L07: Decision-Theoretic Planning

Planning Algorithms in AI

Prof. Gonzalo Ferrer

Skoltech, 2 December 2022

Summary L07, Russell 5.1-5.4+extra readings

- Single decision making
- A plan against nature
- Nondeterministic vs probabilistic
- Sequential decision making
- Minimax Search Trees
- Alpha-beta pruning
- Monte Carlo tree search

A plan (game) against nature

- Now, let's consider two Decision-Makers:
  - **Robot**: so far, what we have studied in previous lectures.
  - **Nature**: This DM is mysterious, unpredictable and unknown for the robot. In general, this agent will interfere with our main DM (robot) to achieve our goals, and it is a way to model <u>uncertainty</u> into planning.

Problem definition 1: Simultaneous actions

1) A nonempty set, the (robot) **action space** $u \in U$
2) A nonempty set, the **nature action space** $\theta \in \Theta$
3) A **reward function** $R : U \times \Theta \to \mathbb{R}$

   Note: the robot now seeks to maximize this reward.

Example: suppose there are three actions for robot and nature:

$$
\begin{array}{cc}
 & \Theta \\
U & \begin{array}{|c|c|c|} \hline 1 & -1 & 0 \\ \hline -1 & 2 & -2 \\ \hline 2 & -1 & 1 \\ \hline \end{array}
\end{array}
$$

Problem definition 2: Nature Knows

1) A nonempty set, the (robot) **action space** $u \in U$

2) For each action *u*, the **nature action space** $\theta \in \Theta(u)$

3) A reward function $R : U \times \Theta \rightarrow \mathbb{R}$

Example 2: Now, the robot chooses an action and then nature chooses:

$$\begin{array}{cc} & \Theta \\ U & \begin{array}{|c|c|c|} \hline 1 & -1 & 0 \\ \hline -1 & 2 & -2 \\ \hline 2 & -1 & 1 \\ \hline \end{array} \end{array}$$

Nondeterministic vs probabilistic models

What is the best decision for the robot, now that it is engaged in a game against nature?

It depends on what information the robot has regarding how nature chooses its actions.

Then, two possible models can be used by the robot to model nature:

Nondeterministic: No idea of what nature will do.

Probabilistic: We have observed nature and gathered statistics.

Nondeterministic model

- Assuming that we know nothing, usually a safe assumption is taking the **worst-case**.
- Nature can be imaged as an adversary.
- Murphy's Law: "if anything can go wrong, it will".
- Known as ***minimax*** or *worst-case*, then the best reward can be calculated. In this case, the robot plays first and nature:

$$R^* = \min_{\theta \in \Theta} \left\{ \max_{u \in U} R(u, \theta) \right\}$$

- The example could be reversed to ***maximin*** and nature plays first and the the robot:

$$R^* = \max_{u \in U} \left\{ \min_{\theta \in \Theta} R(u, \theta) \right\}$$

Probabilistic model

- It is assumed that there is enough data to reliably estimate *P(θ)*
- On this case, it is imagined that natures applies a randomized strategy, where the future action will preserve this PDF.
- Also known as *expected-case analysis* or ***expectimax***

$$u^* = \arg\max_{u \in U} \left\{ \mathbb{E}_\theta \left\{ R(u, \theta) \right\} \right\}$$

$$\mathbb{E}_\theta \left\{ R(u, \theta) \right\} = \sum_{\theta \in \Theta} R(u, \theta) P(\theta)$$

Example nondeterministic vs probabilistic

$$\begin{array}{cc} & \Theta \\ U & \begin{array}{|c|c|c|} \hline 1 & -1 & 0 \\ \hline -1 & 2 & -2 \\ \hline 2 & -1 & 1 \\ \hline \end{array} \end{array}$$

Case nondeterministic:
(minimax)

$$\min_{\theta} \max_{u} R(u, \theta)$$

$$u^* = u_1 \quad (= u_3) \quad , \quad R^*(u^*, \theta) = -1$$

Probabilistic:

$$P(\theta) = 1/3$$

$$\max_u \{ E_\theta[R(u,\theta)] \} = \max_u \left\{ \begin{array}{l} E_\theta[R(u_1,\theta)] \\ E_\theta[R(u_2,\theta)] \\ E_\theta[R(u_3,\theta)] \end{array} \right\} = \max_u \left\{ \begin{array}{c} 0 \\ -1/3 \\ \boxed{2/3} \end{array} \right\}$$

Other models for nature

- Why decision should be worst case or simply expected cases?
- The **worst-case** indicates a perfect rationale behind of what nature (or other agent wants). The **ideal rational agent**.
- The **probabilistic** case assumes that the expected value is correct. This can lead to some issues, such as high cost of unlikely events (collision on robot navigation).
- Is there something in between?
  - Utility functions, Risk functions
  - **Noisy Rational**. It assumes a rational motivation from nature on decision making, but that can be "noisy"-> Sometimes we don't know what we want perfectly.

Regret

- Reward or cost is not the only way to make decisions.
- Regret is the “feeling” you get after doing a bad decision and wishing you could change it.
- More formally, regret is the cost that you could have saved by picking a different action, given the nature action that was applied.

$$T(u,\theta) = \max_{u' \in U}\{R(u',\theta)\} - R(u,\theta)$$

- We can calculate the optimal actions just by substituting cost by regret.

Making use of observations

- Suppose there is a sensor providing information before making decisions. Imagine this is having some information of what nature will do.
- Then, these observations condition the distribution of nature actions:

$$R^* = \min_{\theta \in \Theta(z)} \left\{ \max_{u \in U} \{R(u, \theta)\} \right\}$$

$$u^* = \pi^*(z) = \arg\max_{u \in U} \{\mathbb{E}_\theta \{R(u, \theta) \mid z\}\} = \arg\max_{u \in U} \left\{ \sum_{\theta \in \Theta} R(u, \theta) P(\theta | z) \right\}$$

Example of Decision Making: Classification

- An alternative interpretation of planning as decision making is the classification problem.
- We observe an image and the problem is to classify it. Although a little far fetched, classification could be considered a decision-making problem (and thus a planning problem) given an observation. What is nature doing?

U = {'Cat', 'dog'}

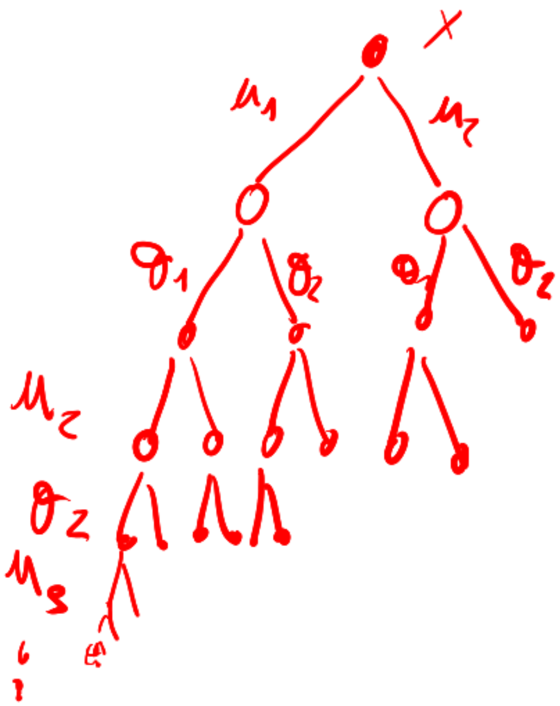
Sequential Decision Making

- Decision making so far was considered for a single action (length-1 plan)
- We will consider now a sequence of actions from the robot and nature. This also includes games.
- This problem can be well expressed as a search tree.

$$\tilde{\theta} = \{\theta_1, \ldots, \theta_k\}$$

$$\tilde{u} = \{u_1, \ldots, u_k\}$$

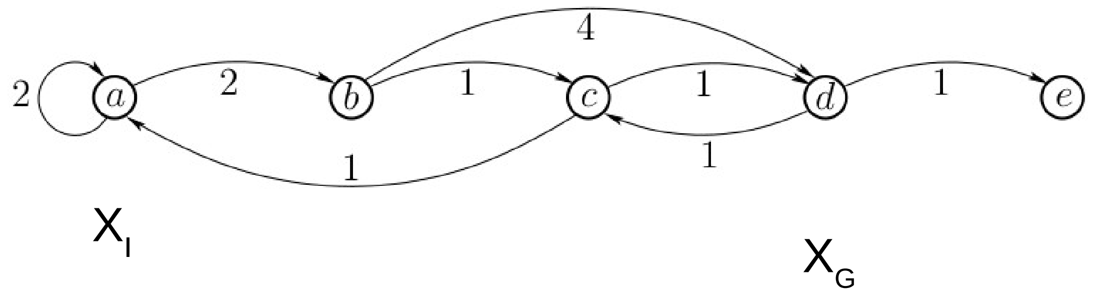
$$\min_{\theta_1} \max_{u_1} \ldots \min_{\theta_k} \max_{u_k} \left\{ R(\tilde{u}, \tilde{\theta}) \right\}$$

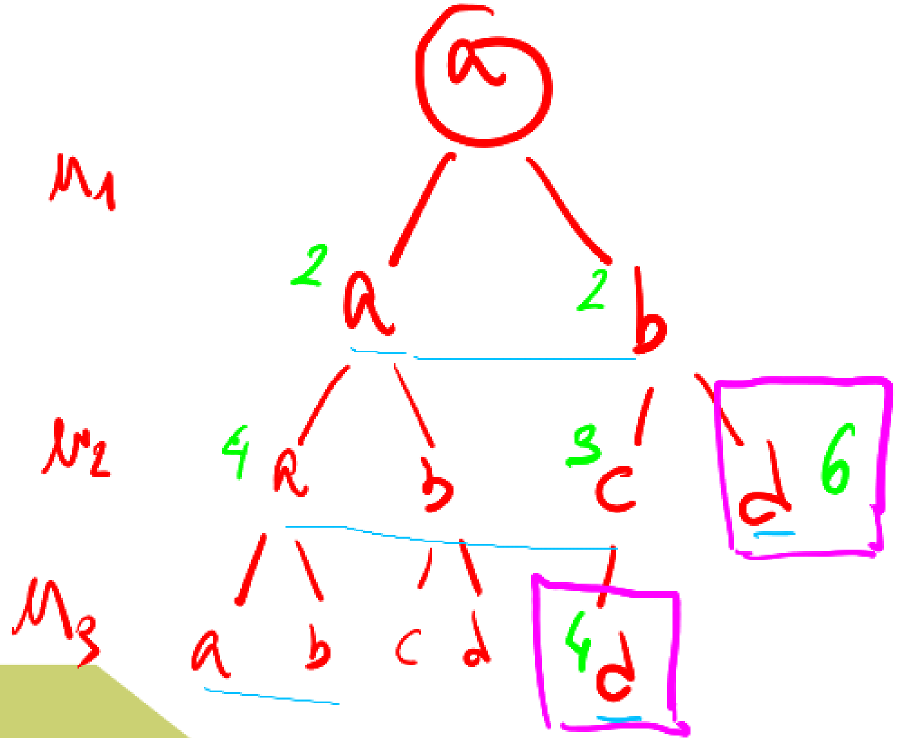


Resultant search tree

Graph Search and Search Trees

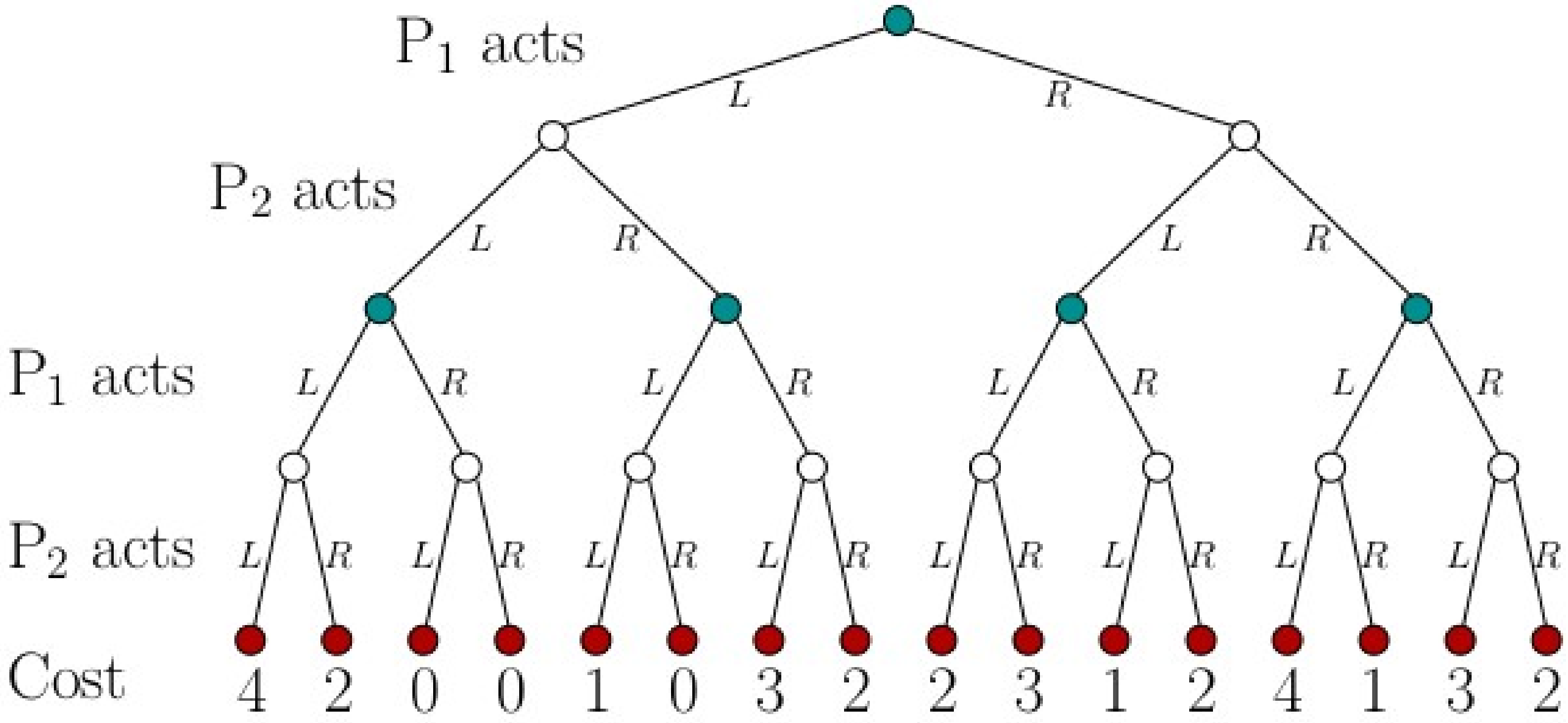
- From L02, let's recall the relation with graphs and trees.
- we built the graph's edges as we were searching.





Graph Search and Search Trees

- Now, games can be explained well with trees:



Sequential Decision Making

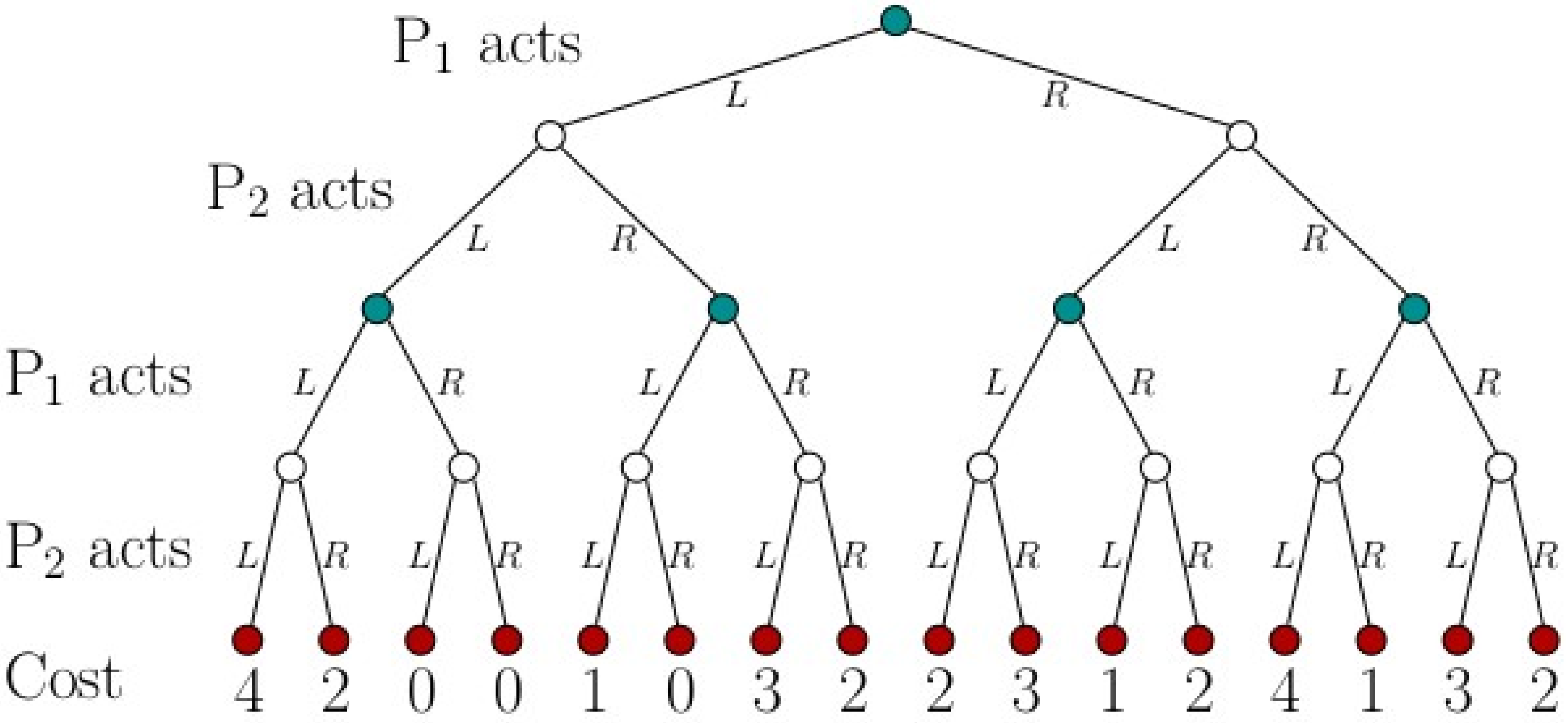
- We will discuss two important techniques to analyze the sequential DM
- The complexity grows exponentially with the depth, making the problem quickly intractable: $O(b^d)$, where b number of legal actions, d – depth.
- For minimax plans, we propose to study the **alpha-beta pruning**.
- For probabilistic (expectimax) plans, we will study the **Monte-Carlo tree search**.
  - In next lecture (L08), we will continue the probabilistic approach, with some extra considerations that will make this problem more tractable, namely Dynamic Programming.

Alpha-Beta pruning, introduction

- For small state spaces and sequences, we can create a tree and explore it exhaustively.
- However, it is unfeasible for larger state spaces or larger plans.
- Solution: Let's be more selective on how we explore, after all, there is no need to evaluate all leaves to obtain the solution to the *minimax* problem.
- The analogy with L02 was the queue of ranked the nodes to explore only those that are promising to improve the cost.
- It belongs to a class of branch and bound algorithms.

Alpha-Beta pruning

- Example of a tree:



Alpha-Beta pruning

- Briefly, the algorithm behaves as follows:
- The alpha parameter controls the *max* value for the robot action *u*.
- The beta parameter controls the *min* value for nature.
- If these values are exceeded, then the branch is cut since there is no need to keep exploring.
- Ideally, the number of nodes to be explored is reduced $O(b^{d/2})$. In other words, we can explore in double depth as compared with exhaustive *minimax*..

Monte-Carlo tree search (MCTS)

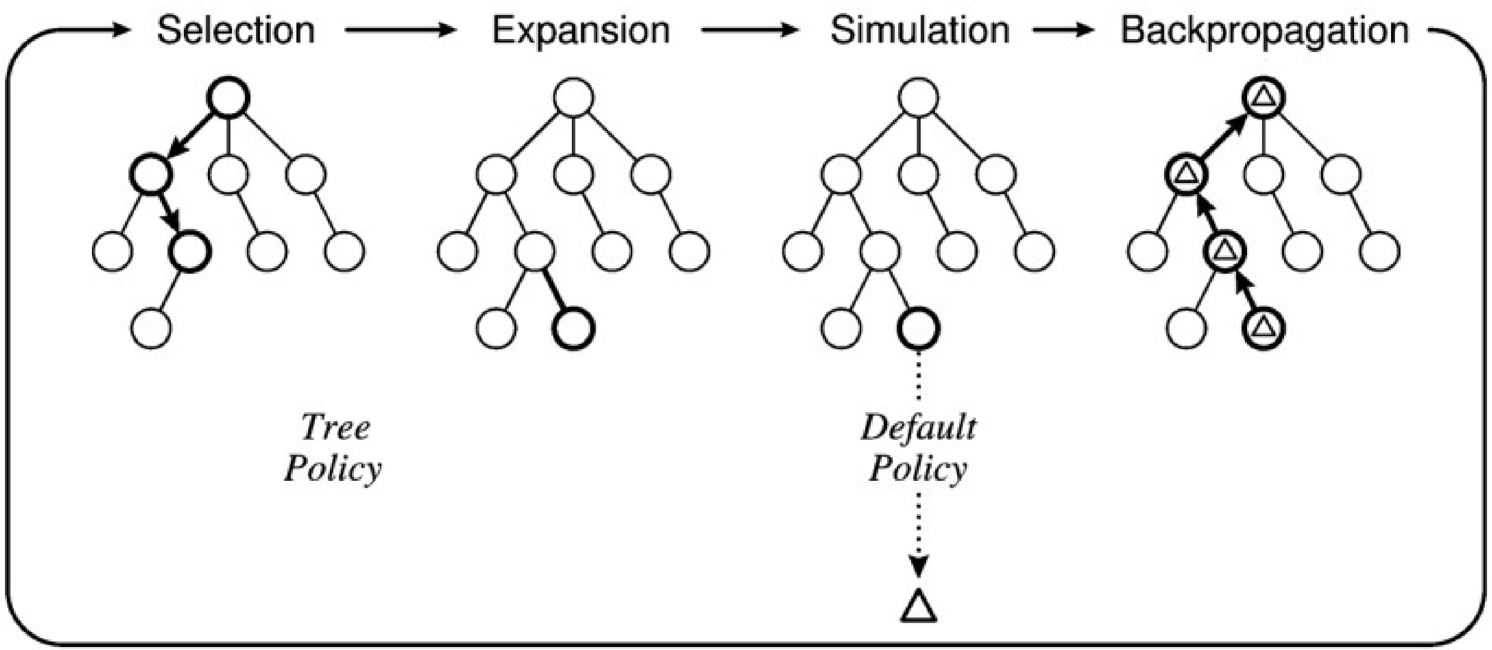
- MCTS is a search method that combines the precision of tree search with the generality of random sampling.
- It is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results
  - It is an anytime algorithm that provides small error probability.
  - Convergence to the best action (*minimax* solution).
- Typically used in planning problems or in sequential games, such as Go.

Monte-Carlo tree search (MCTS)

- **Selection**: the most promising node is selected.
- **Expansion**: A child of the selected node is expanded. If all nodes have been visited or it is terminal state, then this step is skipped
- **Simulation**: A simulation is run from the new node by using the default policy.
- **Backup**: The simulation result is backpropagated “up” though all nodes to update their statistics.

Monte-Carlo tree search (Simulation)

- The **simulation policy** is the result of combining the **tree policy** (which improves as we sample) and the **default policy**.



Monte-Carlo tree search (Evaluation)

- Simulate *N* episodes from the current state x_t using current simulation policy
- Build a search tree containing visited states and actions.
- Evaluate states *Q(x,u)* by **mean return** of episodes from *x, u*:

$$Q(x,u) = \frac{1}{N(x,u)} \sum_{i=1}^{N(x)} \mathbf{1}(x,u) z_i$$

  - *N(x,u)* is the number of times action *u* has been selected from *x*.
  - *N(x)* number of time a game has been played out through state *x*.
  - *z* is the result of the *i*th simulation played from *x*
  - **1**(.) equals 1 if action *u* was selected from state *x* from the *i*th simulation or 0 otherwise.

Monte-Carlo tree search (Evaluation)

- Simulate *N* episodes from the current state x_t using current *simulation policy*
- Build a search tree containing visited states and actions.
- Evaluate states *Q(x,u)* by **mean return** of episodes from *x, u*.
- After search is finished, select the current (real) action with maximum value in the search tree:

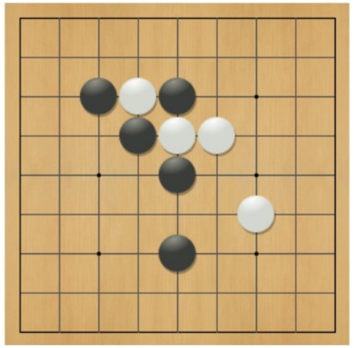
$$u_t = \arg\max_{u \in U} Q(x_t, u)$$

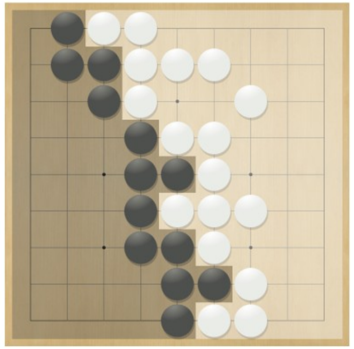
- Converges on the optimal search tree with enough samples:

$$Q(x_t, u) \to Q^*(x_t, u)$$

History of success: MCTS and the Game of Go

- In practice the algorithm has shown great success in games such as Go, beating the best human player in the world in 2016.
- This was considered the hardest classic board game.





Material from Silver D. (2015) “Lectures on Reinforcement Learning”
Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). “Mastering the game of Go with deep neural networks and tree search.” Nature.

MCTS and the Game of Go

- How good is a position *x*?
- Reward function:

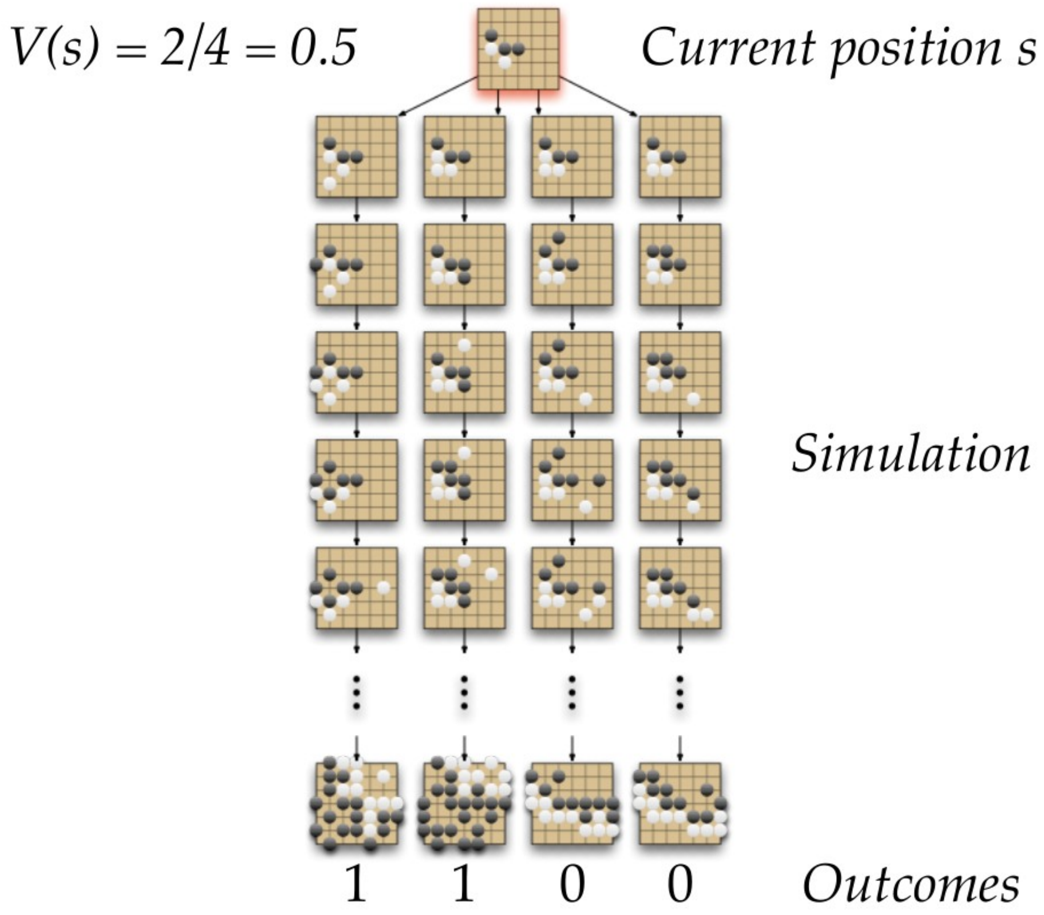
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

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

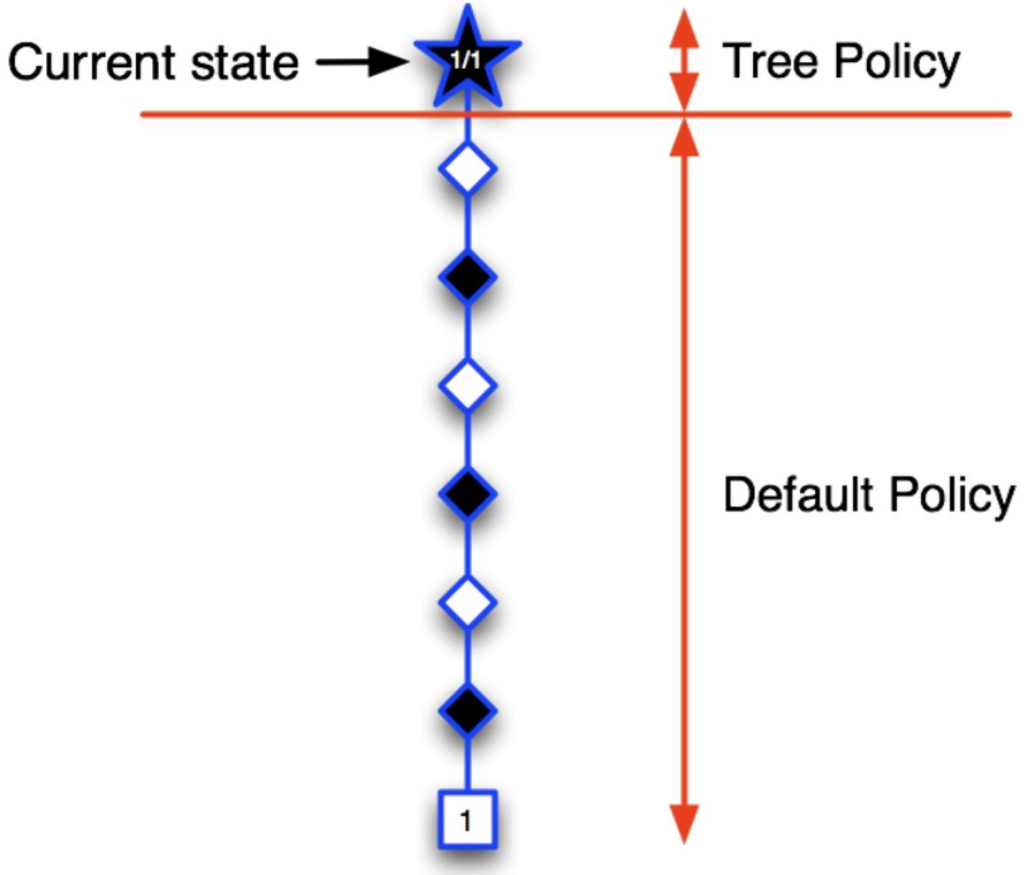
$$v_\pi(x) = \mathbb{E}_\pi \left\{ R_T \,|\, X = x \right\} = P(\text{Black wins} \,|\, X = x)$$

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

Monte-Carlo Evaluation in Go

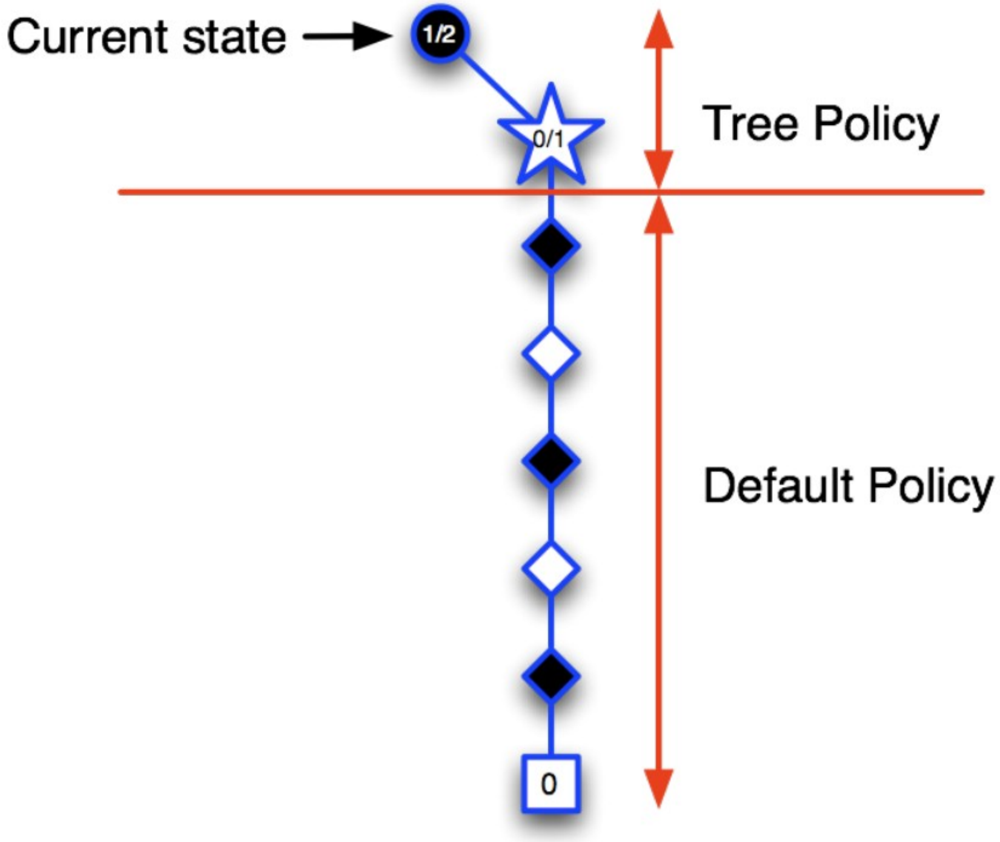


Example of MCTS



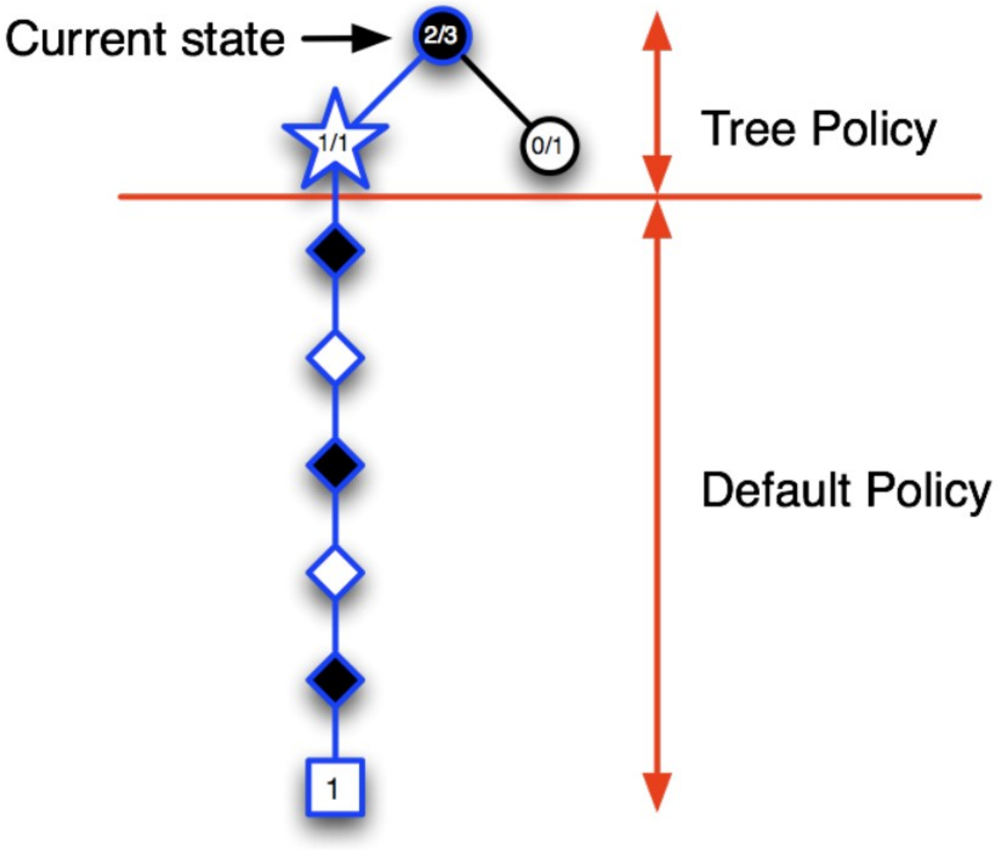
Material from Silver D. (2015) “Lectures on Reinforcement Learning”

Example of MCTS



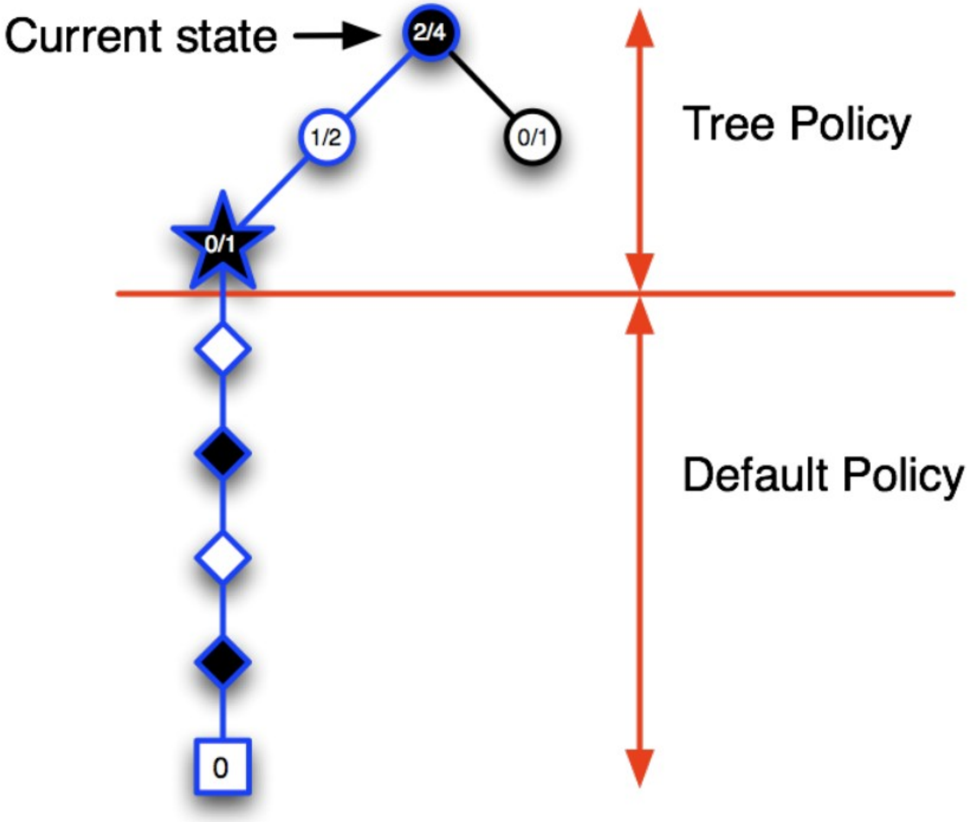
Material from Silver D. (2015) “Lectures on Reinforcement Learning”

Example of MCTS



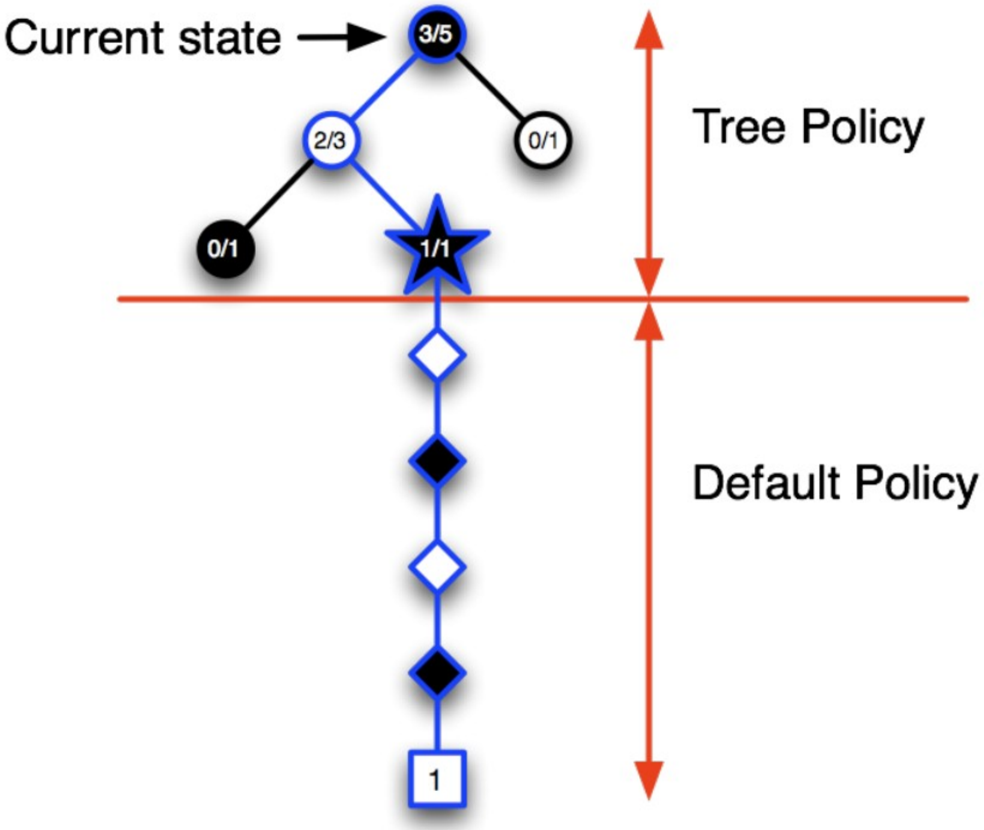
Material from Silver D. (2015) “Lectures on Reinforcement Learning”

Example of MCTS



Material from Silver D. (2015) “Lectures on Reinforcement Learning”

Example of MCTS



Material from Silver D. (2015) “Lectures on Reinforcement Learning”

MCTS

- Highly selective best-first search
- Evaluates states dynamically (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Works for “black-box” models (only requires samples)
- Computationally efficient, anytime, parallelizable

- There are multiple follow ups, for instance, improving the value of each cost, which affects the selection step, by using UCB or UCT.

Material from Silver D. (2015) “Lectures on Reinforcement Learning”