# Markov Decision Processes

## Online Algorithms

CS4246/CS5446

AI Planning and Decision Making

Sem 1, AY2021-22

# Topics

- Online algorithms for solving MDPs (RN17.2.4)
  - Monte Carlo Tree Search (RN5.4, SB 8.10, 8.11)

# Markov Decision Process (MDP)

- Formally:
  - An MDP $M \triangleq (S, A, T, R)$ consists of
  - A set $S$ of states
  - A set $A$ of actions
  - A transition function $T: S \times A \times S \rightarrow [0,1]$ such that:

$$\forall s \in S, \forall a \in A: \sum_{s\prime \in S} T(s, a, s') = \sum_{s\prime \in S} P(s'|s, a) = 1$$

  - A reward function $R: S \rightarrow \Re$
  - Solution is a policy – a function to recommend an action in each state: $\pi: S \rightarrow A$
    - Solution involves careful balancing of risk and reward

# Handling Curse of Dimensionality

- For large problems:
  - State space grows exponentially with number of variables
  - Value iteration and policy iteration iterate through all states
    - Exponential with number of variables
- New solution approaches:
  - To do online search with sampling – decision-time planning
    - Real-time dynamic programming
    - Monte Carlo Tree Search
  - To use function approximation of the utility function – compact representation
    - Linear function of features
    - Deep neural networks
    - Etc.

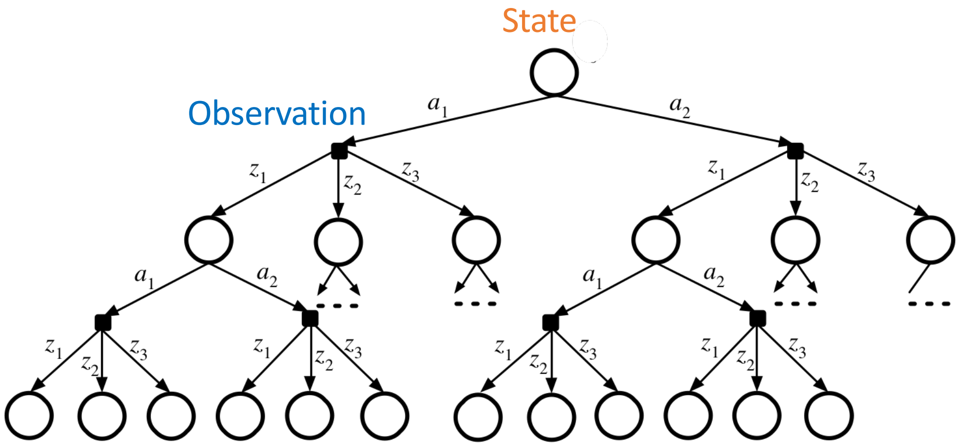
# Online Search

Decision-time planning

# Online Algorithms

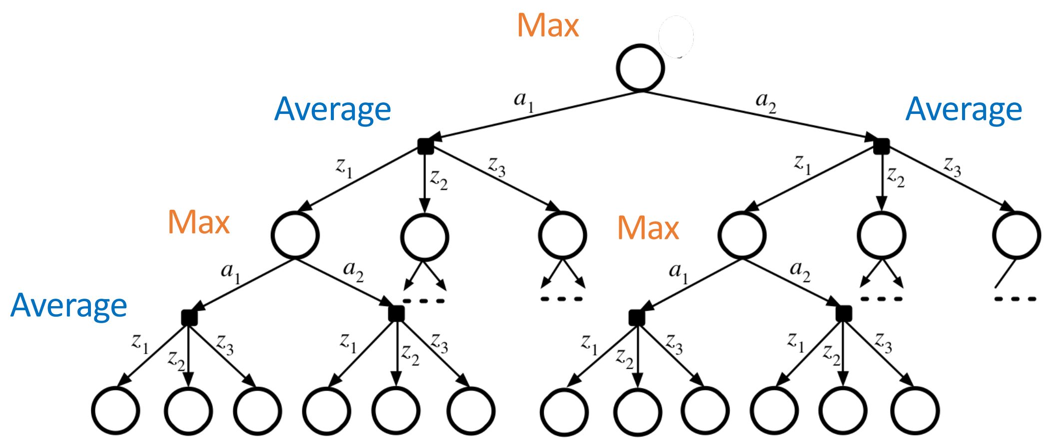
- Approach: Decision-time planning
  - Significant amount of computation at each decision point, rather then operating primarily with precomputed information
- Methods
  - Real-time dynamic programming
    - Good for mid-size problems.
    - State space with very few repeated states for any manageable set of explored states
    - Simple heuristic for frontier nodes may not be enough to guide well,  if rewards are sparse
    - Apply reinforcement learning to generate more accurate heuristics
  - Monte Carlo Tree Search – To look further ahead in the MDP
    - UCT Algorithm for MDP – better suited for large domains, where payoffs go far enough into the future to assess risky potential move

# Online Search



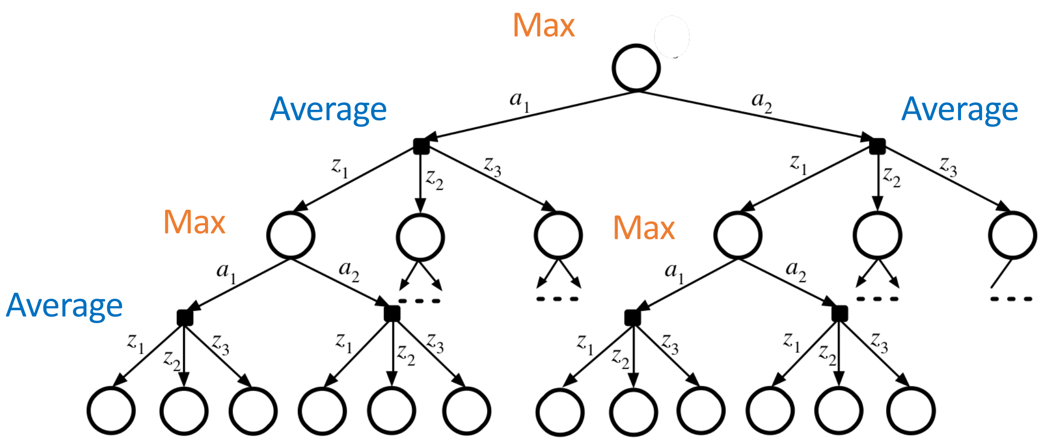

- At every step, construct a search tree.
  - Up to fixed depth $D$.
  - Root is current state.
  - $|A|$ actions – children of root (and other state nodes).
  - $|Z| = |S|$ children of observation nodes (next state nodes)

# Online Search




- To compute value at the root:
  - Initialize leaf with utility estimates (or zeros).
  - At observation nodes, compute expected utilities of the children
  - At state nodes (where actions can be taken), compute max of the children.

# Sparse Sampling




- Approach:
  - Tree size is $|A^D||S^D|$.
  - With sparse sampling* [1], estimate by sampling $k$ observations at observation nodes, instead of using all $|S|$ states as possible observations.
  - Tree size $|A^D||k^D|$
  - Question: Have we solved the curse of dimensionality?

*Michael Kearns, Yishay Mansour, and Andrew Y Ng. "A sparse sampling algorithm for near-optimal planning in large Markov decision processes". In: Machine learning 49.2-3 (2002), pp. 193-208.

# Rollout

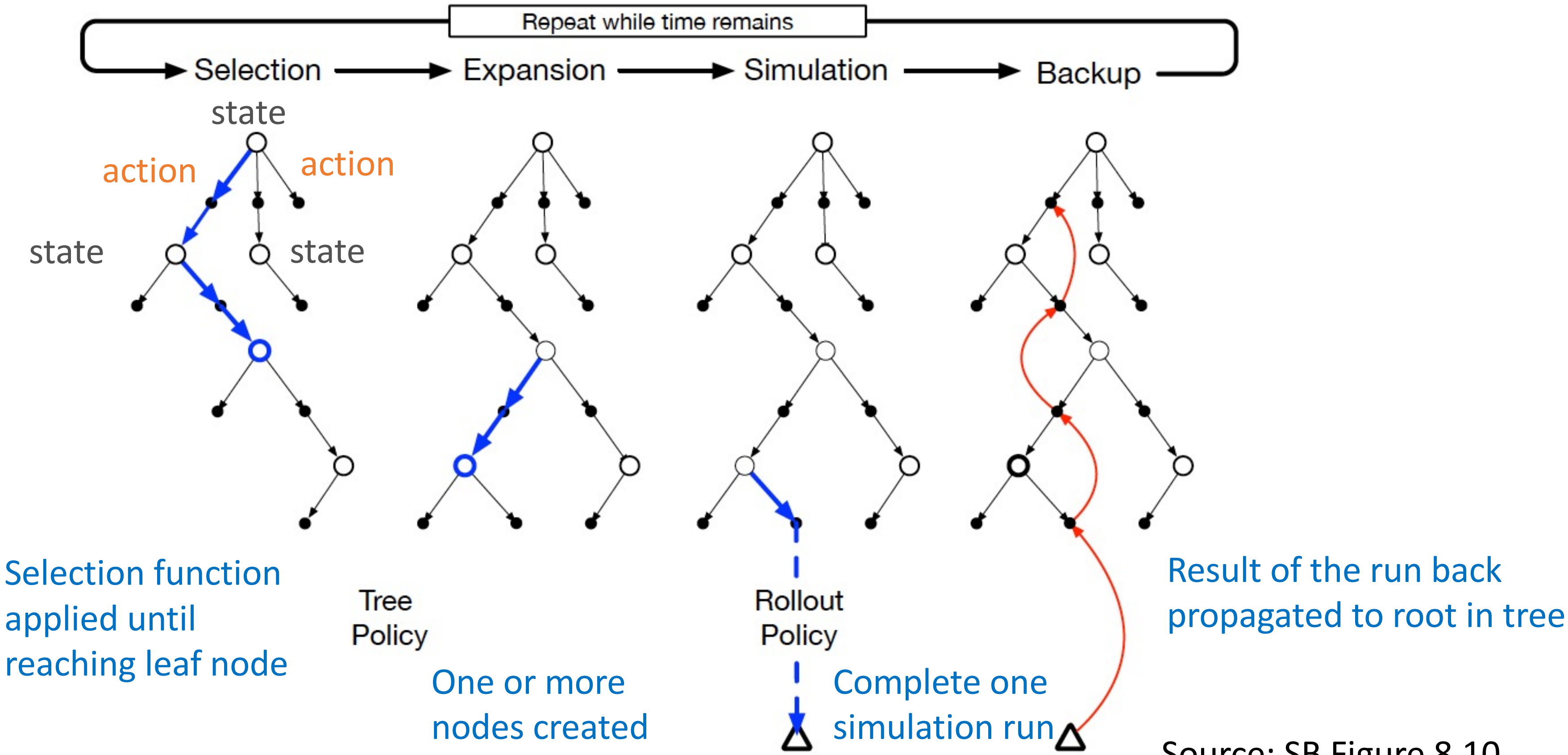
- Assume:
  - You already have a policy $\pi$.
- In rollout, start at state $s$, try to obtain a policy better than $\pi$ by:
  - Estimate Q-function $Q(s, a)$ at $s$ by simulating many trajectories from $s$ using each action $a$ where the simulations are done using $\pi$.
  - Select action that has the highest average return.
- Improvement:
  - If estimates are accurate enough, the policy improvement theorem implies that rollout improves on $\pi$

# Monte Carlo Tree Search

Online search with simulation

Source: Lee WS, Lecture notes, 2020, MDP p39 -54

# Monte Carlo Tree Search




Source: SB Figure 8.10

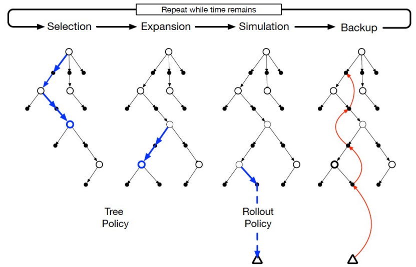
# Monte Carlo Tree Search Algorithm

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
   tree ← NODE(state)
   while IS-TIME-REMAINING() do
      leaf ← SELECT(tree)
      child ← EXPAND(leaf)
      result ← SIMULATE(child)
      BACK-PROPAGATE(result, child)
   return the move in ACTIONS(state) whose node has highest number of playouts
```

**Figure 5.11** The Monte Carlo tree search algorithm. A game tree, *tree*, is initialized, and then we repeat a cycle of SELECT / EXPAND / SIMULATE / BACK-PROPAGATE until we run out of time, and return the move that led to the node with the highest number of playouts.

Source: RN Figure 5.11

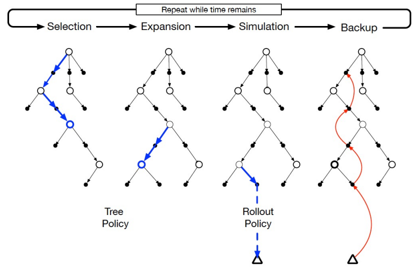
# Monte Carlo Tree Search



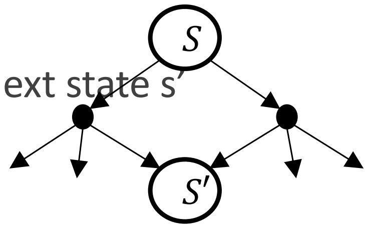

- Commonly used in MDPs and games*
- Uses both tree search and rollout
  - Rollout at leaf of tree instead, selectively expand tree
  - Repeatedly run trials from the root (current state in online search)
- Trial
  - Repeatedly select node to go to at next level until target depth reached, or
  - Selected node has not been discovered
  - Expansion – create a new node, run a simulation using a rollout policy till required depth
  - Back up the outcomes all the way to the root.
- Anytime policy:
  - When time is up, use the action that looks best at the root at that time.

*Guillaume Chaslot et al. "Monte-Carlo Tree Search: A New Framework for Game AI". In: AIIDE. 2008.

# MCTS in MDP




- For an MDP:
  - A tree (actually DAG) node n is associated with a state $s$.
  - A node $n$' at the next level is selected by applying an action $a$ to $s$, then sampling the next state s' (corresponding to $n$') according to $p(s'|s, a)$




- Action selection
  - The action a is selected by balancing exploitation with exploration with exploitation
- Estimated utility:
  - $\widehat{U}(n)$at a node $n$ is the average return of all the trials at $n$.
  - The return $r_t\ (n)$ of trial $t$ starting from $n$ with state $s$ and next node $n$' is $R(s)\ +\ \ \gamma r_t(n')$.
- Estimated Q-function (action-value function)
  - Estimated Q-function at n, $\hat{Q}(n, a)$ is the average return of all trials at $n$ that starts with action $a$.
  - $\hat{Q}(r, a)$ at root $r$ used to select the action to take at the root.
- All these are updated in the back-up operation to the root.

# Upper Confidence Bounds applied to Trees (UCT)

- Selection policy of UCT[1] [2] algorithm at node $n$:

$$\pi_{UCT}(n) = \underset{a}{\operatorname{argmax}}(\hat{Q}(n,a) + c\sqrt{\frac{\ln(N(n))}{N(n,a)}}$$

- Where:
  - $\hat{Q}(n,a)$ is the average return of all trials at $n$ that starts with action $a$
  - $N(n,\mathrm{a})$ is the number of trials through node $n$ that starts with action $a$
  - $N(n)$ is the number of trials through node $n$
- Exploitation term: $\hat{Q}(n,a)$– average utility of $n$ that starts with action $a$
- Exploration term: Square-rooted term with count $N(n,\mathrm{a})$ in the denominator
  - Will be high for nodes that have only been explored a few times
  - Will go to zero as the counts increase if $(n,\mathrm{a})$ is selected some non-zero percentage of time
  - Eventually playouts given to the node with the highest utility
- $c$ – Constant balancing exploitation and exploration – often tuned to do well on the problem

[1]Levente Kocsis and Csaba Szepesvari. “Bandit based monte-carlo planning". In: European conference on machine learning. Springer. 2006, pp. 282-293.

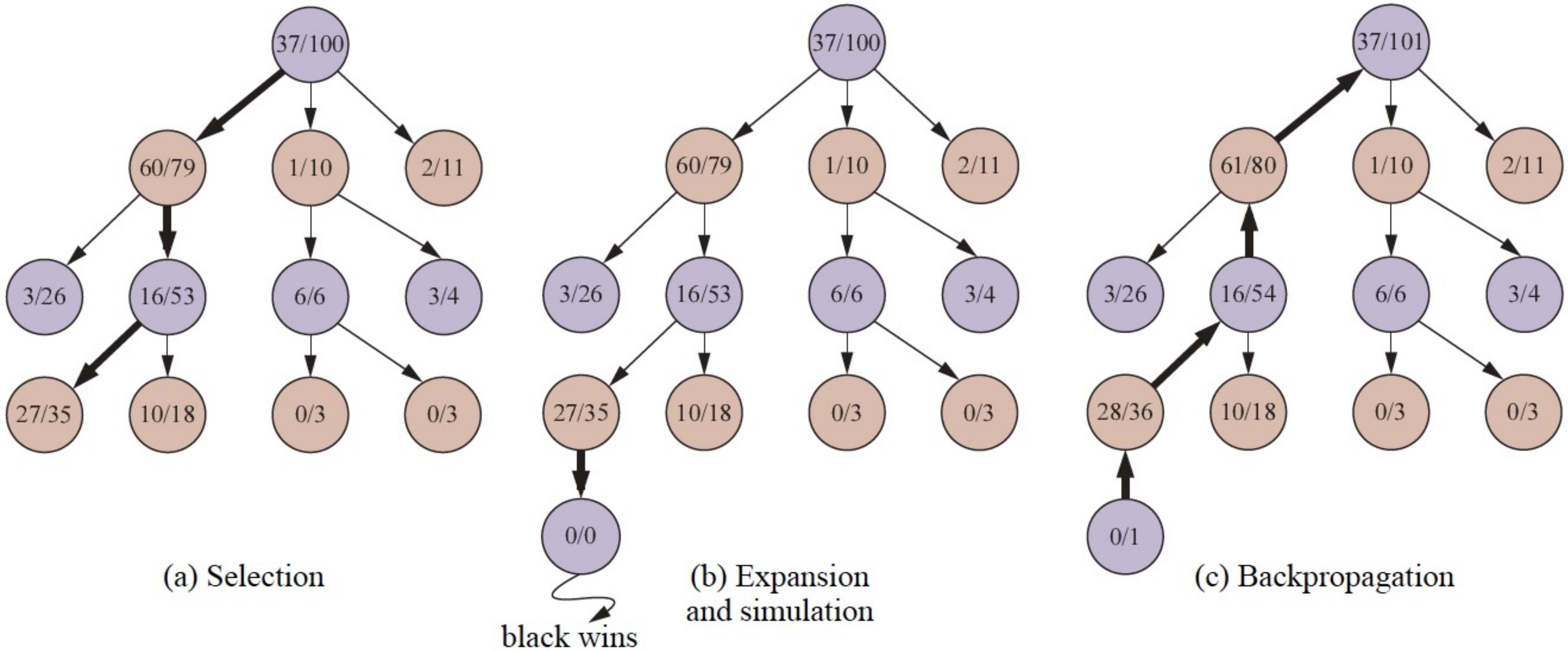
# Upper Confidence Bounds applied to Trees (UCT)

- Observations:
  - UCT will eventually converge to the optimal policy with enough trials
  - Worst case can be very bad[1] : $\Omega(\overbrace{\exp(\exp(\ldots\exp(1)\ldots))}^{D-1\ times})$
  - Often works well in practice.
    - PROST Planner[2] won the ICAPS International Probabilistic Planning Competition for MDP in 2011 and 2014 uses UCT.

[1]Coquelin, P.-A. and R. Munos, *Bandit algorithms for tree search*, in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*. 2007, AUAI Press: Vancouver, BC, Canada. p. 67–74.
[2]https://github.com/prost-planner/prost

# Example: Monte Carlo Tree Search in Game

Using upper confidence bounds applied to trees (UCT) selection metric




(a) Selection

(b) Expansion and simulation

(c) Backpropagation

Source: RN Figure 5.10

# MCTS in practice

Player 2 uses MCTS

- Visualizing MCTS:
  - https://www.youtube.com/watch?v=FvRSxNLTg7U&ab_channel=DaveDyer
- Playing Super Mario:
  - https://www.youtube.com/watch?v=HRiEUUC9TUA&ab_channel=Emil
- AlphaGo Zero
  - https://youtu.be/tXlM99xPQC8
  - Toward breakthrough in AI

# AlphaGo Zero

https://youtu.be/tXlM99xPQC8

David Silver et al. Mastering the game of Go without human knowledge". In: Nature 550.7676 (2017), p. 354.

# AlphaGo Zero

- Go Game
  - Space size of about $10^{170}$ with branching factor that starts at 361
  - Difficult to define good evaluation function
  - Need function approximation to represent value and policy functions
- AlphaGo Zero* [3] uses combination of MCTS and policy iteration
- For playout policy:
  - AlphaGo Lee (which defeated Lee Sedol) used a combination of expert games as well as self play.
  - AlphaGo Zero uses only self-play (only provided with the rules of Go); defeated AlphaGo Lee 100-0.
- Use deep neural network with two "heads"
  - Value head – outputs real value estimate of the value function
  - Policy head – outputs vector of size $190 \times 190$
    - Each component represents the probability that the policy will play that board position.

*David Silver et al. Mastering the game of Go without human knowledge". In: Nature 550.7676 (2017), p. 354.

# AlphaGo Zero

- For (action) selection policy:
  - Variant of UCT that exploits policy head output of neural network $P(s, a)$

$$\pi_{UCT}(s) = \operatorname*{argmax}_a \hat{Q}(s,a) + cP(s,a)\sqrt{\frac{\sum_b N(s,b)}{1+N(s,a)}}$$

  - When a leaf node is reached, the value head of the neural network is used to evaluate the state instead of doing a roll-out (simulation).
- Go is a zero-sum turn taking game instead of an MDP:
  - Search alternates between:
    - Selecting action that maximizes when it is first player's turn
    - Selecting action that minimizes (multiply estimate by $-1$ then maximize) for second player’s turn
  - At termination: reward $+1$ for first player win and $-1$ for second player win.

# AlphaGo Zero

- Approximate policy iteration with self-play
  - Policy iteration has 2 stages: policy evaluation and policy improvement
  - AlphaGo Zero does both using supervised learning
  - With current value function, MCTS viewed as policy improvement operator gives improved policy values for evaluated states
    - Improve policy for a set of states
  - Self-play with search gives the policy evaluation for the evaluated states
    - Evaluate value of improved policy
  - Supervised learning is used to interpolate values and policy over whole domain using data from a set of states.

# Homework

- Readings
  - [RN] 17.2.4, 5.4 (Online algorithms, MCTS)
  - [SB] 8.10, 8.11 (Online algorithms, MCTS)

  - [SB] Sutton, R. S. and A. G. Barto. Reinforcement Learning: An introduction. 2nd ed. MIT Press, 2018, 2020
    [Book website: http://incompleteideas.net/book/the-book.html ]
    [e-Book for personal use:
    http://incompleteideas.net/book/RLbook2020.pdf ]

# References


- Sparse Sampling:
    1. Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes". In: Machine learning 49.2-3 (2002), pp. 193–208.
- The UCT algorithm:
    2. Kocsis, L. and C. Szepesvári, Bandit based Monte-Carlo planning, in Proceedings of the 17th European conference on Machine Learning. 2006, Springer-Verlag: Berlin, Germany. p. 282–293.
- AlphaGo Zero:
    3. David Silver et al. Mastering the game of Go without human knowledge". In: Nature 550.7676 (2017), p. 354.