Learning in Games Lecture 3

Jonathan Shapiro

School of Computer Science
University of Manchester

November 7, 2023

Announcements

- Lectures end this week (so one more on Thursday)
- No more Examples classes
- The Project group allocation has been redone. Unchanged for most, but a few errors corrected.
- Next week you start the project, which runs until the end of term.

Monte Carlo Methods

In TD-learning you learn, and then you play.

In Monte Carlo learning you learn *while* you play. You learn between moves. (Like minimax search).

Monte Carlo board evaluation

This is a simple (mindless) way to generate a heuristic function. Often called “playouts”.

- To evaluate a board position v, do the following N times.
  - Play the game to the end, all players choose random moves
  - Return the average payoff

Estimates the strength of the board position by the average payoff of the nodes accessible to it.

The downside is the time cost to perform the N playouts per board position. Although you are not playing against a random player, it can be very useful.

Monte Carlo Tree Search (MCTS)

A specific realization of a Monte Carlo method, used for Go and other games.

- Go was a game which presented a challenge for computers.
- It is a zero-sum game of perfect information, so mini-max search with alpha-beta pruning is applicable.
- No one could think of a remotely good heuristic.
- It is estimated that the size of the GO tree is 5×10^{359} which is 10^{236} times bigger than the game tree for chess (from the AlphaGo paper).
- Prior to AlphaGo (2016), MCTS was the state-of-the-art approach.

Scales

- Atoms in a glass of water: 10^{24};
- Atoms in the observable universe: $10^{78} - -10^{821}$;
- Legal moves in chess: 10^{402};
- Legal moves in Go: 10^{1703}

[1] Source: Liverpool Museum
[2] ibid
[3] ibid

Elements of MCTS

- A game tree is built, incrementally but asymmetrically.
- For each iteration, a *tree policy* is used to find the most urgent mode to expand, using a strategy which balances exploration and exploitation.
- A simulation is run from the selected node using a *default policy* (often random choice of action).
- Update the search tree according to the result. This updates the value of the selected node and all its ancestors.
- Return the best child of the current root node.

MCTS — A good resource

- “A survey of Monte Carlo Tree Search Methods”, Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis and Simon Colton, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 1, MARCH 2012.
- Available at `https://ieeexplore.ieee.org/document/6145622`

Some of this lecture follows this reference.

General approach to find the next move

1. The current board position is v_0.
2. **While** (time remaining) **do**
   - 2.1 Use `TreePolicy`(v_0) to find next node to expand v_l
   - 2.2 Use `DefaultPolicy`(v_l) to simulate from v_l to a terminal node T (e.g. random action choice)
   - 2.3 Backup(v_l, T)
3. **Return** MOVE $=$`BestChild`(v_0)

Four phases of MCTS

Selection: Starting at the root, child selection policy is recursively applied to traverse the game tree, looking for the most important expandable node. A node is expandable if it is non-terminal and has unvisited children.

Expansion: Increase the size of the current partial tree (typically by one node) by expanding nodes (e.g. visit an unvisited child).

Play-out (simulation): Run a simulation from the added node by *random* self-play.

Back-up: The statistics of the added node(s) and its ancestors are updated.

Four phases of MCTS

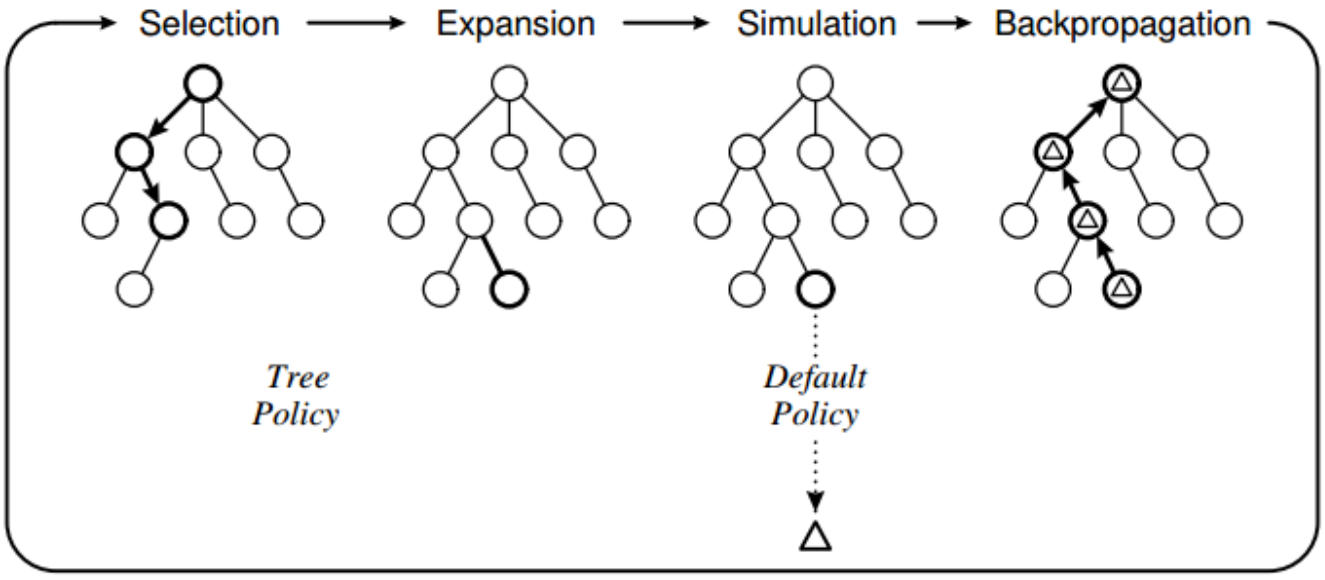
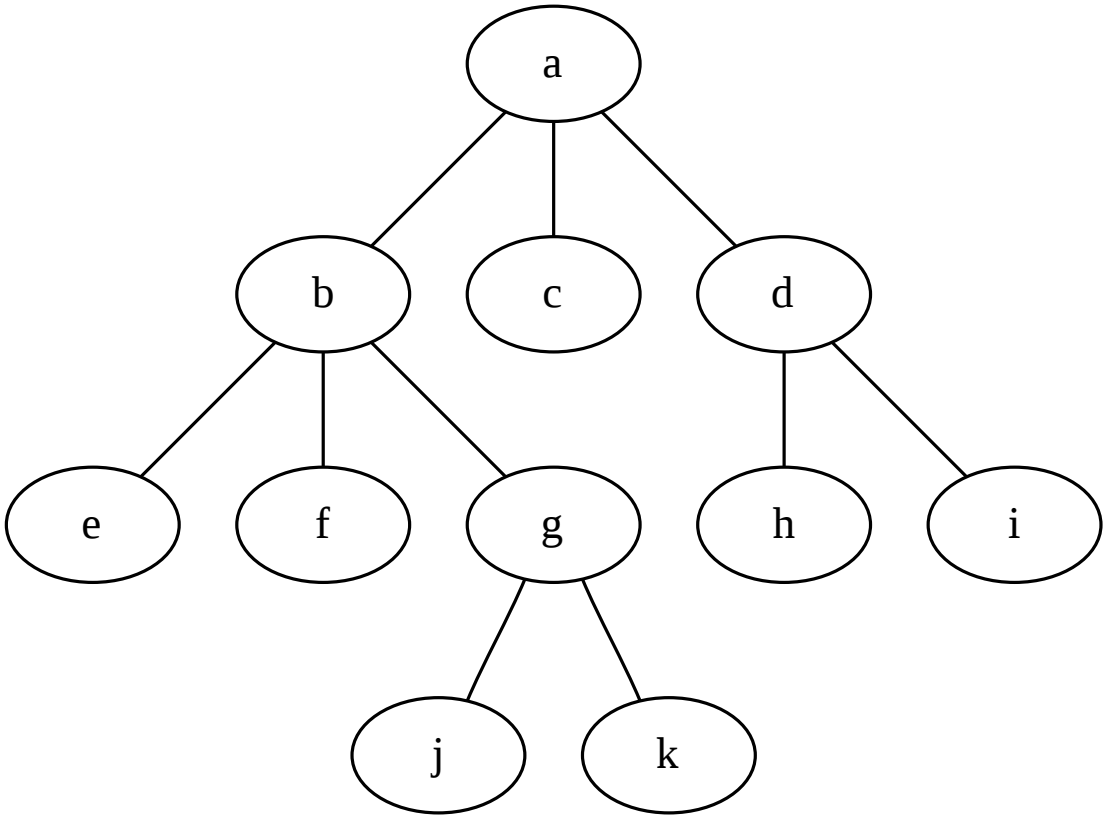
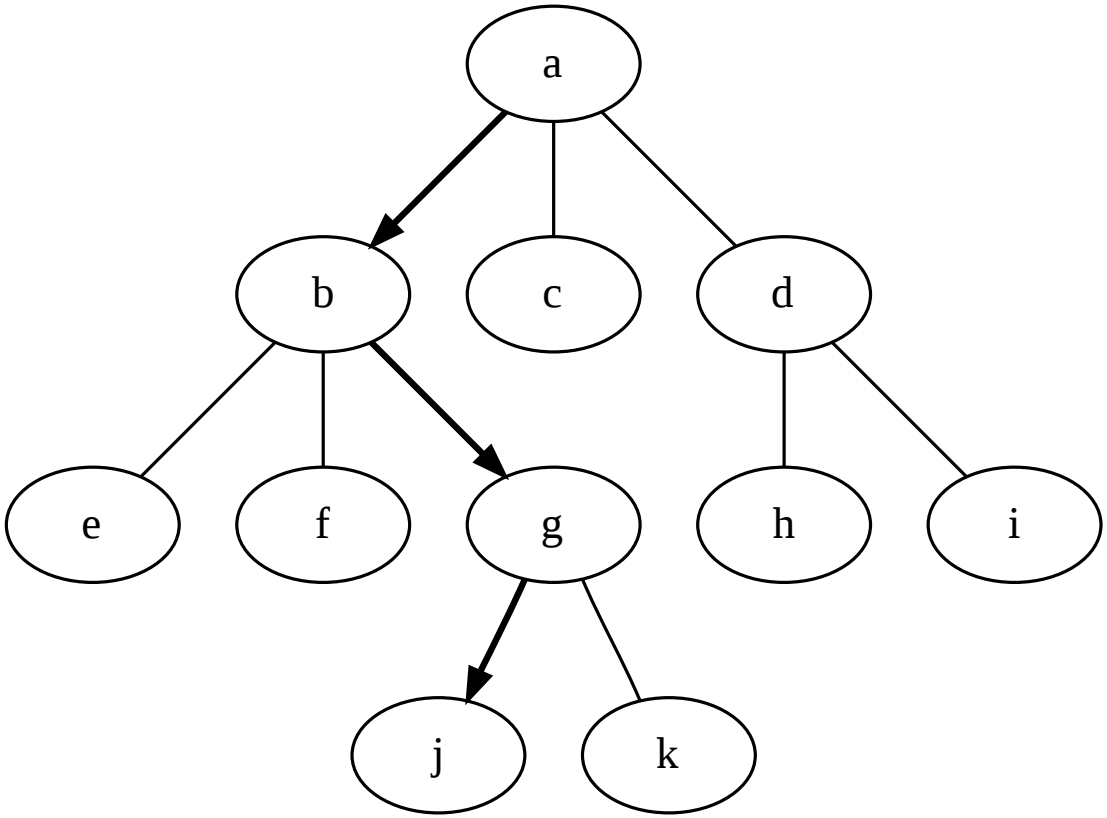


Figure from Browne et. al. (2012)

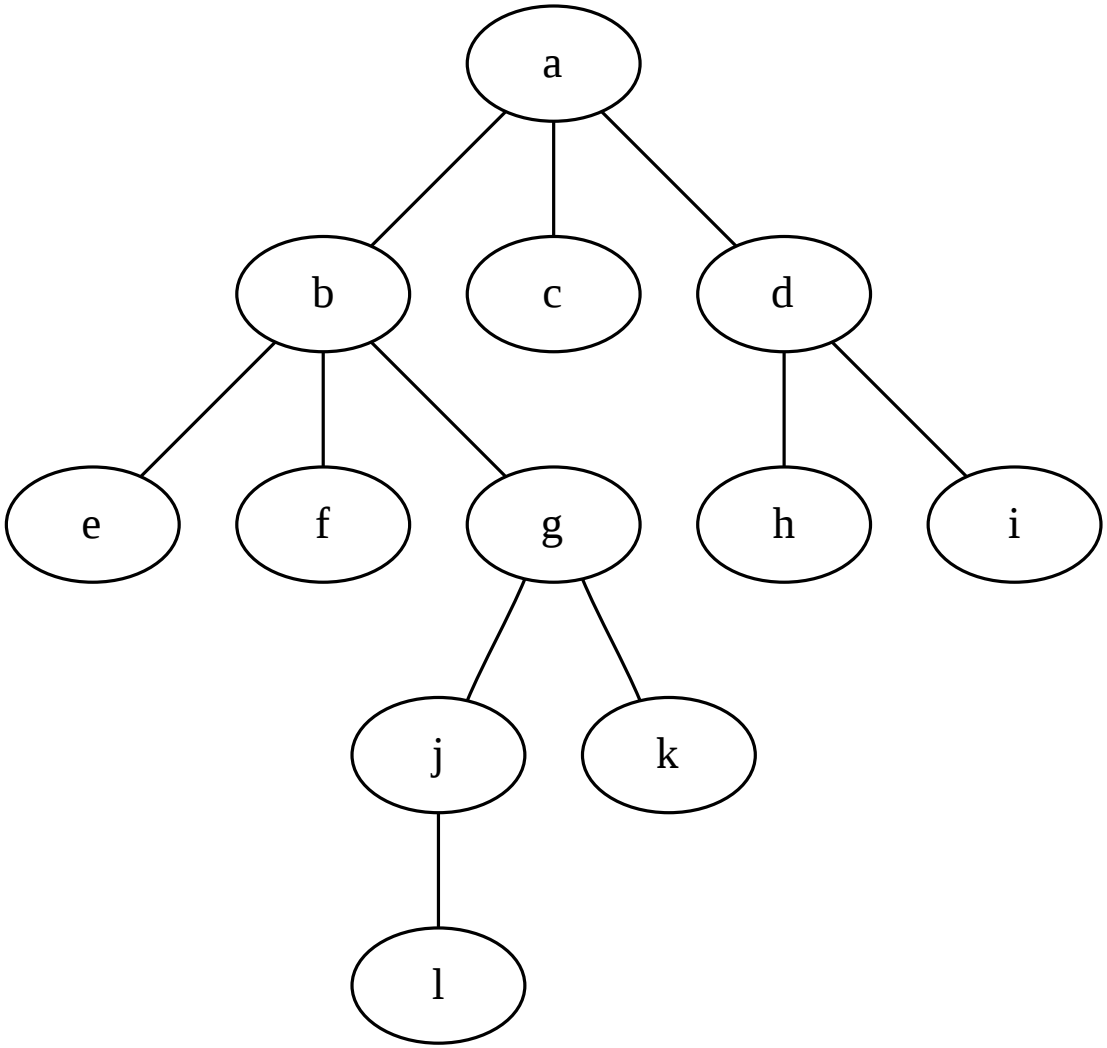
Elements of MCTS



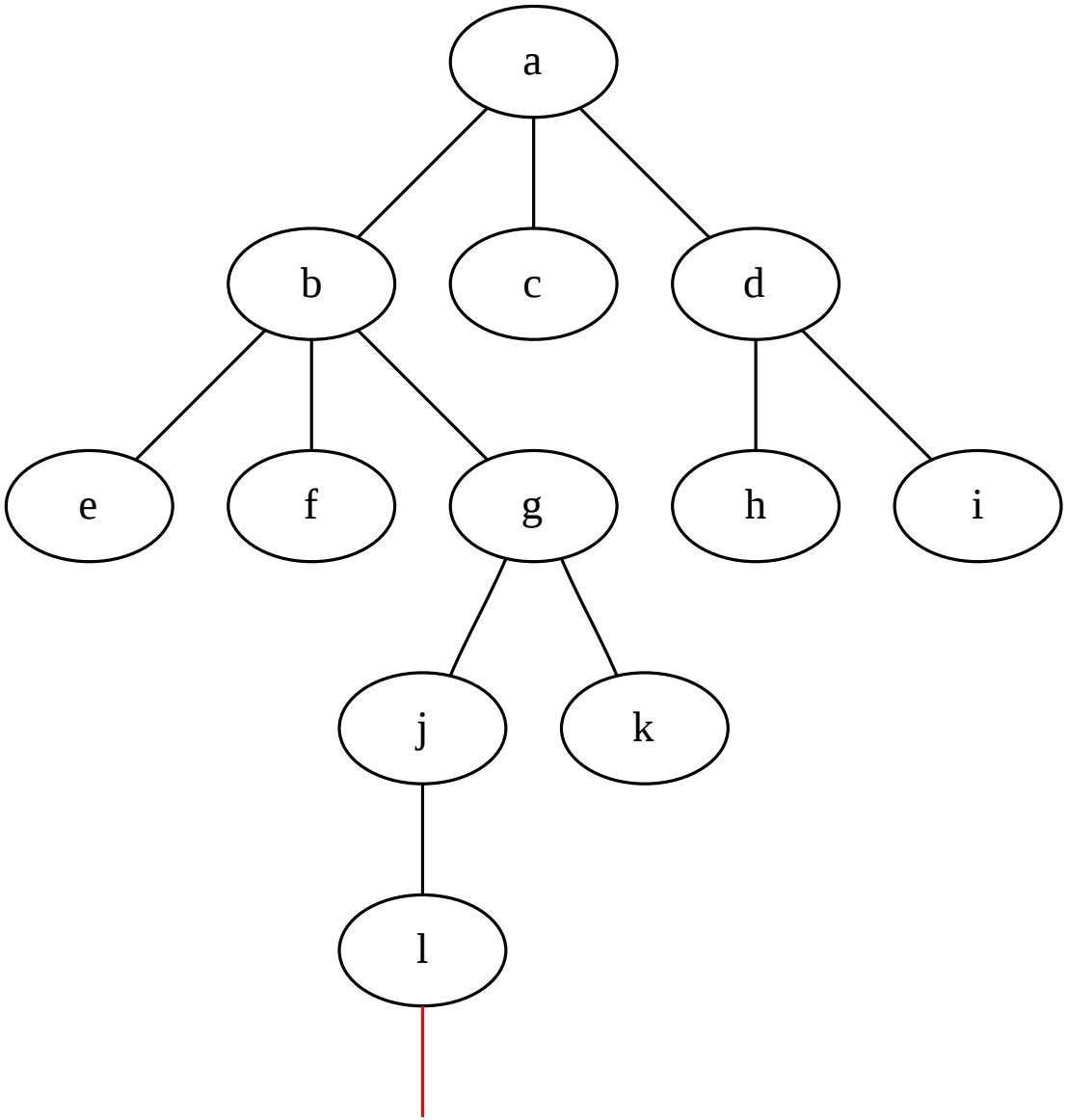
Elements of MCTS — Selection



Elements of MCTS — Expansion

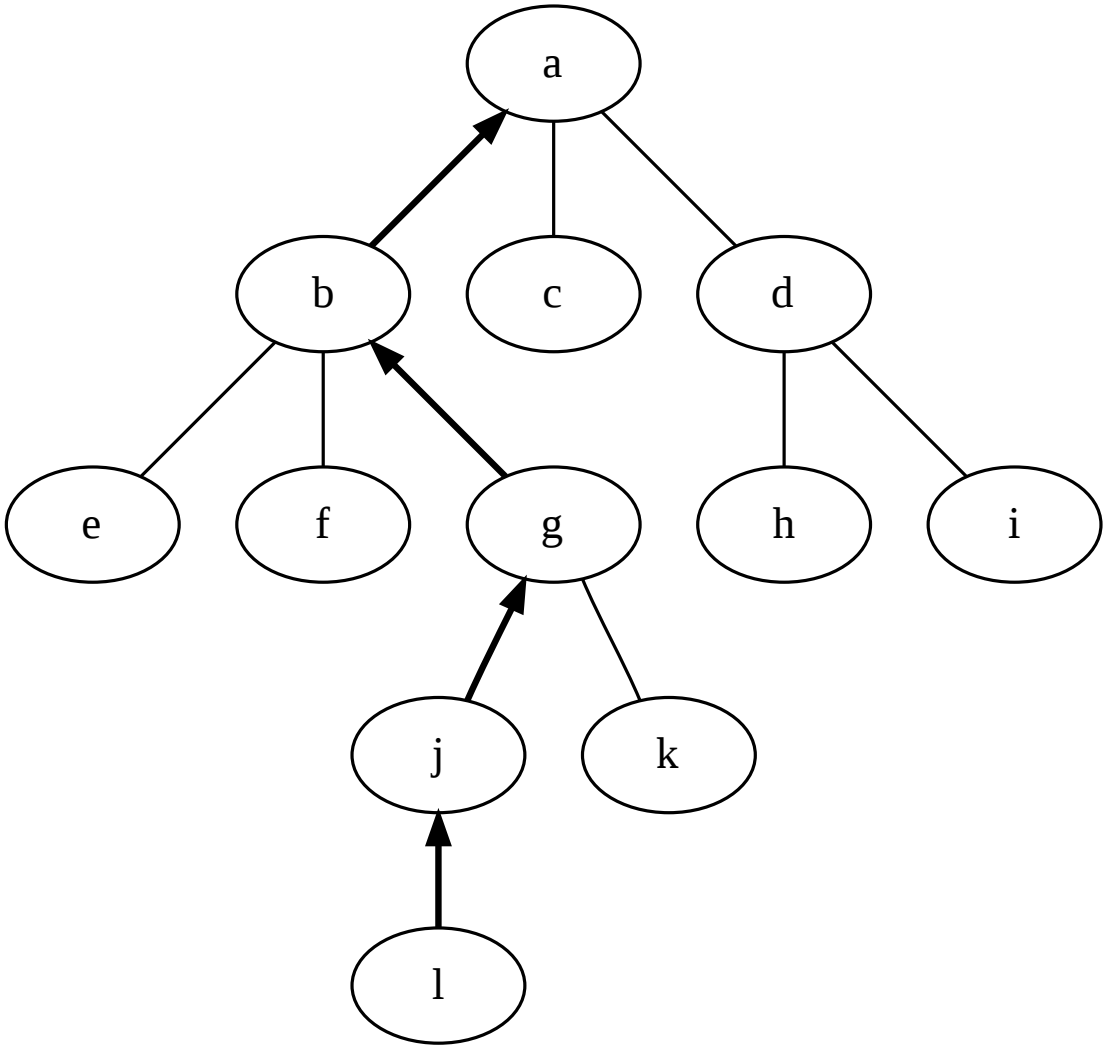


Elements of MCTS — Simulation



Learn to end of game

Elements of MCTS — Backup



UCT — preferred selection method

- An exploring strategy, like ϵ-greedy.
- Always chooses unselected children.
- Between two children of the same parent with the same average value, chooses child selected less often.
- In the long run, children with lower true values will be chosen less often. The best child will be chosen exponentially more often.

UCT — preferred selection method

Each node v stores two quantities,

1. $Q(v)$ — the sum of all payoffs received (all playouts which passed through this node).
2. $N(v)$ — a count of the number of times node visited.

So $Q(v)/N(v)$ is an estimate of the value of node v.

UCT — preferred selection method

At parent node v, choose child v' which maximizes

$$\underbrace{\frac{Q(v')}{N(v')}}_{\text{Exploit}} + \underbrace{C\sqrt{\frac{2\ln N(v)}{N(v')}}}_{\text{Explore}}.$$

From the point of view of the algorithm, `BestChild` returns

$$\operatorname{argmax}_{v' \in \text{children of } v} \left[\frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \right]$$

- Unvisited nodes always visited first.
- Explore term gets bigger when child is not explored but parent is;
- Explore term gets smaller when child is explored.
- C determines exploration-exploitation trade-off (tuned experimentally).

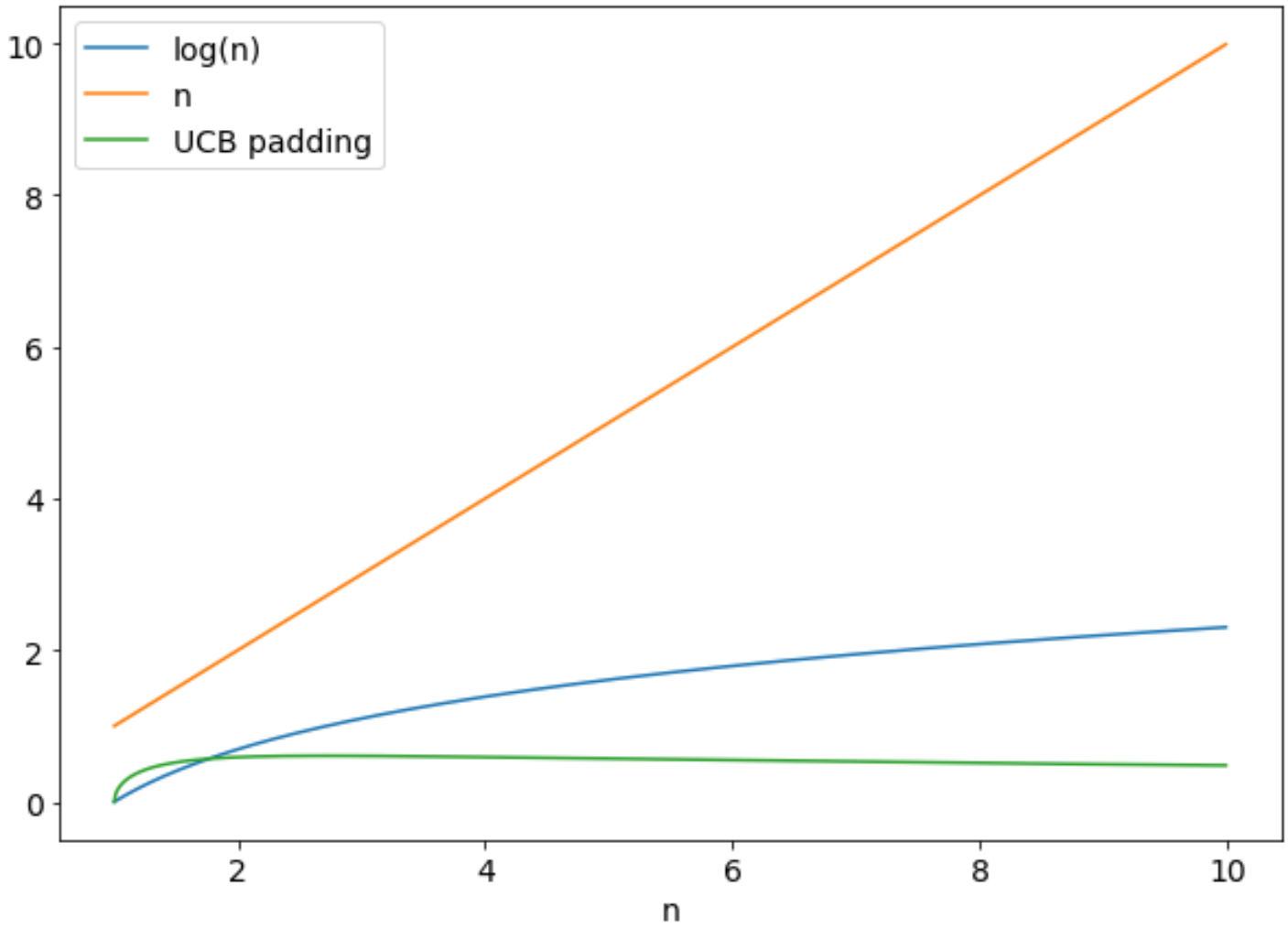
Upper confidence bound (UCB)

The exploration-exploitation strategy is actually called UCB.
UCT - Upper Confidence bounds for game Trees.

$$\text{UCB value of an action} = \text{ average reward } + C\sqrt{\frac{\log N}{n_a}}.$$

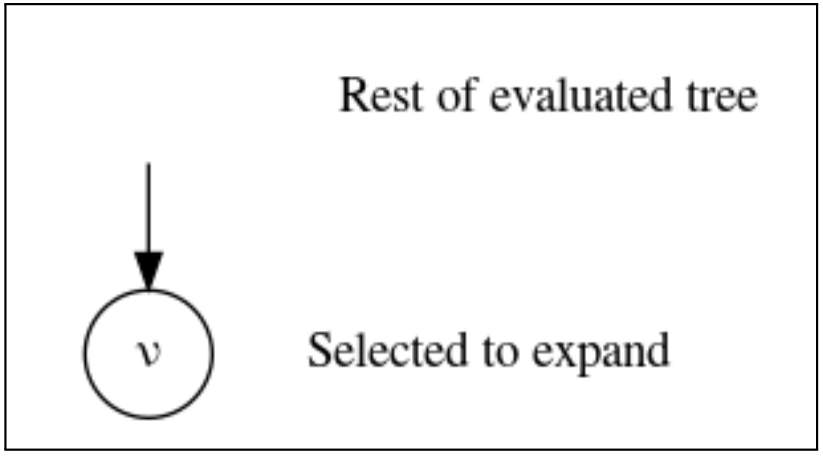
Here n_a is the number of times action a used; N number of times *any* action used.
Choose the action with largest UCB value.

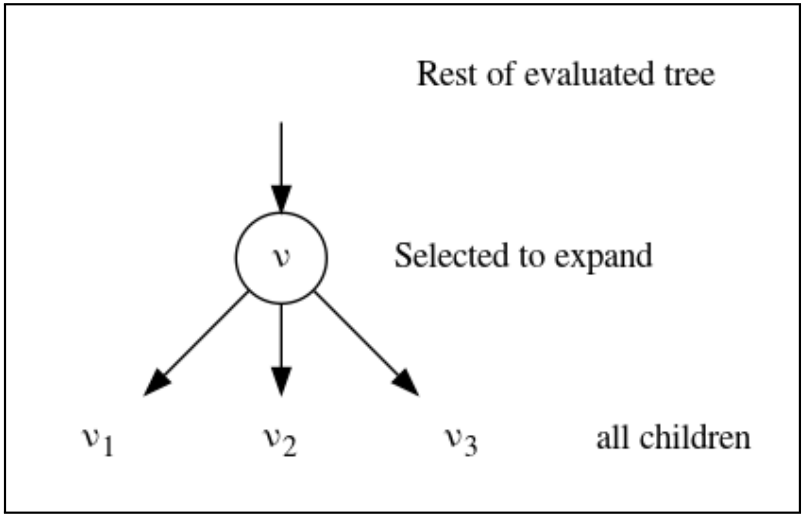
$$\text{UCB value of an action} = \text{average reward} + C\sqrt{\frac{\log N}{n_a}}.$$



UCT algorithm

1. Select next node to be evaluated (using selection method).
2. Evaluate all children once.
3. Repeatedly evaluated the child with the highest UCB value.





Demo

Actual but unknown winning rates for each child:

- Child 1 wins at a rate 0.1;
- Child 2 wins at a rate 0.8;
- Child 3 wins at a rate 0.2

Aside on exploration-exploitation strategies

Ideally, you might want a strategy which:

- Explores a lot, when little is known;
- stops exploring when the optimum is found.

- That's impossible! (Lai and Robbins, 1985)

You may never know when you have found the optimum, if there is stochasticity (randomness) or change.

The best you can do

- The best you can is explore the best action/child exponentially more often than suboptimal action/child.
- Details of which depend on the explore-exploit algorithm.

UCB achieves this. ϵ-greedy with constant ϵ does not (obviously).

Ending the search and making a move

When the time budget runs out, the algorithm returns the move to be made. This will be a child of the root node. Possibilities:

Max child (default): Choose the child with the highest average reward (highest Q/N).

Robust child: Choose the child most often visited (highest N).

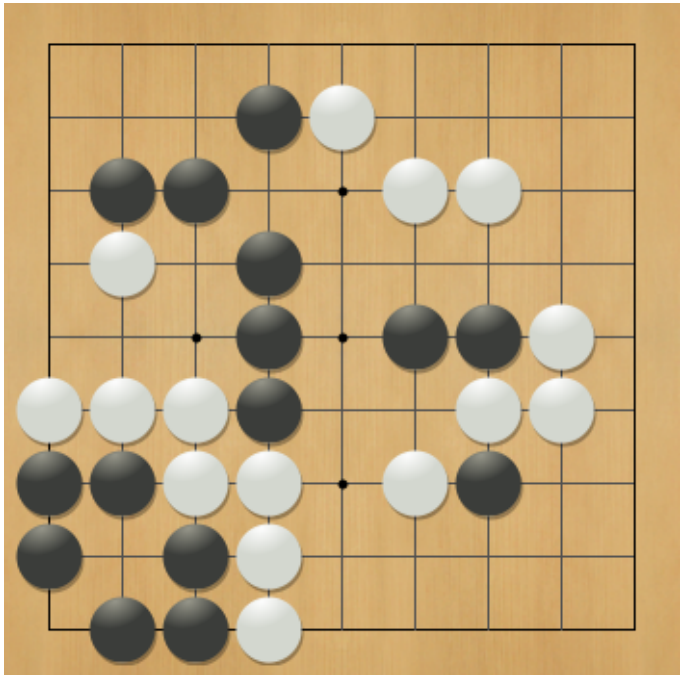
Max-Robust child: Choose the child which is maximal in both reward and visits. If none exists, run longer until one does exist.

Secure child: Choose child which maximizes a lower confidence interval, such as (note minus sign):

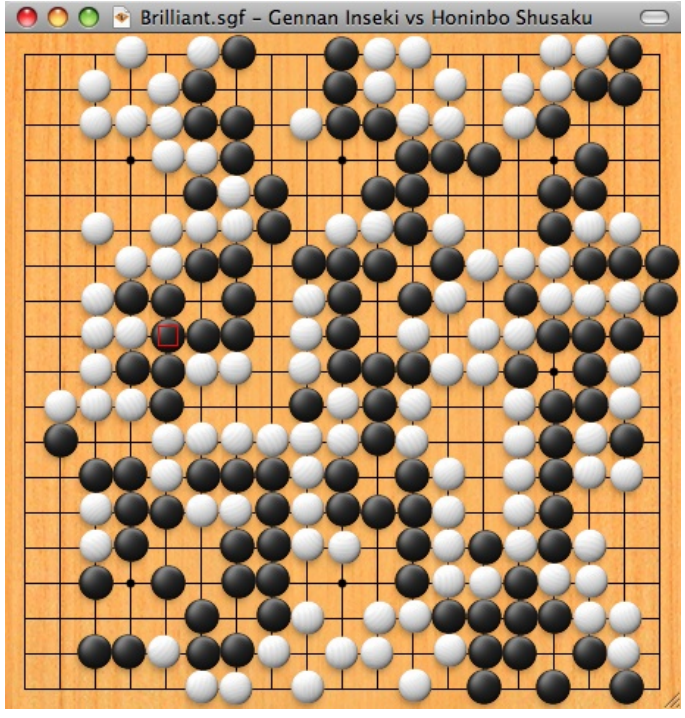
$$\frac{Q(v')}{N(v')} - C\sqrt{\frac{2\ln N(v)}{N(v')}}.$$

Application to Go

9x9 board (small)



19x19 board (large)



Claims about Go

2005: Computer Go is impossible.

2006: UCT proposed and applied to 9X9 Go.

2007: Human master level achieved at 9X9 Go

2008: Human grandmaster achieved at 9x9 Go (Teytaud et. al.)

2009: Beat a human (1 dan) professional on a 19x19 board, but with a 6 stone handicap.

2015: AlphaGo beats Fan Hui, a professional 2 dan, and 2013, 2014 and 2015 European Go champion 5-0 in games. In March 2016, it beats Lee Sedol, 9 dan and the second most winning professional in the world. Summer 2017, beats Ke Jie 3 games to zero.

MCTS algorithms win most computer tournaments since 2007 – 2015.

Comparison

TD-learning:

- Needs to learn in self-play before used as a game-playing agent.
- Learns board value-function or Q table.

Monte Carlo tree search:

- Learns while playing the game.
- Uses a form of exploration-exploitation strategy (UCT) to selection nodes to evaluate.
- Uses random play-out to evaluate them.

AlphaGo

Combines supervised learning, reinforcement learning, and Monte Carlo tree search.

References:

- **Mastering the Game of Go with Deep Neural Networks and Tree Search.** David Silver et al. in Nature, Vol. 529, pages 484–489; January 28, 2016. `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`
- See also **How the Computer Beat the Go Player**, Christof Koch, Scientific American Mind (July/August 2016), 27, 20–23. `http://www.nature.com/scientificamericanmind/journal/v27/n4/full/scientificamericanmind0716-20.html`

Next lecture

- AlphaGo
- Other reinforcement learning approaches.
- About the project.