Tutorial: Monte Carlo Game Tree Search

CSC384 – Introduction to Artificial Intelligence

March 17, 2017

by Alberto Camacho and Amit Kadan

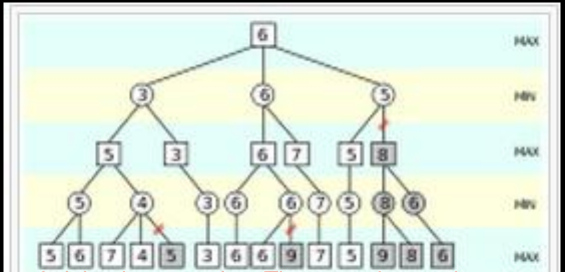
These slides have been obtained from the following sources:

- AAAI-14 Games Tutorial, by Martin Muller:
  - https://webdocs.cs.ualberta.ca/~mmueller/courses/2014-AAAI-games-tutorial/slides/AAAI-14-Tutorial-Games-5-MCTS.pdf
- International Seminar on New Issues in Artificial Intelligence, by Simon. Lucas:
  - http://scalab.uc3m.es/~seminarios/seminar11/slides/lucas2.pdf
- Constraint Programming 2012, Tutorial on Monte-Carlo Tree Search, by Michele Sebag:
  - https://www.lri.fr/~sebag/Slides/InvitedTutorial_CP12.pdf
- Artificial Intelligence course at Swarthmore College, by Bryce Wiedenbeck
  - https://www.cs.swarthmore.edu/~bryce/cs63/s16/slides/2-15_MCTS.pdf
- Introduction to Monte Carlo Tree Search, by Jeff Bradberry
  - https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

Conventional Game Tree Search

https://en.wikipedia.org/wiki/Alpha–beta_pruning

- Minimax with alpha-beta pruning, transposition tables
- Works well when:
  - A good heuristic value function is known
  - The branching factor is modest
- E.g. Chess, Deep Blue, Rybka etc.

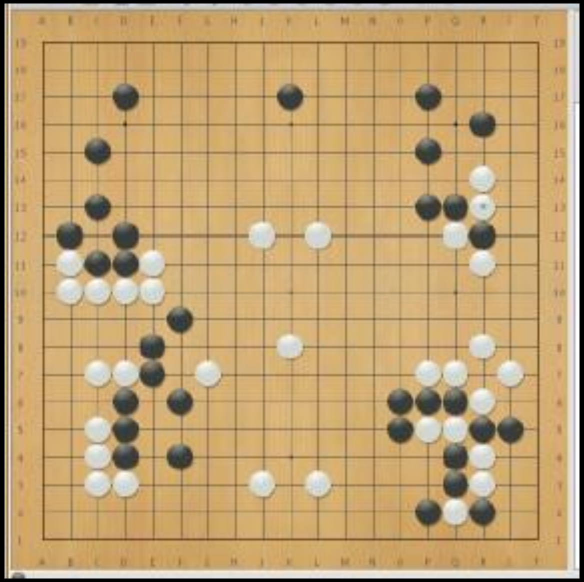


An illustration of alpha–beta pruning. The grayed-out subtrees need not be explored (when moves are evaluated from left to right), since we know the group of subtrees as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result.

Whenever the maximum score that the minimizing player(beta) is assured of becomes less than the minimum score that the maximizing player(alpha) is assured of (i.e. beta <= alpha), the maximizing player need not consider the descendants of this node as they will never be reached in actual play.

Go

- Much tougher for computers
- High branching factor
- No good heuristic value function



“Although progress has been steady, it will take many decades of research and development before world-championship–calibre go programs exist”.
Jonathan Schaeffer, 2001

Monte Carlo Tree Search (MCTS)

- Revolutionised the world of computer go
- Best GGP players (2008, 2009) use MCTS

General game playing (GGP) is the design of artificial intelligence programs to be able to play more than one game successfully

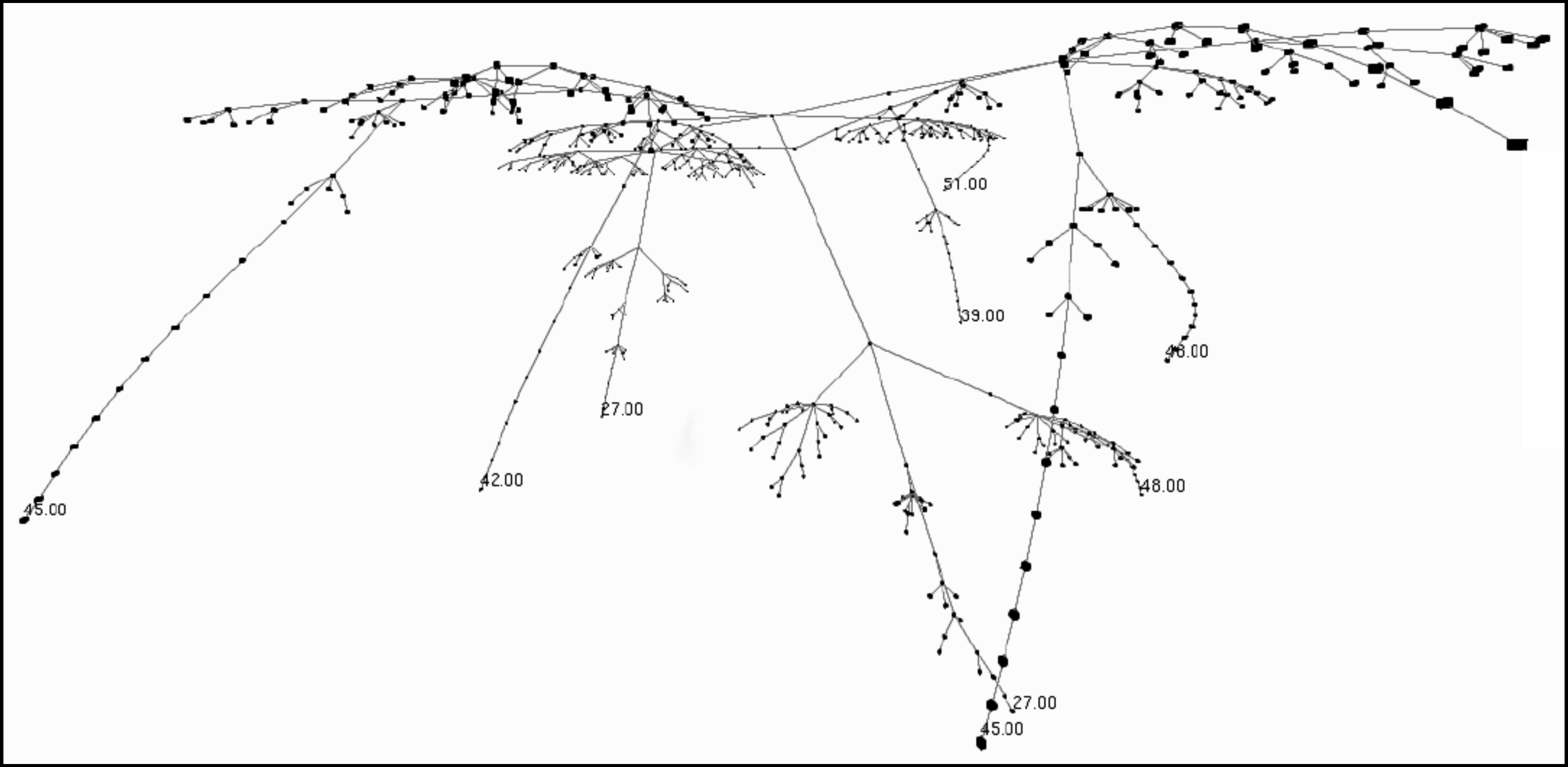
- More CPU cycles leads to smarter play
  - Typically lin / log: each doubling of CPU time adds a constant to playing strength
- Uses statistics of deep look-ahead from randomised roll-outs
- Anytime algorithm

MCTS

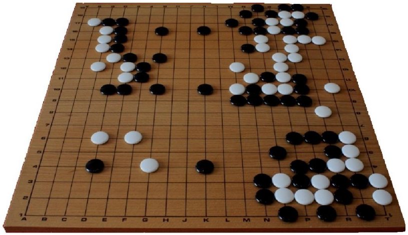
- Builds and searches an asymmetric game tree to make each move
- Phases are:
  - Tree search: select node to expand using tree policy
  - Perform random roll-out to end of game when true value is known
  - Back the value up the tree

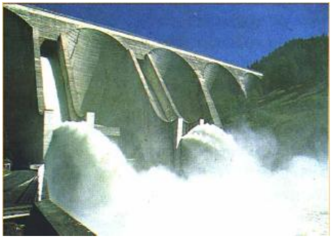
Sample MCTS Tree

(fig from CadiaPlayer,
Bjornsson and Finsson, IEEE T-CIAIG)

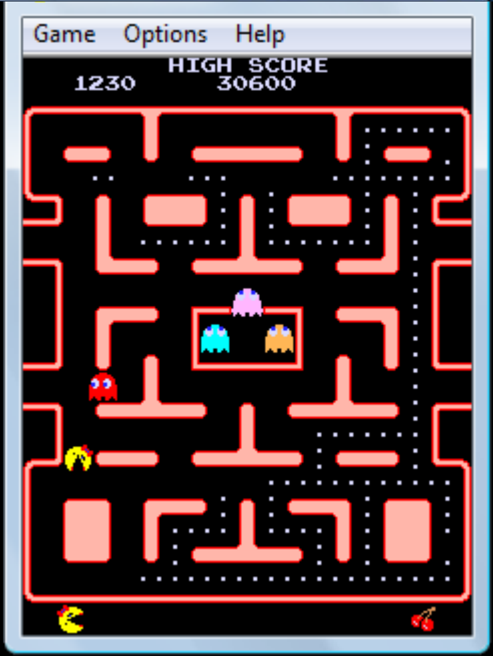


Not just a game: same approaches apply to optimal energy policy





Sample Games



Brief History of Monte Carlo Methods

Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle.

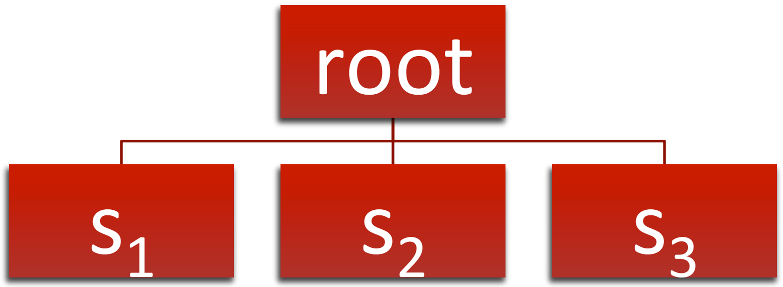
1940's – now	Popular in Physics, Economics, … to simulate complex systems
1990	(Abramson 1990) expected-outcome
1993	Brügmann, *Gobble*
2003 – 05	Bouzy, Monte Carlo experiments
2006	Coulom, *Crazy Stone*, **MCTS**
2006	(Kocsis & Szepesvari2006) **UCT**
2007 – now	*MoGo*, *Zen*, *Fuego*, many others
2012 – now	MCTS survey paper (Browne et al 2012); huge number of applications

https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

(Very) Basic Monte Carlo Search

- Play lots of random games
  - start with each possible legal move
- Keep winning statistics
  - Separately for each startingmove
- Keep going as long as you have time, then…
- Play move with best winning percentage

Example – Basic Monte Carlo Search

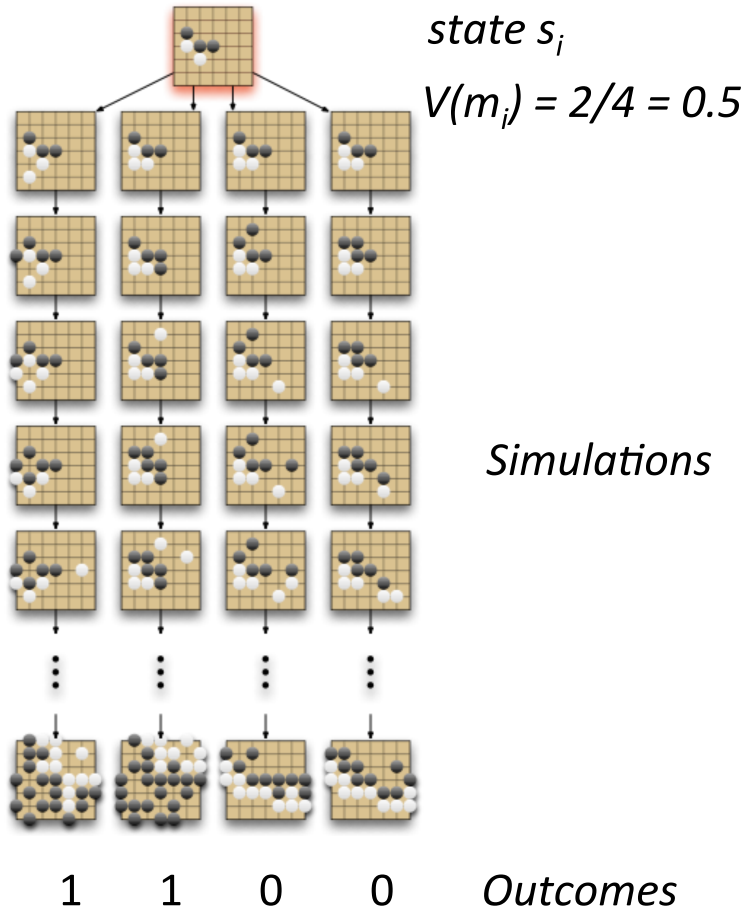


1 ply tree

root = current position

s_1 = state after move m_1

s_2 = …



Naïve Approach

- Use simulations directly as an evaluation function for αβ
- Problems
  - Single simulation is very noisy, only 0/1 signal
  - running many simulations for one evaluation is very slow
  - Example:
    - typical speed of chess programs **1 million** eval/second
    - Go: 1 million moves/second, 400 moves/simulation, 100 simulations/eval = **25** eval/second
- Result: Monte Carlo was ignored for over 10 years in Go

Monte Carlo Tree Search

- Idea: use results of simulations to guide growth of the game tree
- **Exploitation**: focus on promising moves
- **Exploration**: focus on moves where uncertainty about evaluation is high
- Two contradictory goals?
  - Theory of *bandits* can help

Bandits

- Multi-armed bandits (slot machines in Casino)
- Assumptions:
  - Choice of several *arms*
  - each arm pull is independent of other pulls
  - Each arm has *fixed, unknown average payoff*
- Which arm has the best average payoff?
- Want to minimize *regret* = loss from playing non-optimal arm

Explore and Exploit with Bandits

- *Explore* all arms, but also:
- *Exploit*: play promising arms more often
- Minimize *regret* from playing poor arms

in order to ensure that the optimal arm is not missed due to temporarily promising rewards from a sub-optimal arm.
It is thus important to place an upper confidence bound on the rewards observed so far that ensures this.

Upper confidence bound (UCB)

Pick each node with probability proportional to:

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

parent node visits

value estimate

tunable parameter

number of visits

- probability is decreasing in the number of visits (explore)
- probability is increasing in a node's value (exploit)
- always tries every option once

UCB1 Formula (Auer et al 2002)

- Name UCB stands for <u>U</u>pper <u>C</u>onfidence <u>B</u>ound
- Policy:

1. First, try each arm once
2. Then, at each time step:
   - choose arm *i* that maximizes the *UCB1 formula* for the upper confidence bound:

$$\bar{x}_i + \sqrt{\frac{2\ln(n)}{n_i}}$$

Theoretical Properties of UCB1

- Main question: rate of convergence to optimal arm
- Huge amount of literature on different bandit algorithms and their properties
- Typical goal: regret O(log n) for n trials
- For many kinds of problems, cannot do better asymptotically (Lai and Robbins 1985)
- UCB1 is a simple algorithm that achieves this asymptotic bound for many input distributions

The case of Trees: From UCB to UCT

Upper Confidence Bound 1 applied to trees

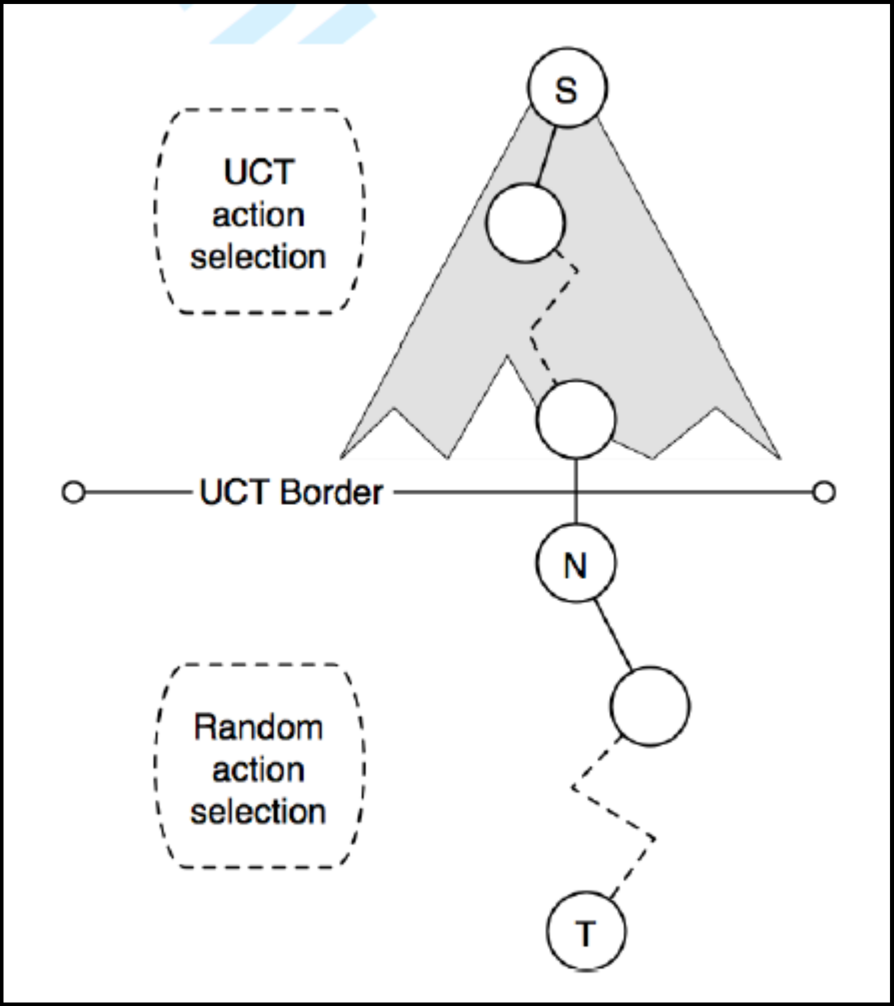
- UCB makes a single decision
- What about sequences of decisions (e.g. planning, games)?
- Answer: use a lookahead tree (as in games)
- Scenarios
  - Single-agent (planning, all actions controlled)
  - **Adversarial** (as in games, or worst-case analysis)
  - Probabilistic (average case, “neutral” environment)

Our Focus

MCTS Operation

(fig from CadiaPlayer,
Bjornsson and Finsson, IEEE T-CIAIG)

- Each iteration starts at the root
- Follows tree policy to reach a leaf node
- Then perform a random roll-out from there
- Node ‘N’ is then added to tree
- Value of ‘T’ back-propagated up tree



Generic Monte Carlo Tree Search

- *Select* leaf node L in game tree
- *Expand* children of L
- *Simulate* a randomized game from (new) leaf node
- *Update* (or backpropagate) statistics on path to root

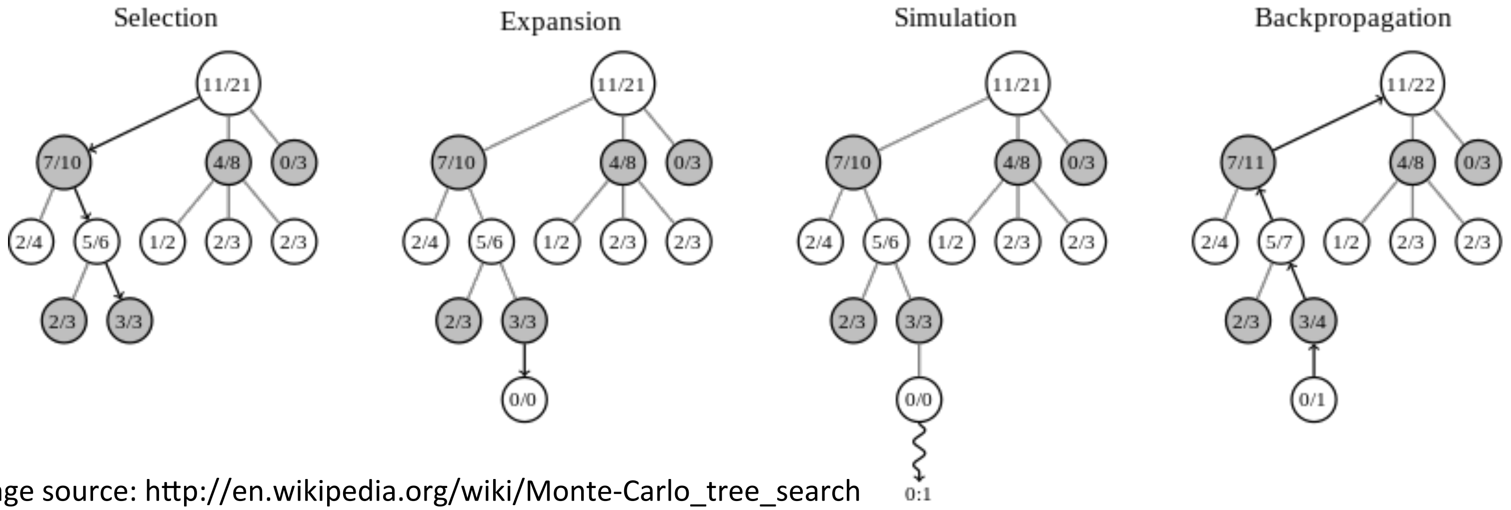
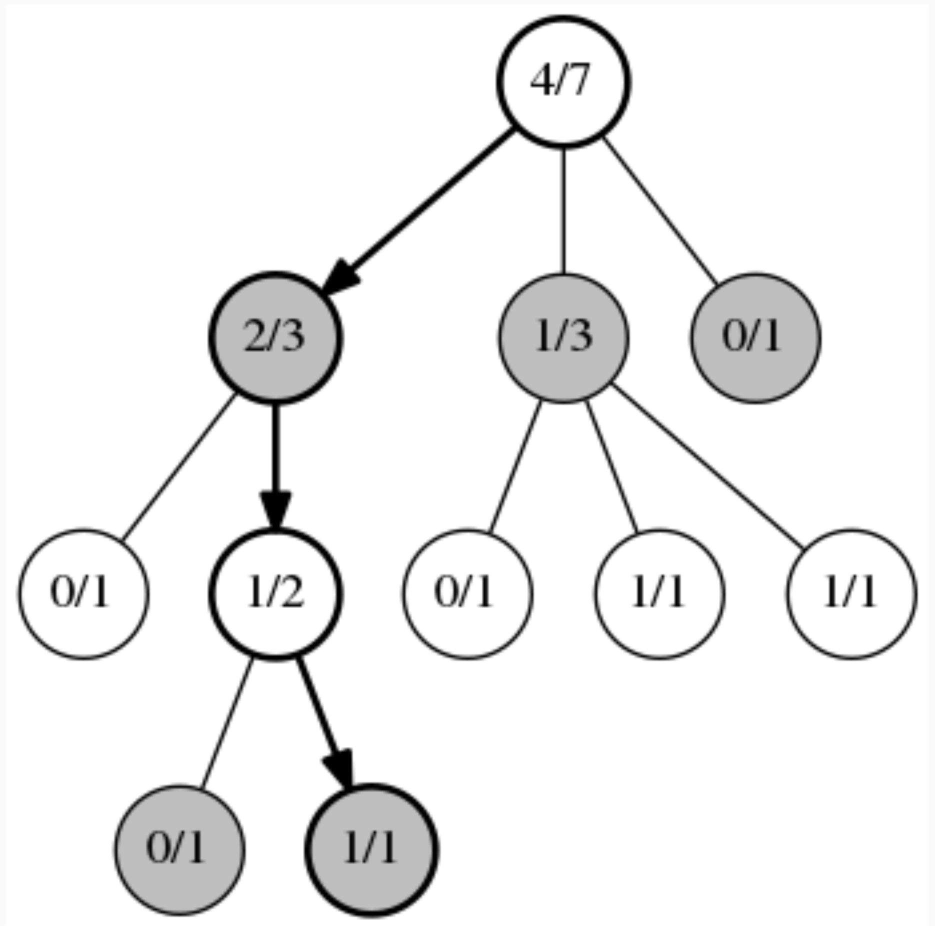


Image source: http://en.wikipedia.org/wiki/Monte-Carlo_tree_search

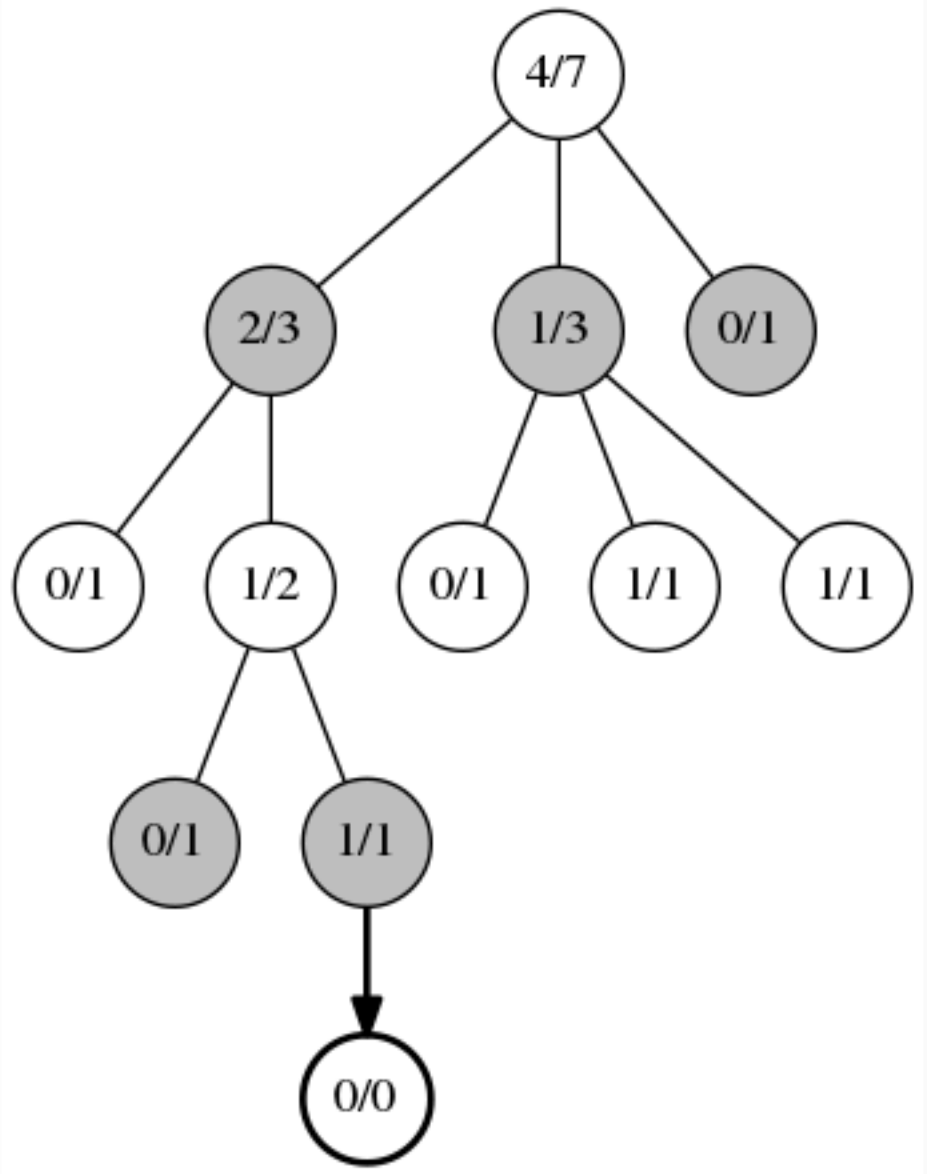


Selection

Here the positions and moves selected by the UCB1 algorithm at each step are marked in bold. Note that a number of playouts have already been run to accumulate the statistics shown. Each circle contains the number of wins / number of times played.

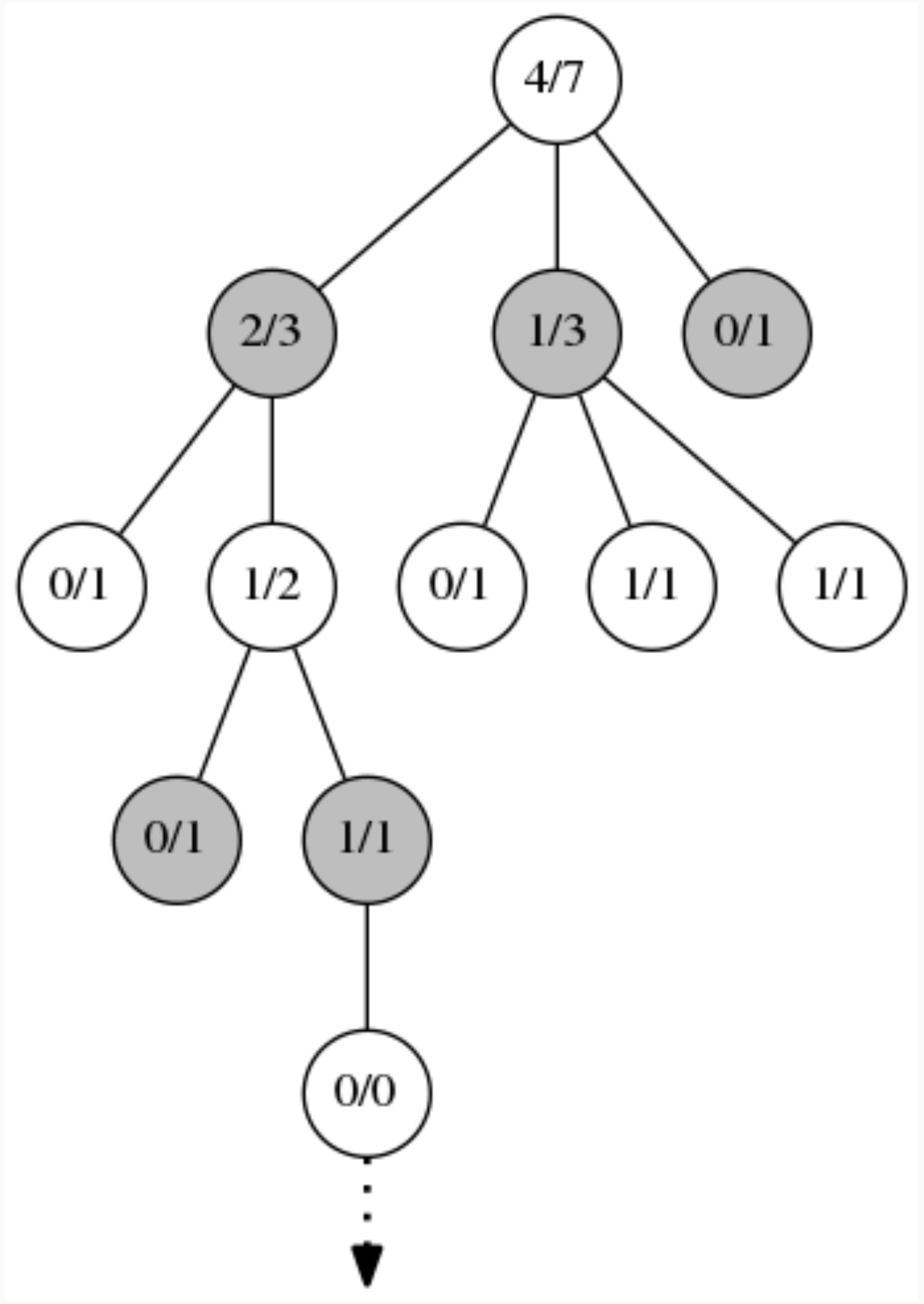
The first phase, *selection*, lasts while you have the statistics necessary to treat each position you reach as a multi-armed bandit problem. The move to use, then, would be chosen by the UCB1 algorithm instead of randomly, and applied to obtain the next position to be considered. Selection would then proceed until you reach a position where not all of the child positions have statistics recorded.

The second phase, *expansion*, occurs when you can no longer apply UCB1. An unvisited child position is randomly chosen, and a new record node is added to the tree of statistics.



Expansion

The position marked 1/1 at the bottom of the tree has no further statistics records under it, so we choose a random move and add a new record for it (bold), initialized to 0/0.

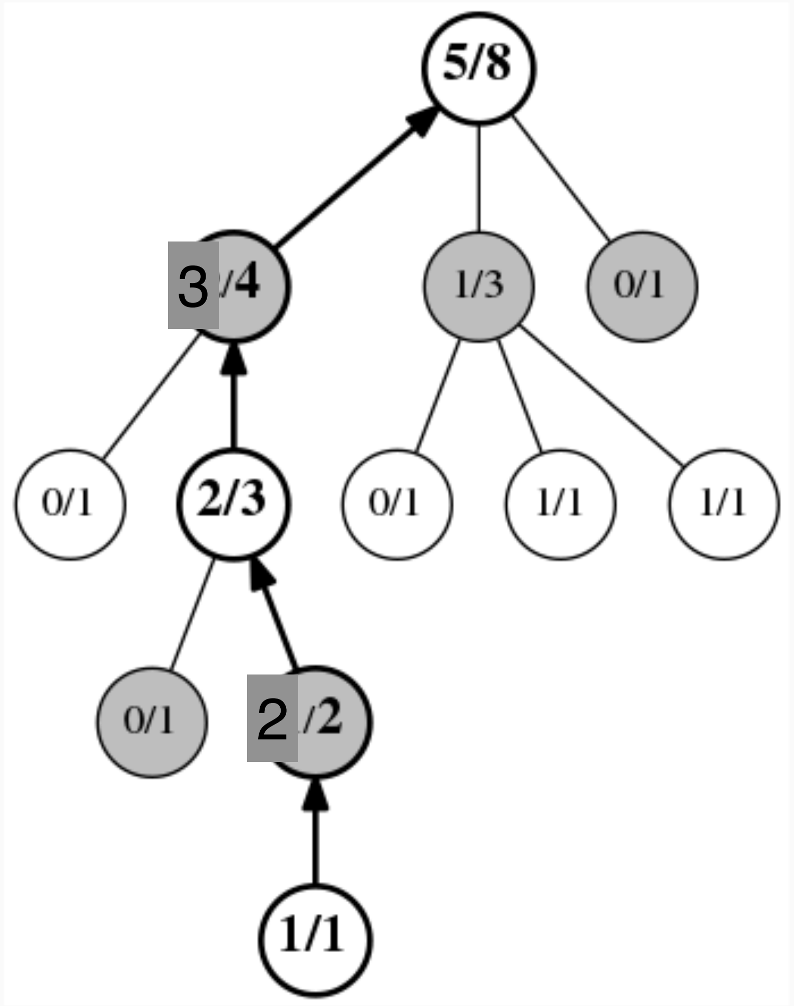


Simulation

Once the new record is added, the Monte Carlo simulation begins, here depicted with a dashed arrow. Moves in the simulation may be completely random, or may use calculations to weight the randomness in favor of moves that may be better.

After expansion occurs, the remainder of the playout is in phase 3, *simulation*. This is done as a typical Monte Carlo simulation, either purely random or with some simple weighting heuristics if a *light playout* is desired, or by using some computationally expensive heuristics and evaluations for a *heavy playout*. For games with a lower branching factor, a light playout can give good results.

Finally, the fourth phase is the *update* or *back-propagation* phase. This occurs when the playout reaches the end of the game. All of the positions visited during this playout have their play count incremented, and if the player for that position won the playout, the win count is also incremented.



Back-Propagation

After the simulation reaches an end, all of the records in the path taken are updated. Each has its play count incremented by one, and each that matches the winner has its win count incremented by one, here shown by the bolded numbers.

Summary – MCTS So Far

- UCB, UCT are very important algorithms in both theory and practice
- Well founded, convergence guarantees under relatively weak conditions
- Basis for extremely successful programs for games and many other applications

Impact - Applications of MCTS

- Classical Board Games
  - Go, Hex
  - Amazons
  - Lines of Action, Arimaa, Havannah, NoGo, Konane,…
- Multi-player games, card games, RTS, video games
- Probabilistic Planning, MDP, POMDP
- Optimization, energy management, scheduling, distributed constraint satisfaction, library performance tuning, …

Impact – Strengths of MCTS

- Very general algorithm for decision making
- Works with very little domain-specific knowledge
  - Need a simulator of the domain
- Can take advantage of knowledge when present
- Successful parallelizations for both shared memory and massively parallel distributed systems