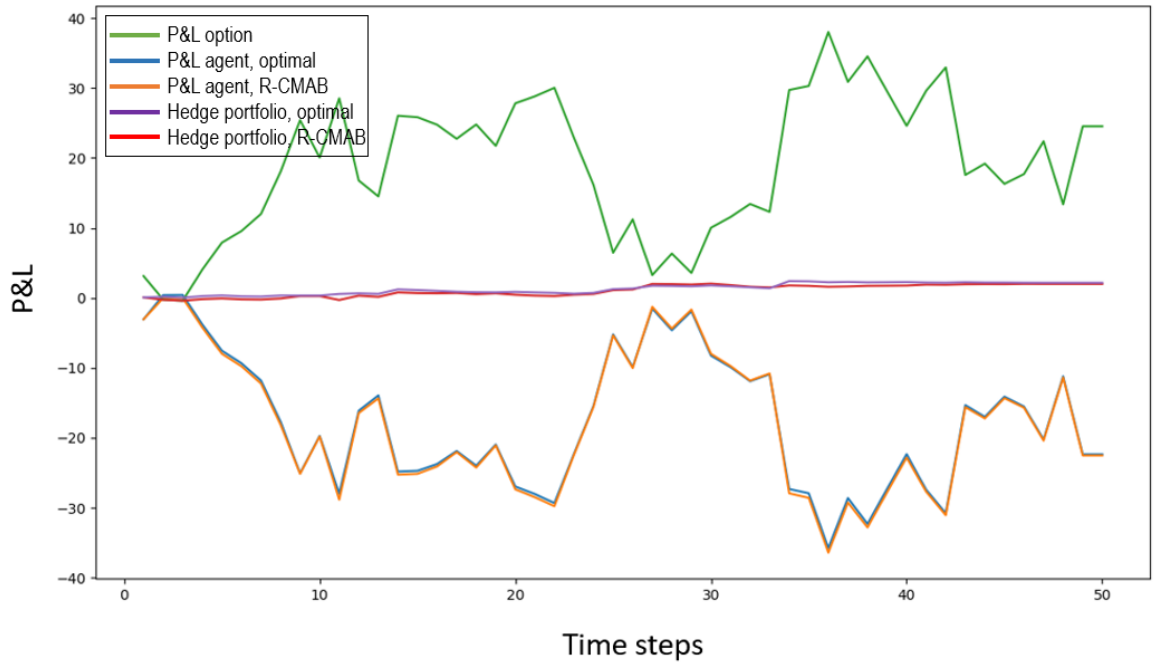
1. Airline realized that if oil price rises, they wont be able to pay their business: $\rightarrow$ enters option on oil

Pricing: What price should Credit Suisse ask today?

Hedging: Off-set risks by replicating derivative over investment horizon using hedging portfolio.

$\downarrow$

Planning problem in stochastic environment



The k-armed bandit

Prototypical example of Reinforcement Learning (RL):

Learner chooses repeatedly from k actions starting with 0 knowledge.
Receives reward upon each choice.

Goal: *Learn from rewards and find best action!*

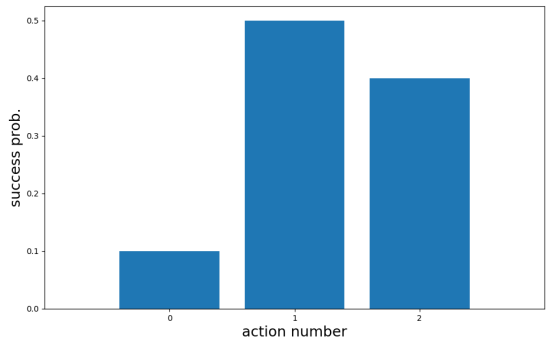
The k-armed bandit

Prototypical example of Reinforcement Learning (RL):

Learner chooses repeatedly from k actions starting with 0 knowledge. Receives reward upon each choice.

Goal: *Learn from rewards and find best action!*

Example 1: (Bernoulli Bandit)
Each round action a_1, a_2, a_3 chosen.
Playing a_i agent receives reward $\{0, 1\}$. 500 rounds. Probability of 1 is p_i, fixed over rounds but unknown.



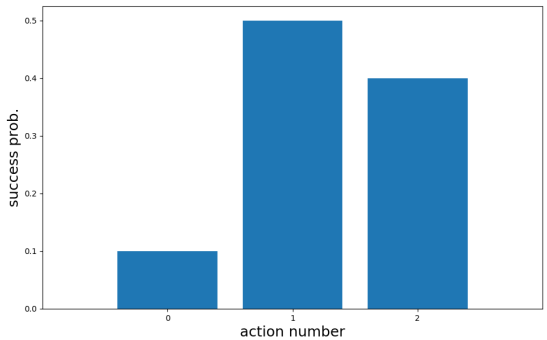
The k-armed bandit

Prototypical example of Reinforcement Learning (RL):

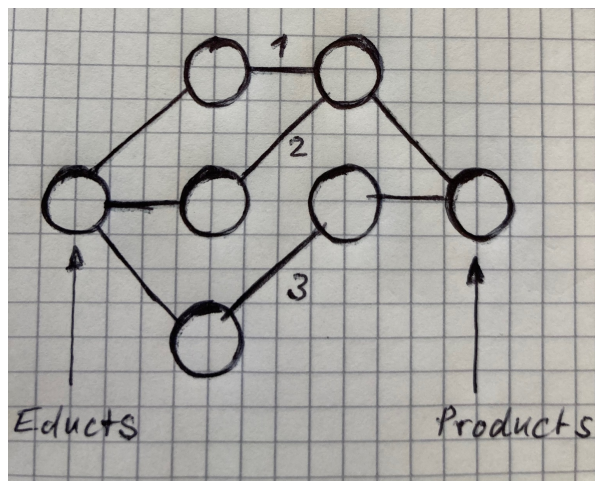
Learner chooses repeatedly from k actions starting with 0 knowledge.
Receives reward upon each choice.

Goal: *Learn from rewards and find best action!*

Example 1: (Bernoulli Bandit)
Upon playing action a_i agent receives reward in $\{0, 1\}$. Probability of 1 is p_i.
In each round p_i fixed but unknown.



Example 2: (Process cost minimization)
Follow different paths through graph, receiving a terminal cost per path.
Path corresponds to one (of exponentially many) actions.
Find cheapest course of action/ process.



The k-armed bandit

Formally:

- *k-armed bandit* is tuple $(A, R, T, \textit{Bandit iteration})$ with
  1. $A = \{a_1, ..., a_k\}$ set of possible actions
  2. $R(a)$ reward obtained by taking $a \in A$, i.i.d over time
  3. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *Bandit iteration*:
  1. *Agent* chooses $a \in A$
  2. Agent receives $R(a)$

The k-armed bandit

Formally:

- *k-armed bandit* is tuple $(A, R, T, \textit{Bandit iteration})$ with
  1. $A = \{a_1, ..., a_k\}$ set of possible actions
  2. $R(a)$ reward obtained by taking $a \in A$, i.i.d over time
  3. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *Bandit iteration*:
  1. *Agent* chooses $a \in A$
  2. Agent receives $R(a)$

How to solve bandit problem?

1. Supervised learning:
   1. Train agent on history $\mathcal{H}_{t-1} = \{(a_{k_1}, R(a_{k_1}), ..., (a_{k_{t-1}}, R(a_{k_{t-1}}))\}$
   2. Greedy choice: $a_{k_t} = \text{argmax } \textit{Learner}_{\mathcal{H}_{t-1}}$
   3. Receive $R(a_{k_t})$, add $(a_{k_t}, R(a_{k_t})$ to $\mathcal{H}_{t-1}$ and go to 1.

The k-armed bandit

Formally:

- *k-armed bandit* is tuple $(A, R, T, \textit{Bandit iteration})$ with
  1. $A = \{a_1, ..., a_k\}$ set of possible actions
  2. $R(a)$ reward obtained by taking $a \in A$, i.i.d over time
  3. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *Bandit iteration*:
  1. *Agent* chooses $a \in A$
  2. Agent receives $R(a)$

How to solve bandit problem?

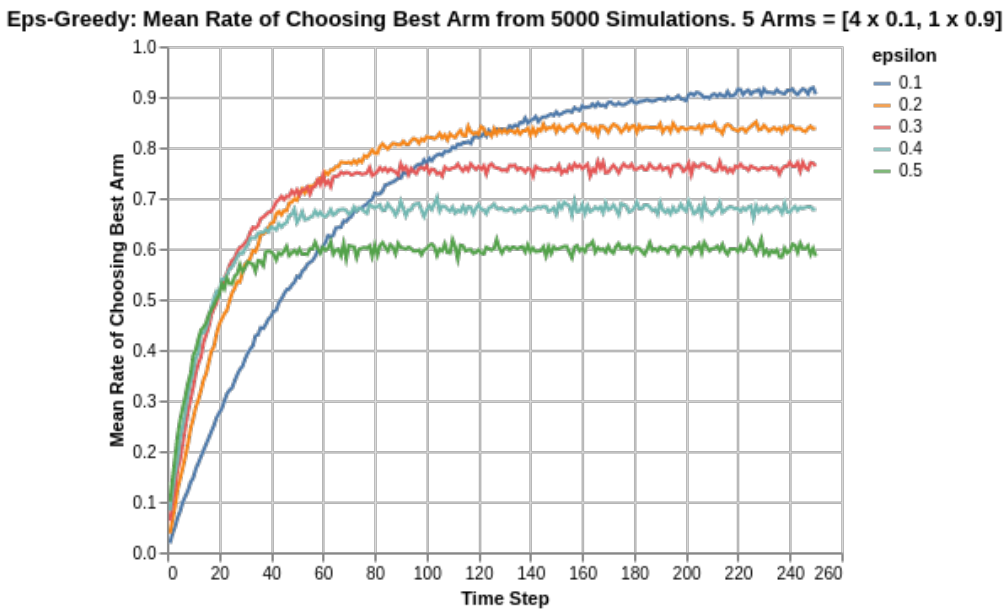
1. Supervised learning:
   1. Train agent on history $\mathcal{H}_{t-1} = \{(a_{k_1}, R(a_{k_1}), ..., (a_{k_{t-1}}, R(a_{k_{t-1}}))\}$
   2. Greedy choice: $a_{k_t} = \text{argmax } \textit{Learner}_{\mathcal{H}_{t-1}}$
   3. Receive $R(a_{k_t})$, add $(a_{k_t}, R(a_{k_t})$ to $\mathcal{H}_{t-1}$ and go to 1.

$\hookrightarrow$ **But:** get stuck in a_i and no guarantee it is good.

How to solve bandit problem?

2. ϵ-Dithering:
   1. Train agent on history
      $\mathcal{H}_{t-1} = \{(a_{k_1}, R(a_{k_1}), ..., (a_{k_{t-1}}, R(a_{k_{t-1}}))\}$
   2. Act greedy with prob. $1 - \epsilon$, random with prob. ϵ
   3. Receive $R(a_{k_t})$, add $(a_{k_t}, R(a_{k_t})$ to $\mathcal{H}_{t-1}$ and train ...

$\hookrightarrow$ **But:** ϵ 'knows nothing', i.e. keep exploring actions
i) forever *ii)* known to be bad



Thompson Sampling for the Bernoulli Bandit

Idea: *Bayesian model to represent learned content*

Recap on Bayes:

Bayes' rule:

X,Y discrete random vars.

$$P(Y = y|X = x) = \frac{P(X=x|Y=y)P(Y=y)}{P(X=x)}$$

Bayesian info. gain/ inference:

$$\underbrace{p(\text{model}|\text{data})}_{\text{posterior}} = \frac{\overbrace{P(\text{data}|\text{model})}^{\text{likelihood}}\overbrace{p(\text{model})}^{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$

Thompson Sampling for the Bernoulli Bandit

Idea: *Bayesian model to represent learned content*

Recap on Bayes:

Bayes' rule:
X,Y discrete random vars.
$P(Y = y|X = x) = \frac{P(X=x|Y=y)P(Y=y)}{P(X=x)}$

Bayesian info. gain/ inference:

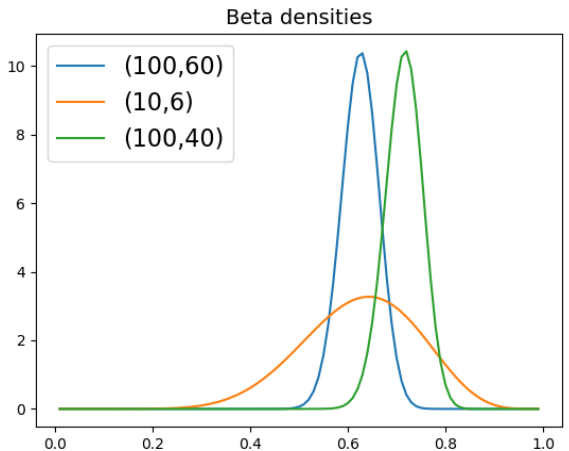
$$\underbrace{p(\text{model}|\text{data})}_{\text{posterior}} = \frac{\overbrace{P(\text{data}|\text{model})}^{\text{likelihood}}\overbrace{p(\text{model})}^{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$

Model: Param. model of expected rewards with Beta density:

$$\mathbb{E}[R(a_k)] \text{ has density } \rho_k(x) = \frac{1}{B(\alpha_k, \beta_k)} x^{\alpha_k - 1}(1-x)^{\beta_k - 1}$$

Properties of β:

1. If $\alpha_k = \beta_k = 1$ then uniform.
2. Mean: $\alpha_k/(\alpha_k + \beta_k)$.
3. Distribution concentrates as $\alpha_k + \beta_k$ grows.



Thompson Sampling for the Bernoulli Bandit

Update, for each action individually:

Prior for $\mathbb{E}[R(a_k)]$*:* $\rho_k(x)$

Likelihood: Prob. of reward $R(a_k)$ given $\rho_k(x)$

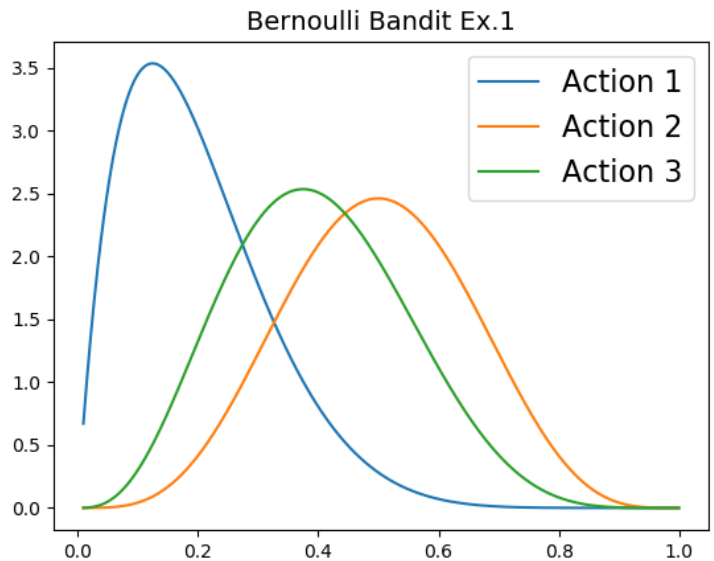
Posterior: $\rightarrow$ according to the updating rule:

Updating Rule: For action a_k posterior is again Beta with:

$$(\alpha_k^{posterior}, \beta_k^{posterior}) = (\alpha_k^{prior} + R(a_k), \beta_k^{prior} + (1 - R(a_k)))$$

$\rightarrow \alpha_k, \beta_k$ called pseudo-counters.

Learning Progress:



Thompson Sampling for the Bernoulli Bandit

Update, for each action individually:

Prior for $\mathbb{E}[R(a_k)]$*:* $\rho_k(x)$

Likelihood: Prob. of reward $R(a_k)$ given $\rho_k(x)$

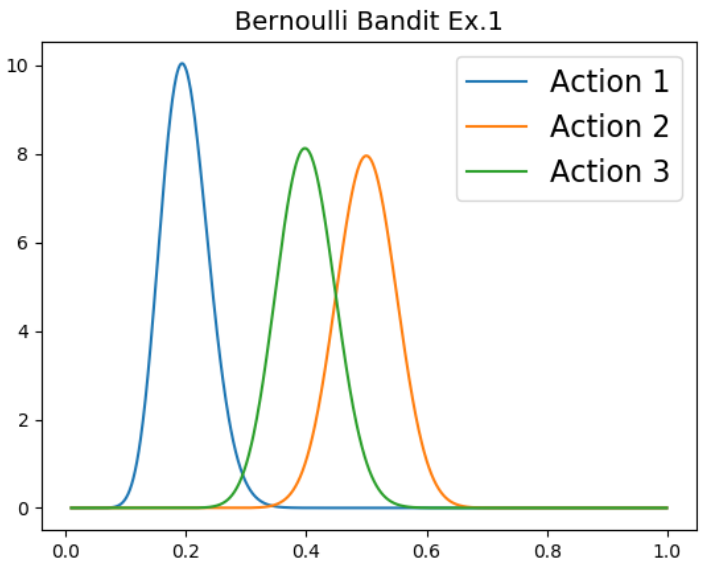
Posterior: $\rightarrow$ according to the updating rule:

Updating Rule: For action a_k posterior is again Beta with:

$$(\alpha_k^{posterior}, \beta_k^{posterior}) = (\alpha_k^{prior} + R(a_k), \beta_k^{prior} + (1 - R(a_k)))$$

$\rightarrow \alpha_k, \beta_k$ called pseudo-counters.

Learning Progress:



Thompson Sampling for the Bernoulli Bandit

Update, for each action individually:

Prior for $\mathbb{E}[R(a_k)]$*:* $\rho_k(x)$

Likelihood: Prob. of reward $R(a_k)$ given $\rho_k(x)$

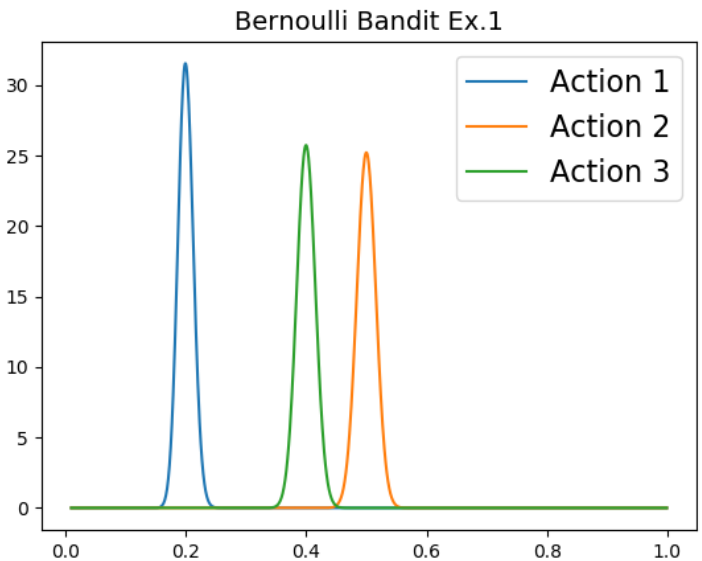
Posterior: $\rightarrow$ according to the updating rule:

Updating Rule: For action a_k posterior is again Beta with:

$$(\alpha_k^{posterior}, \beta_k^{posterior}) = (\alpha_k^{prior} + R(a_k), \beta_k^{prior} + (1 - R(a_k)))$$

$\rightarrow \alpha_k, \beta_k$ called pseudo-counters.

Learning Progress:



Thompson Sampling for the Bernoulli Bandit

Algorithms:

Greedy Beta:

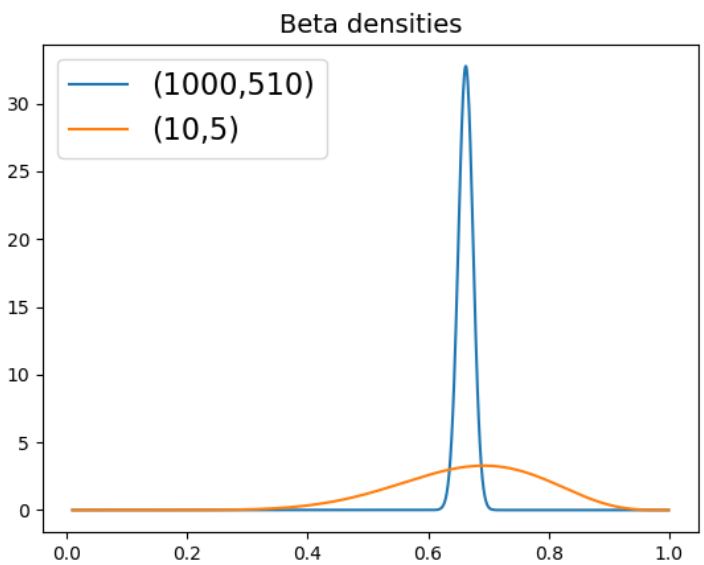
Compute estimates
$\hat{R}(a_k) = \alpha_k/(\alpha_k + \beta_k)$
Choose $a_k = \text{argmax}\hat{R}(a_k)$

Thompson Sampling:

Choose a_k according to the probability that it is optimal.

TS in practice: Sample $\hat{R}(a_k)$ from $\rho_k(x)$, $a_k = \text{argmax}\hat{R}(a_k)$

Why is Thompson sampling better?



- *Greedy:* Plays blue (orange) for ever.
- *Dithering:* First doesn't explore enough, then too much
- *Thompson:* Explore where it helps to identify optimal actions, but avoids trying futile actions.

Thompson Sampling for the Bernoulli Bandit

Algorithms:

Greedy Beta:

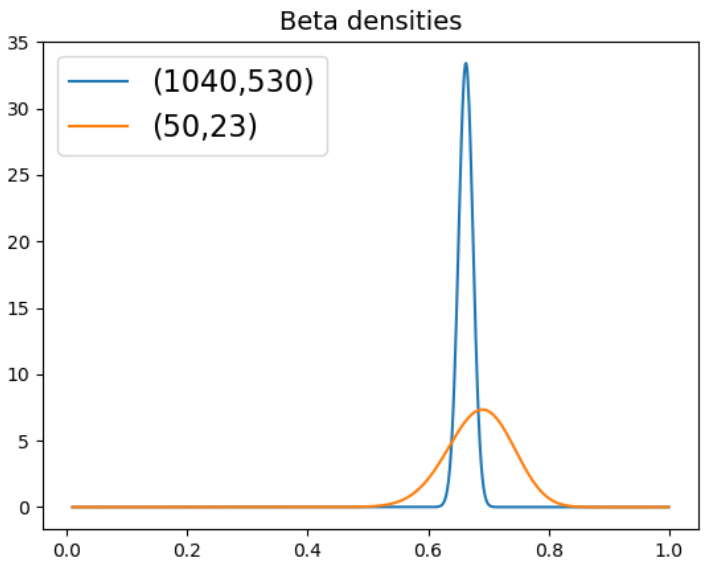
Compute estimates
$\hat{R}(a_k) = \alpha_k/(\alpha_k + \beta_k)$
Choose $a_k = \text{argmax}\hat{R}(a_k)$

Thompson Sampling:

Choose a_k according to the probability that it is optimal.

TS in practice: Sample $\hat{R}(a_k)$ from $\rho_k(x)$, $a_k = \text{argmax}\hat{R}(a_k)$

Why is Thompson sampling better?



- *Greedy:* Plays blue (orange) for ever.
- *Dithering:* First doesn't explore enough, then too much
- *Thompson:* Explore where it helps to identify optimal actions, but avoids trying futile actions.

Thompson Sampling for the Bernoulli Bandit

Algorithms:

Greedy Beta:

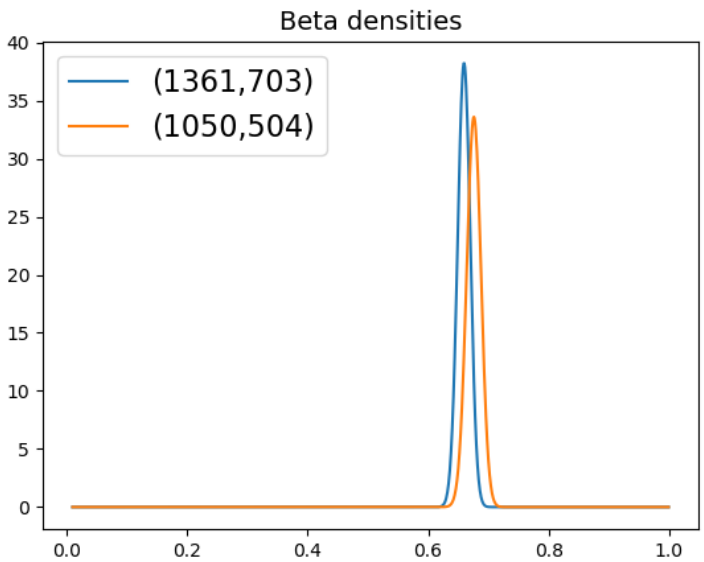
Compute estimates
$\hat{R}(a_k) = \alpha_k/(\alpha_k + \beta_k)$
Choose $a_k = \text{argmax}\hat{R}(a_k)$

Thompson Sampling:

Choose a_k according to the probability that it is optimal.

TS in practice: Sample $\hat{R}(a_k)$ from $\rho_k(x)$, $a_k = \text{argmax}\hat{R}(a_k)$

Why is Thompson sampling better?



- *Greedy:* Plays blue (orange) for ever.
- *Dithering:* First doesn't explore enough, then too much
- *Thompson:* Explore where it helps to identify optimal actions, but avoids trying futile actions.

Alternatives to Thompson Sampling

Important alternative:

- Algos. based on confidence interval for expected reward.
- UCB1: Maximize

$$UCB1(a_k) = \bar{R}(a_k) + c_{n,n_{a_k}},$$

$c_{n,l} = \sqrt{2\ln(n)/l}$, $\bar{R}(a_k)$ average realized reward,
n_{a_k} number of chosen a_k

- Know-how: 'Thomp. sam. and UCB1 essentially optimal'

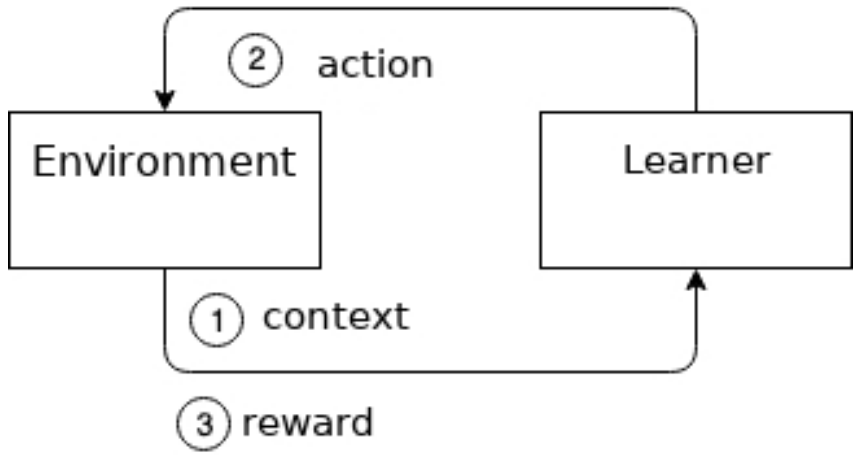
Practical:

- Other distributions are possible in Thomp. sam., e.g. normal, log-normal priors, but not all are practical
- Computing posteriors and sampling expensive; Approximate posterior sampling: Overkill!
- For real-world probs. Thom. sam. with *normal* prior often (if not too high dimension) good enough. Normal distribution has closed form Bayes update formulas.

Contextual Bandits

As k-armed bandit but learner faces *non-stationary* reward distributions.
To assist choice: Agent receives *context* information

Goal: *Associate expected reward to context and find rule of action!*

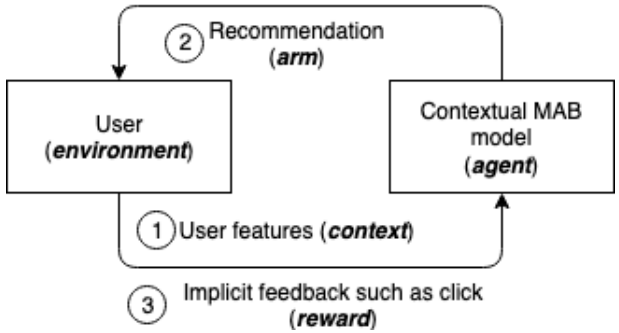


- Interpret context as 'state of system' or 'hint'
- Without context learning non-stationary reward distributions 'impossible' (unless assumptions, e.g. pseudo-stationarity).
- Goal: No universal optimal action, but rule of action: *policy*

But: Learners actions have *no influence on context*.
Actions do not influence state of the environment.

Contextual Bandits: Examples

Online adds (Amazon):



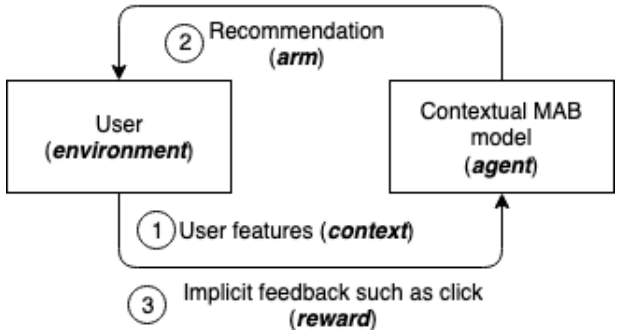
- *Context:* User enters website with her data
- *Action:* Agent chooses advertisement to post
- *Reward:* Agent clicks 'BUY :-)'

Portfolio Selection (UBS):

- *Context:* Investor considers markets today
- *Action:* Investor chooses assets to invest (Northrop Grum. & Lockheed Mar. Stoxx, Gold)
- *Reward:* Investor counts money tomorrow

Contextual Bandits: Examples

Online adds (Amazon):



- *Context:* User enters website with her data
- *Action:* Agent chooses advertisement to post
- *Reward:* Agent clicks 'BUY :-)'

Portfolio Selection (UBS):

- *Context:* Investor considers markets today
- *Action:* Investor chooses assets to invest (Northrop Grum. & Lockheed Mar. Stoxx, Gold)
- *Reward:* Investor counts money tomorrow

'Living for the moment' (PhD):

- *Context:* State of the world
- *Action:* Go to a bar today or prepare lecture tomorrow

$\hookrightarrow$ Context. bandit/lecturer goes to the bar, no future reward plans

Contextual Bandits: Formalization

Formally: (differences to the ordinary bandit in red)

- *Contextual bandit* is tuple $(S, A, R, T, \textit{Bandit iteration})$ with
  1. $S = \{s_1, s_2, s_3, ..., \}$ set of admissible states/ contexts
  2. $A = \{a_1, ..., a_k\}$ set of possible actions
  3. $R(s, a)$ reward obtained by taking $a \in A$ given context $s \in S$
  4. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *Contextual-bandit iteration*:
  1. Agent receives context $s \in S$
  2. Agent chooses a given s with prob. $\pi(a|s)$, π *policy*
  3. Agent receives $R(s, a)$

Contextual Bandits: Formalization

Formally: (differences to the ordinary bandit in red)

- *Contextual bandit* is tuple $(S, A, R, T, \textit{Bandit iteration})$ with
  1. $S = \{s_1, s_2, s_3, ..., \}$ set of admissible states/ contexts
  2. $A = \{a_1, ..., a_k\}$ set of possible actions
  3. $R(s, a)$ reward obtained by taking $a \in A$ given context $s \in S$
  4. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *Contextual-bandit iteration*:
  1. Agent receives context $s \in S$
  2. Agent chooses a given s with prob. $\pi(a|s)$, π *policy*
  3. Agent receives $R(s, a)$

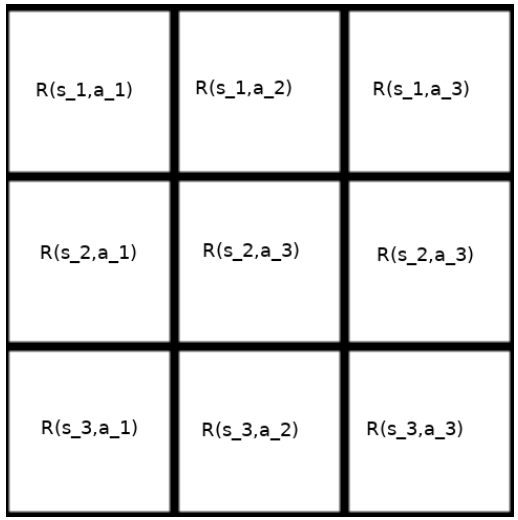
How to solve contextual bandit problem?

'Two-in-one problem'

1. *Associate context and expected rewards:*
   Set up specific model.
2. *Manage exploration:*
   As above ϵ-Dithering, Thompson Sampling, UCB1,...

Contextual Bandits: Algorithms

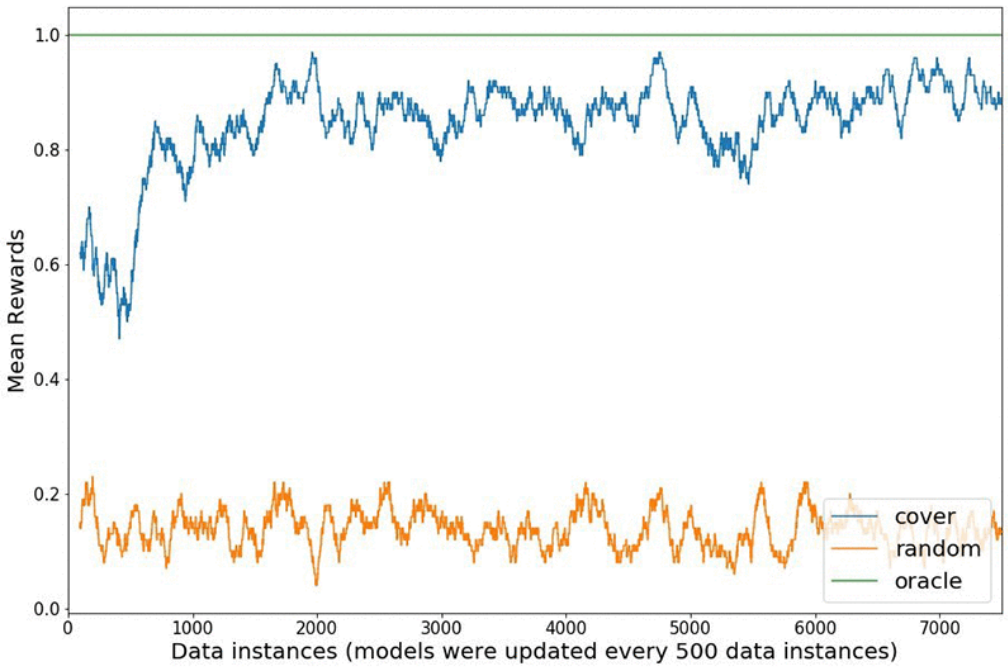
- *Finite number of contexts:*
  E.g. *action* $\in \{a_1, a_2, a_3\}$,
  context $\in \{s_1, s_2, s_3\}$.
  $\hookrightarrow$ View the 3-context,
  3-action bandit as 9-armed
  ordinary bandit.



- *Infinite 'number' of contexts:*
  - Set of mappings between infinite context set an expected rewards: 'Very very infinite'
    $\hookrightarrow$ impossible to learn without further structure. Add it:
  - Linear relationship:
    $\mathbb{E}[R(a_k)|s \in \mathbb{R}^D] = (\theta_k^*)^T \cdot s$. Learn params. θ_k^*
    1. Via estimator $\hat{\theta} \rightarrow$ estimator of exp. reward $\rightarrow$ context. UCB1
    2. Via Bayesian model $\rightarrow$ Posterior of exp. rew. $\rightarrow$ play a_k accord. to post. prob. it is optimal $\rightarrow$ context. Thomp. S.
  - Non-linear relationship RKHS: Natural generalization of 2.
  - Non-linear relationship Neural Net: later...

Add selection: Amazon

Rolling mean reward over 100 samples in Amazon add selection.



(Model received pre-training)

Markov Decision Processes (MDPs)

MDPs: Framework for modeling sequential decision making in uncertain environment

As a contextual bandit but now agent's decisions influence environment, which generates the new context.

Goal: *Maximize accumulated reward over planning horizon!*

Markov Decision Processes (MDPs)

MDPs: Framework for modeling sequential decision making in uncertain environment

As a contextual bandit but now agent's decisions influence environment, which generates the new context.

Goal: *Maximize accumulated reward over planning horizon!*

Chess:

- *Action:* Choose move with White
- *Environment:* Board from White's perspective (including Black's move)
- *Reward:* E.g. immediate material count

'Living for the moment' (PhD):

- *Environment:* State of the world
- *Action:* Go to a bar today or prepare lecture tomorrow

$\hookrightarrow$ MDP. lecturer prepares the lecture....

Markov Decision Processes: Formalization

- *MDP* is tuple $(S, A, P, R, T, \textit{MDP iteration})$ with
  1. $S = \{s_1, s_2, ..., s_n\}$ set of admissible states, assumed finite here
  2. $A = \{a_1, ..., a_k\}$ set of possible actions, assumed finite here
  3. $P(s', a, s)$ prob. of entering s' by a from s
  4. $R(s, a)$ reward obtained by taking $a \in A$ in $s \in S$
  5. $T \in \mathbb{N}$ number of rounds played, planning horizon

Markov Decision Processes: Formalization

- *MDP* is tuple $(S, A, P, R, T, \textit{MDP iteration})$ with
  1. $S = \{s_1, s_2, ..., s_n\}$ set of admissible states, assumed finite here
  2. $A = \{a_1, ..., a_k\}$ set of possible actions, assumed finite here
  3. $P(s', a, s)$ prob. of entering s' by a from s
  4. $R(s, a)$ reward obtained by taking $a \in A$ in $s \in S$
  5. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *MDP iteration*:
  1. *Agent* chooses a given s with prob. $\pi(a|s)$, π *policy*
  2. Agent receives $R(s, a)$
  3. System transitions to s' with prob. $P(s'|s, a)$

Markov Decision Processes: Formalization

- *MDP* is tuple $(S, A, P, R, T, \textit{MDP iteration})$ with
  1. $S = \{s_1, s_2, ..., s_n\}$ set of admissible states, assumed finite here
  2. $A = \{a_1, ..., a_k\}$ set of possible actions, assumed finite here
  3. $P(s', a, s)$ prob. of entering s' by a from s
  4. $R(s, a)$ reward obtained by taking $a \in A$ in $s \in S$
  5. $T \in \mathbb{N}$ number of rounds played, planning horizon
- *MDP iteration*:
  1. *Agent* chooses a given s with prob. $\pi(a|s)$, π *policy*
  2. Agent receives $R(s, a)$
  3. System transitions to s' with prob. $P(s'|s, a)$
- *MDP goal*: Find policy π^* that maximizes value of state s

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \middle| s_0 = s\right], \gamma \in (0, 1],$$

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

Key point about MDPs: Taking action to maximize immediate reward might lead to a state, from which no further reward is possible.

Sacrifice: *Less reward today for a better future!*

Solving MDPs I (that's a topic to fill books)

So how to act in an uncertain and non-forgiving world?

Bellman: Optimal policy solves

$$V^{\pi^*}(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi^*}(s') \right\}$$

$\rightarrow$ *in theory*: solve this and you're good

Algorithms for solving Bellman:

- *Naive:* Check all possibilities (many times, if P not deterministic) $\hookrightarrow$ very exponential, too much work
- *Moral:* Moral actions are approximation to optimal policy $\hookrightarrow$ not always available, hard to prove guarantees
- *Value iteration:* Iterative updates $\forall s \in S$:

$$\tilde{V}(s) \leftarrow \max_{a \in A(s)} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|a,s) \tilde{V}(s') \right\},$$

$A(s)$ actions from s, converge to V^{π^*} in $\mathcal{O}(poly(N)e^{-\lambda N})$
$\hookrightarrow$ Requires $P(s'|s,a)$ and if $S, A(s)$ large, too much work

Solving MDPs II (that's a topic to fill books)

Question 1: Can you simulate environment $P(s'|s,a)$?

No$\rightarrow$ learn from experience:

- *Monte Carlo:*
  1. Run width C, depth T Monte Carlo
  2. $\hat{V}_{new}(s) \leftarrow \hat{V}_{old}(s) + \alpha \left[\sum_t^T R_t - \hat{V}_{old}(s)\right]$, Step-size α

     weights relevance of new sample versus held estimate $\hat{V}_{old}$.
     Output: stochast. policy from MC But: C needs be very large, updates only after episodes complete $\hookrightarrow$ impractical

Solving MDPs II (that's a topic to fill books)

Question 1: Can you simulate environment $P(s'|s,a)$?

No$\rightarrow$ learn from experience:

- *Monte Carlo:*
  1. Run width C, depth T Monte Carlo
  2. $\hat{V}_{new}(s) \leftarrow \hat{V}_{old}(s) + \alpha \left[\sum_t^T R_t - \hat{V}_{old}(s)\right]$, Step-size α weights relevance of new sample versus held estimate $\hat{V}_{old}$. Output: stochast. policy from MC But: C needs be very large, updates only after episodes complete $\hookrightarrow$ impractical
- *Temporal-difference Monte Carlo:*
  1. Don't wait until episode finished, update after each new reward.
  2. $\hat{V}_{new}(s) \leftarrow \hat{V}_{old}(s) + \alpha \underbrace{\left[R_{t+1} + \gamma \hat{V}_{old}(s_{new}) - \hat{V}_{old}(s)\right]}_{TD\ error}$,

     Two types of estimates:
     a. Ordinary Monte Carlo estimate as above
     b. Bellmann estimate $R_{t+1} + \gamma \hat{V}_{old}(s_{new})$ for total reward of the episode $\sum_t^T R_t$.
  3. Iteration converges to optimal value function if step-size small

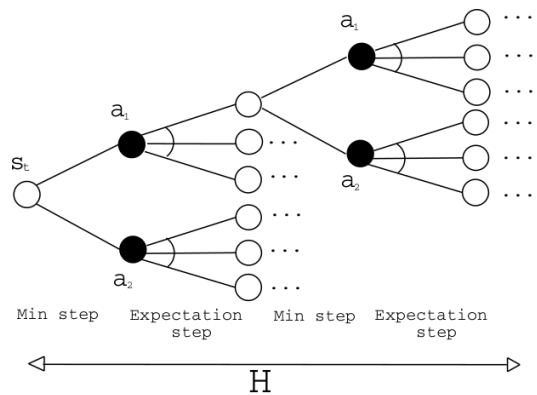
Solving MDPs III (that's a topic to fill books)

Question 1: Can you simulate environment $P(s'|s,a)$?

Yes$\rightarrow$ learn from simulation:

- *Look-ahead search:* Focus on relevant states.
  Starting from s 'envelope of states' is build. Iterate:
  1. Value 'ends of envelope' s' by approx. value function
  2. Update ancestors of s' using Bellman update

  But:
  - If $S, A(s)$ very large, too much work



Solving MDPs III (that's a topic to fill books)

Question 1: Can you simulate environment $P(s'|s,a)$?

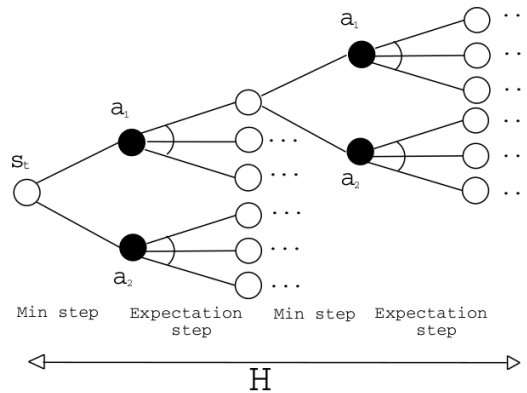
Yes$\rightarrow$ learn from simulation:

- *Look-ahead search:* Focus on relevant states.
  Starting from s 'envelope of states' is build. Iterate:
  1. Value 'ends of envelope' s' by approx. value function
  2. Update ancestors of s' using Bellman update

  But:
  - If $S, A(s)$ very large, too much work
- *Kearn's sparse sampling (Monte Carlo):*
  Do not assume $P(s'|s,a)$ but *simulator*
  1. Run width C, depth H Monte Carlo
  2. Update until depth H

  $$V_{h,C}(s) = \begin{cases} \tilde{V}(s) \text{ if } h = 0 \\ \max R(s,a) + \gamma \frac{1}{C} \sum_{s' \in S(C)} V_{h-1,C}(s') \end{cases}$$

  $S(C)$ contains C samples beginning with (s,a)



Output: stochast. policy from MC tree $\rightarrow \pi^*$ *indep.* of S

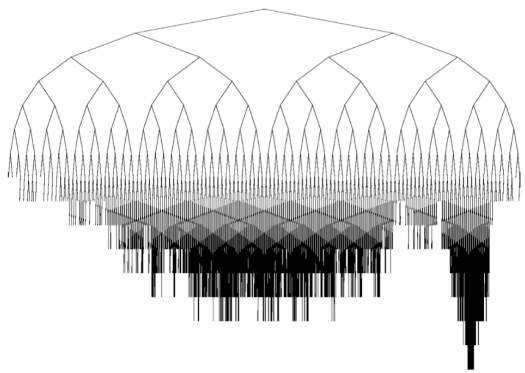
But: H, C need be very large $\hookrightarrow$ impractical

Towards Monte Carlo Tree Search (MCTS)

Key Idea: Build specific, restricted, asymmetric decision tree

- MCTS involves two policies:
  1. *Default policy*: Used to value tree leafs
  2. *Tree policy*: Selects or creates leaf nodes from given tree
- Tree policy's purpose:
  - Adding new nodes $\rightarrow$ Exploration
  - Simulation of promising line $\rightarrow$ Exploitation

} Balance this!



Towards Monte Carlo Tree Search (MCTS)

Key Idea: Build specific, restricted, asymmetric decision tree

- MCTS involves two policies:
  1. *Default policy*: Used to value tree leafs
  2. *Tree policy*: Selects or creates leaf nodes from given tree
- Tree policy's purpose:
  - Adding new nodes $\rightarrow$ Exploration
  - Simulation of promising line $\rightarrow$ Exploitation

  } Balance this!

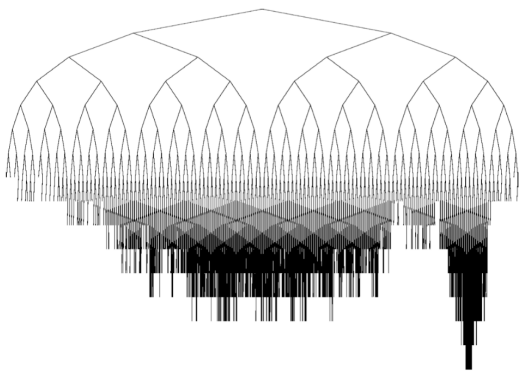
Tree-policy: Employ k-**armed bandit**!

- TS take actions according to their posterior prob. of being optimal
- $UCB1$ policy maximizes

  $UCB1_a = \bar{R}_a + c_{n,n_a}$ with $c_{n,l} = \sqrt{\frac{2 \ln n}{l}}$

  $\bar{R}_a$ average realized reward,

  n_a number of chosen a
- TS, $UCB1$ resolves exploration management, optimal regret $\mathcal{O}(log(n))$



Monte Carlo Tree Search (MCTS)

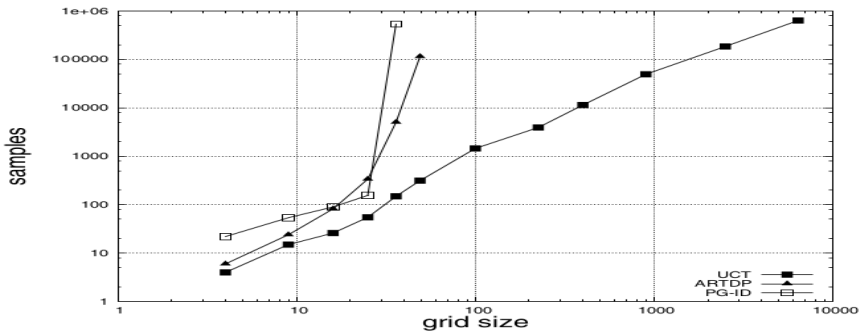
MCTS: Iterative improvement of tree policy and Bellman updates

1. *Selection:* Apply tree policy recursively to descend through tree until most relevant node
2. *Expansion:* Child node added to tree (according to tree policy)
3. *Simulation:* From new node default policy is applied to simulate reward
4. *Backpropagation:* Simulation result used to update statistics of ancestor nodes through the tree (count in $UCB1_i$)

```
1.  function MonteCarloPlanning(state)
2.  repeat(computationBudget): search(state,0)
3.  return bestAction(state,0)
4.
5.  function search(state,depth)
6.  if Terminal(state): return 0
7.  if Leaf(state): return Evaluate(state)
8.  action = selectAction(state,depth)
9.  (nextState, reward) = simulateAction(state,action)
10. q = reward + γ search(nextstate,depth+1)
11. update(state,action,q,depth)
12. return q
```

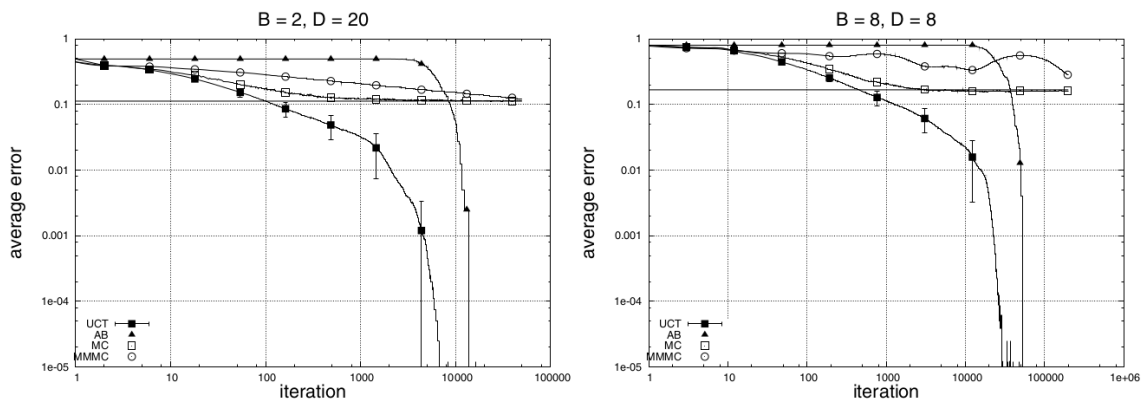
MCTS Examples

1. Simulator given ad hoc $\rightarrow$ MDP.
   Example: Find shortest path on map with stochastic Markovian friction (Sailing domain,...).



Number of samples to achieve error < 0.1

2. Simulator reflects 'worst case' $\rightarrow$ adversarial game, two MCTS instances compete, each is the simulator for the other
   Example: *P*-games (Chess, Go, ...)



Failure rate in *P*-game

MCTS Characteristics and Extensions

Characteristics:

1. *Asymptotic:* Converges to best decision if enough resources (Equiv. MinMax in P-games)
2. *Aheuristic:* No domain-specific knowledge, no eval. function (Chess: heuristics are available, $\alpha\beta$ very strong. Go: not)
3. *Anytime:* Tree statistics are update immediately. Small error prob. if stopped prematurely ($\rightarrow$ careful 'trap states' in Chess)

MCTS Characteristics and Extensions

Characteristics:

1. *Asymptotic:* Converges to best decision if enough resources (Equiv. MinMax in P-games)
2. *Aheuristic:* No domain-specific knowledge, no eval. function (Chess: heuristics are available, $\alpha\beta$ very strong. Go: not)
3. *Anytime:* Tree statistics are update immediately. Small error prob. if stopped prematurely ($\rightarrow$ careful 'trap states' in Chess)

Extensions: (exist in any direction!)

1. Tree policy:
   a) *Bandits:* UCB-Tuned, Bayesian Bandit, Continuous actions,...
   b) *Selection:* Domain Specific, Large Branching Issues (FPU), add pre-search, transposition Tables, history heuristics,...
   c) *Other*: Proof-number search, Pruning, Policy representation (NN),...
2. Simulation: Rule-based policy, Policy representation (NN)...
3. Back-propag.: Weighted rewards,...

Neural Network-based algorithms

To be added!