# Advanced Games Programming (AI)
# Part 6: Sampling-Based Search

Michael Buro

December 23, 2024

[Under Construction]

# Change Log

- [Oct 31] Created

# Outline

1. Sampling-Based Methods
2. Monte Carlo Tree Search
3. UCB
4. UCT

# [AI-Lec 16 L26] Sampling-Based Methods

What do you do when you have imperfect information, huge state spaces, or no idea how to evaluate positions heuristically?

- Poker [1], Contract Bridge [2], Skat [8]:
  we don't know the opponent's cards
- Scrabble [3,4]: we don't know the opponent's tiles
- 19x19 Go [7,11,14]: many moves, complex state evaluation
- RTS games [10,13]: huge state and action spaces, Fog of War, realtime

Regular search often does not work well because of combinatorial explosion

# Choices?

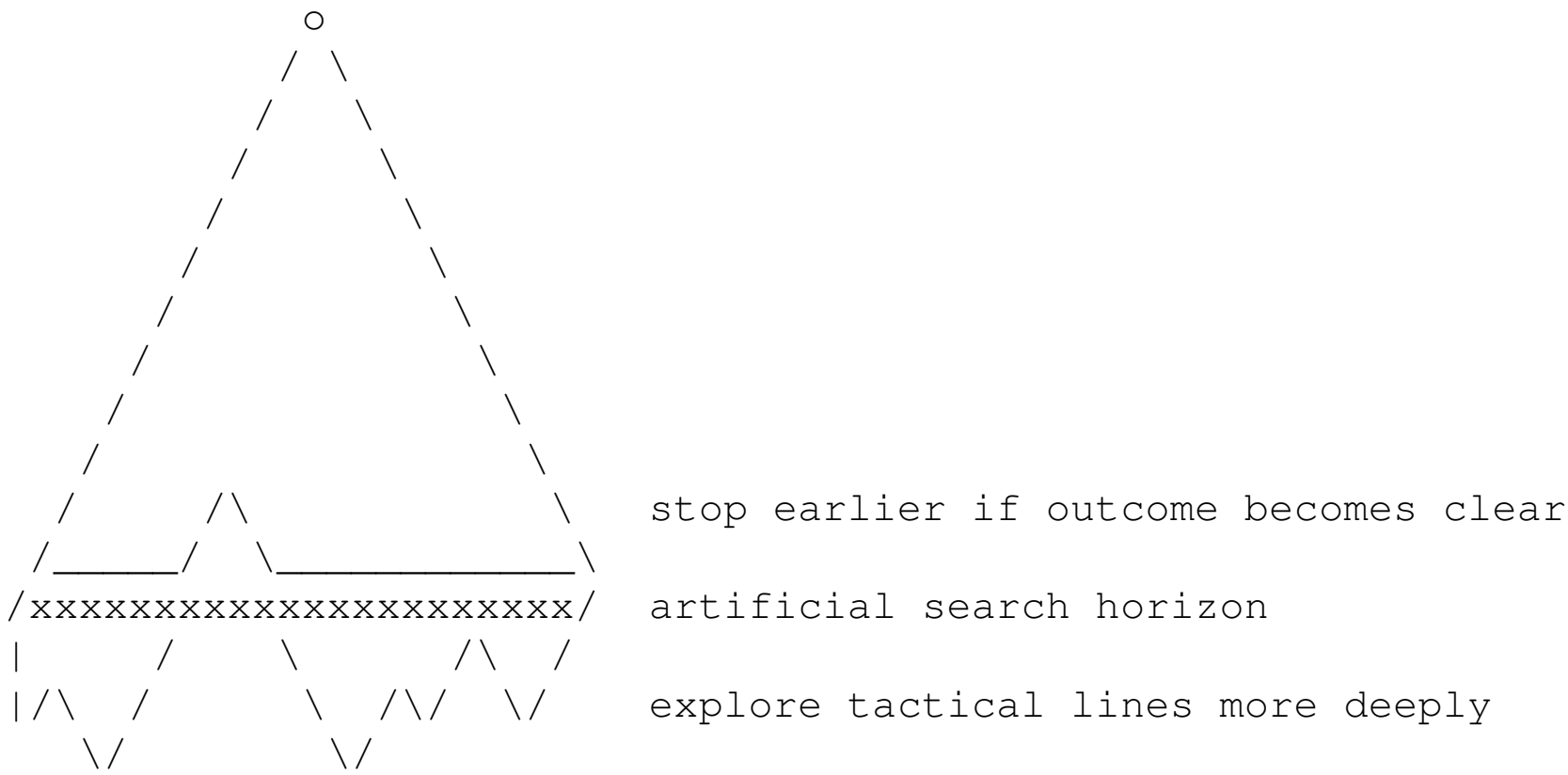
Alpha-Beta?

*-Minimax (Alpha-Beta applied to game trees with chance nodes)?

- We don't know what moves the opponent might have
- Could have a large branching factor, meaning little search depth

How do you get meaningful results when there can be stochastic events and hidden information?

# Traditional Search

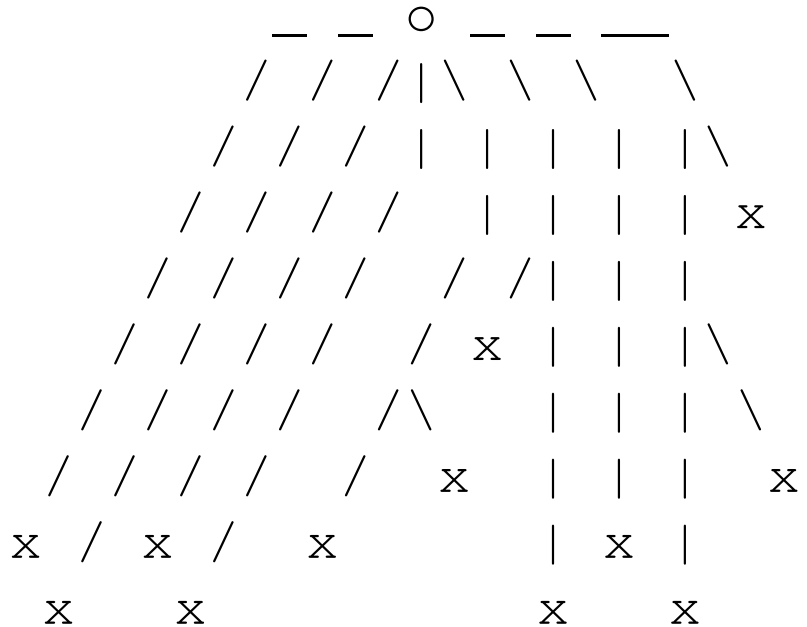



All nodes at a fixed search depth (modulo search extensions/reductions)

# Alternative?

Sample from the space of possibilities

If you get “enough” samples, then you may have a good approximation of the true move values



Some nodes are deep in the tree

# Sampling-Based Decision Making

1. For each move choice at the root node
   - Gather samples
   - Compute performance metric (e.g., average score achieved)

   Repeat until resources are exhausted (usually time)

2. Choose move with the best statistical outcome
   - Poker: betting choice leading to best average winnings
   - Scrabble: move that leads to best average number of points
   - Contract Bridge: move that leads to the most tricks won on average
   - Skat: move that leads to the highest score average

In general: maximize expected payoff

# How Many Samples?

- Iterate until time runs out
- Iterate until statistically confident
  (i.e., until one move choice is significantly better than the alternatives)

Smart sampling can be used to help reduce the number of samples needed to converge to a useful result [4]

# Selective Sampling

We don't want uniform random sequences

(like in simple Monte Carlo sampling for numerical integration where an area is approximated by counting how many samples fall inside and outside)

All scenarios are not equal; use all available information to bias the sampling towards likely scenarios

E.g., in trick-based card games generate card distributions that are consistent with move history and likely — given the move history

# Advantages of Sampling-Based Search

Conceptionally a simple search algorithm

Can realize complex behaviors with no explicit knowledge

$\leadsto$ lessens dependence on expert knowledge

Prefers robust positions with many winning continuations

# Disadvantages of Sampling-Based Search

Problems in tactical situations

- Narrow lines of play are hard to find by randomized search
- May not converge to a winning move at all – if one exists
- Or may not converge to a clear 'winner'

Doesn't approximate mixed strategies out of the box (i.e., compute move probabilities), but see [12]

To get useful results may need to include opponent modeling

- Observe opponents to determine their likely move choices
- Opponent modeling is a hard problem!

# Results?

- World-championship calibre play in Scrabble
- Strong card play in Contract Bridge and Skat
- Near perfection in Backgammon (using millions of rollouts)
- Recent considerable improvement in computer Go and Chess!

# The Rise of Monte Carlo Tree Search (MCTS)

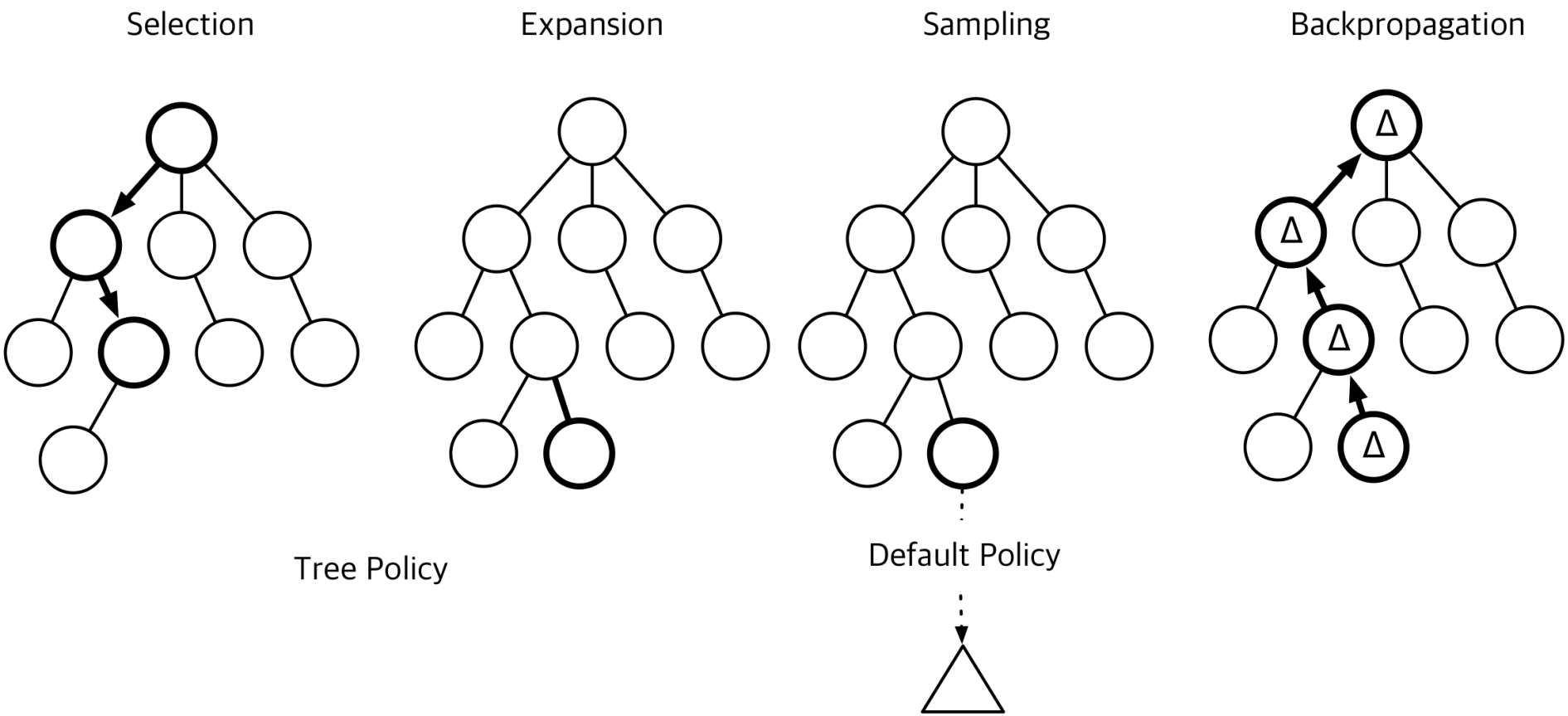
Sampling-based approaches even work in deterministic perfect information games!

Started computer Go revolution in 2006 culminating in AlphaGo [11] which defeated Lee Sedol (one of the strongest 9-dan professional players) 4-1 in March 2016

# Monte Carlo Tree Search

MCTS is an iterative search framework based on Monte Carlo sampling

It maintains a tree of visited game states and result statistics that determine the next move sequence to consider



# Monte Carlo Tree Search (continued)

Each iteration has 4 phases:

- **Selection:** start from root $R$ and select successive child nodes down to a leaf (or leaf predecessor) node $L$, choosing nodes that lets the game tree expand towards most promising regions
- **Expansion:** unless $L$ ends the game with a win/loss for either player, either create one or more child nodes and choose from them node $C$
- **Sampling:** starting in $C$ finish the game (semi)-randomly ("rollout")
- **Backpropagation:** use the rollout result to update information in the nodes on the path from $C$ to $R$

When the search time expires, report the best move at the root

Most commonly the move with the best average value is chosen

Sometimes, the most visited root move is chosen because its evaluation is more stable

# UCT: The First MCTS Algorithm

UCT = “Upper-Confidence Bound for Trees”

Sampling-based method for game tree search

UCT [6] is based on the UCB [5] ("Upper Confident Bound") algorithm for solving multi-armed bandit problems

# UCB Algorithm (Auer et al. 2002, [5])

"Upper Confidence Bound Heuristic" for the multi-armed bandit problem:

> Given $n$ slot machines with random rewards $X_i \in [0, 1]$, find the machine with the highest expected value, while mimizing the regret, by pulling arms

**Regret:** how much we lose while experimenting compared to playing the best move right from the start

E.g., for $T$ trials:

- Psychic policy guesses the best slot machine: expected regret = 0
- Random policy that pulls arms randomly: expected regret = $\Theta(T)$

  [unless all payout distributions have the same mean]

  Any suboptimal pull will incur expected regret $> 0$

# UCB (continued)

UCB solves the so-called **exploration-exploitation** problem for the bandit problem:

> "I know something about the machines already. Should I continue exploiting this knowledge or should I look for something better?"

Define:

- $\hat{X}_i$ = average reward for the $i$-th arm thus far
- $T_i$ = the number of trials for arm $i$
- $T = \sum_i T_i$ = total number of trials

# UCB (continued)

Assuming statistically independent random variables $X_i$ with values in $[0, 1]$, iterate until time runs out:

1. If an arm has not been pulled yet, pull it and observe reward
2. Otherwise, pull an arm that maximizes $\hat{X}_i + C\sqrt{\frac{logT}{T_i}}$ and observe reward

$C > 0$ is the so-called **exploration constant**. The higher it is, the more UCB will explore less frequently visited choices

UCB is optimistic: it selects the arm which it currently thinks can have the highest *potential* reward

# UCB Motivation: Central Limit Theorem

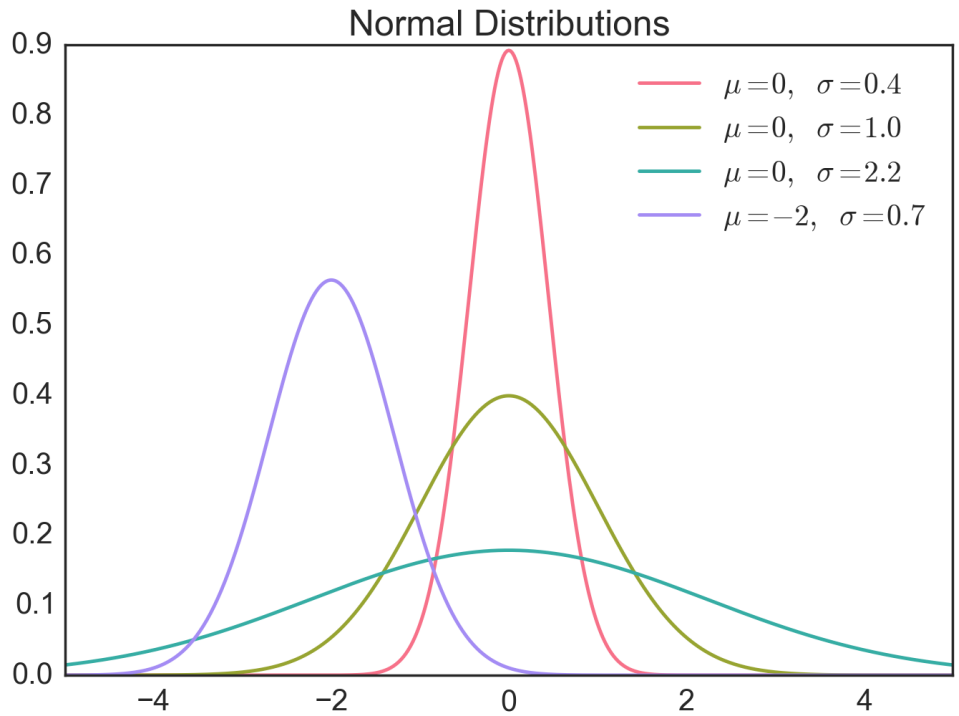
For random variable $X$ with mean value $\mu$ and standard deviation $\sigma < \infty$

$$\hat{X}(N) := (X_1 + X_2 + ... + X_n)/N$$

is approximately normally distributed with mean $\mu$ and variance $\sigma^2/N$

$$\hat{X}(N) \sim \text{Normal}(\mu,\ \sigma^2/N)$$

($\sim$ = distributed like)



# UCB (continued)

Thus, the more we sample, the better $\hat{X}(N)$ approximates $\mu$, because $\sigma^2/N \to 0$ for $N \to \infty$

The estimator's standard deviation ($= \sqrt{\text{Variance}} = \sigma/\sqrt{N}$) measures the estimation uncertainty

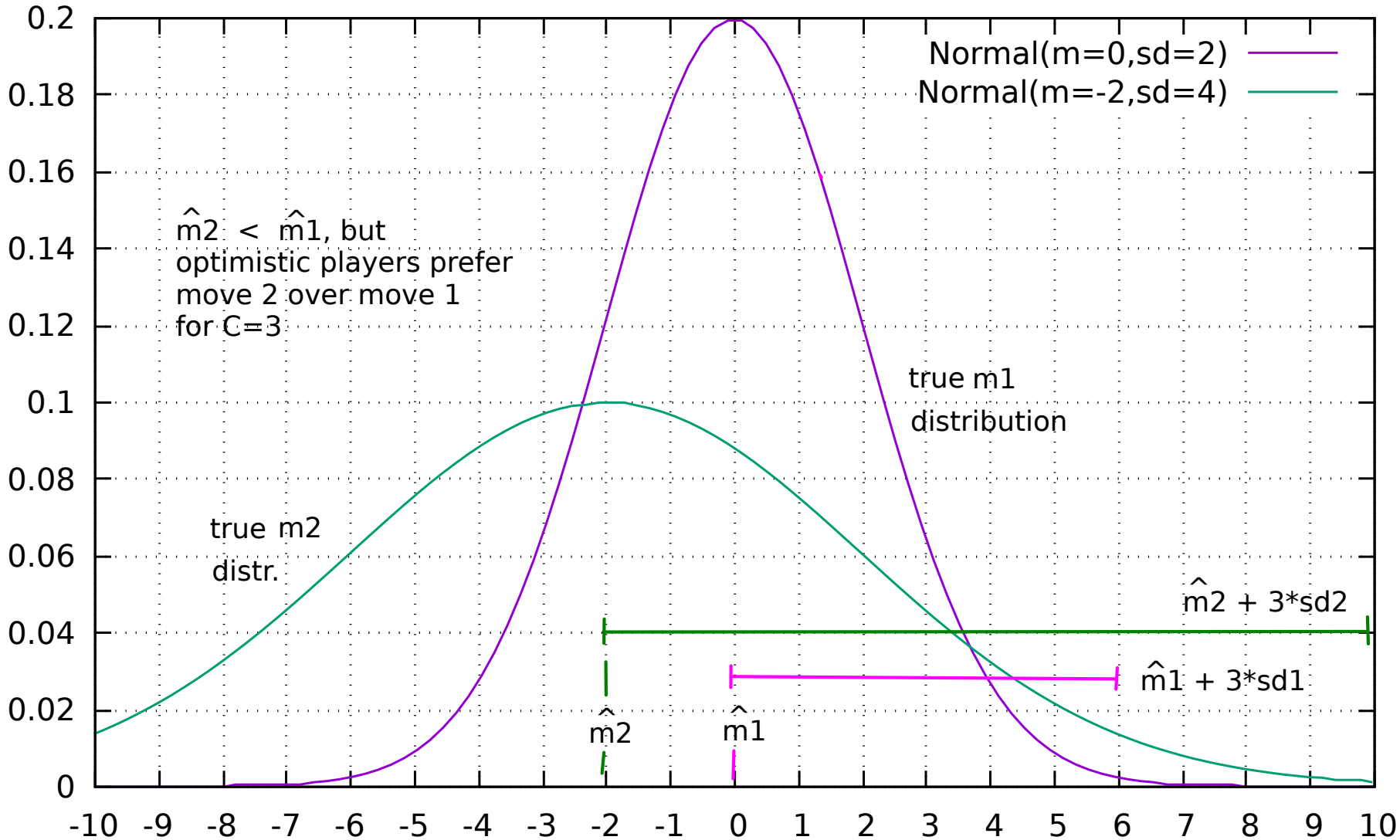
Fact: For the normal distribution with mean $\mu$ and standard deviation $\sigma$ approximately 68% of the probability mass is contained in the interval $[\mu - \sigma, \mu + \sigma]$. In general, mass $p > 0$ is contained in $[\mu - C\sigma, \mu + C\sigma]$ for some $C > 0$ that only depends on $p$

This allows us to control the level of optimism when selecting an action to explore next. The move value

$$\hat{X}(N) + C\sigma/\sqrt{N}$$

represents an optimistic estimate for $\mu$ which we might achieve or exceed with a probability depending only on $C > 0$. $C = 1$, for example, leads to a 16% confidence that we will reach this value or exceed it

# UCB Rule Illustration



# UCB (continued)

In the case of UCB all random variable ranges are assumed to be $[0, 1]$

This allows us to simplify the move value formula a bit so that we do not need to estimate $\sigma_i$ (the standard deviation of payoff $X_i$)

Claim: The variance of any random variable $X \in [0, 1]$ is $\leq 1/4$

Proof: Let $EX$ denote the expected value of $X$ and $VX$ its variance

Because $X \in [0, 1]$, $X^2 \leq X$ and $EX^2 \leq EX =: \mu \in [0, 1]$

Therefore, $VX \stackrel{def.}{=} EX^2 - \mu^2 \leq \mu - \mu^2 = \mu(1 - \mu) \leq 1/4$ □

For arm $i$ in UCB we have: $N = T_i,\ \mu = EX_i,\ \sigma^2 \leq 1/4$,. Therefore,

$$\hat{X}_i + C/\sqrt{T_i}$$

represents an optimistic value arm $i$ might achieve with a probability (roughly) depending on $C$

# UCB (continued)

Auer [5] shows:

- The additional $\sqrt{\log T}$ factor used in step 2 of the UCB rule ensures that UCB never stops pulling any particular arm

  [ if it stops, $C\sqrt{\frac{\log T}{T_i}}$ will eventually exceed the values of the pulled arms, which is a contradiction ]

- In the limit, UCB "finds" an arm with highest expected reward in the sense that it chooses the best arms exponentially more often than others with smaller reward

- UCB's expected regret in the worst case is $\Theta(\log T)$, which can be proven to be optimal and much better than the worst case $\Theta(T)$ "achieved" by the random arm selection procedure

# UCT Search

Applying UCB to game tree search: UCT = UCB on trees

UCT [6] is a MCTS method which regards the game tree as collection of bandit problems — one for each interior node

It builds a tree close to root node and finishes games in a semi-random fashion

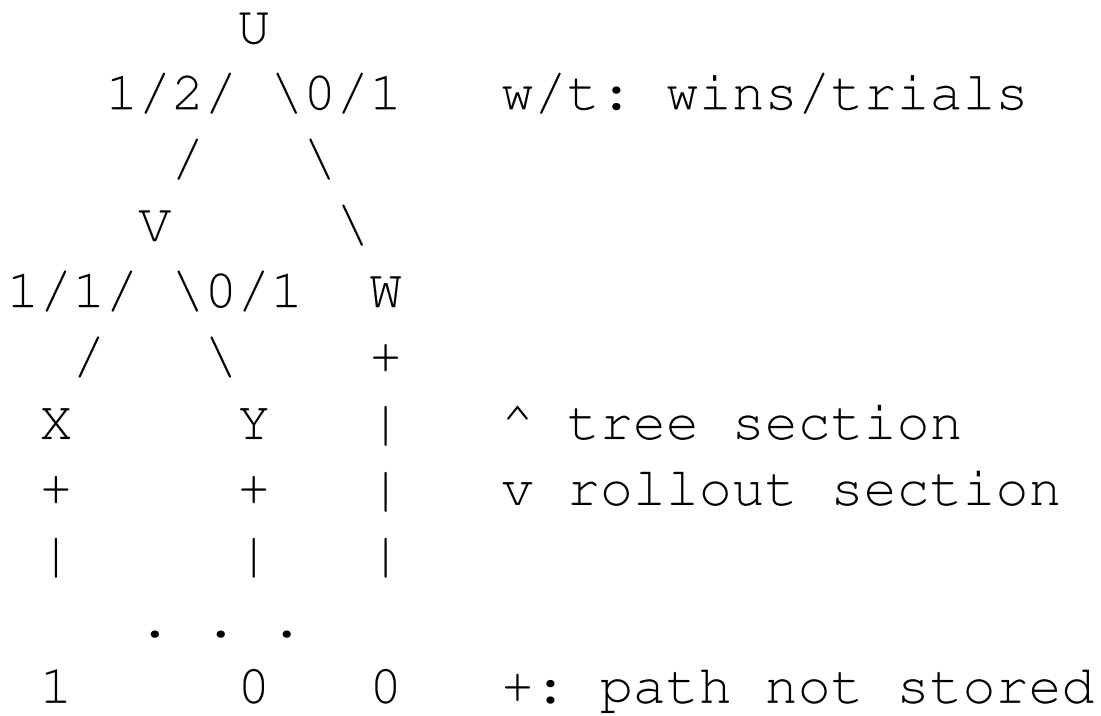
# UCT Search (continued)

- **Selection and Expansion:** In in-tree nodes, use UCB to select the next child. When reaching a node in which not all children have been visited, create the next child and append it to tree
- **Sampling:** Finish game using a (semi-)random rollout policy
- **Backpropagation:** Update value statistics for all visited tree moves

Using this algorithm node values converge to the minimax value [6]

When time runs out, we pick a move with the highest average score (or the move vistited most often for improved robustness)

# UCT Example



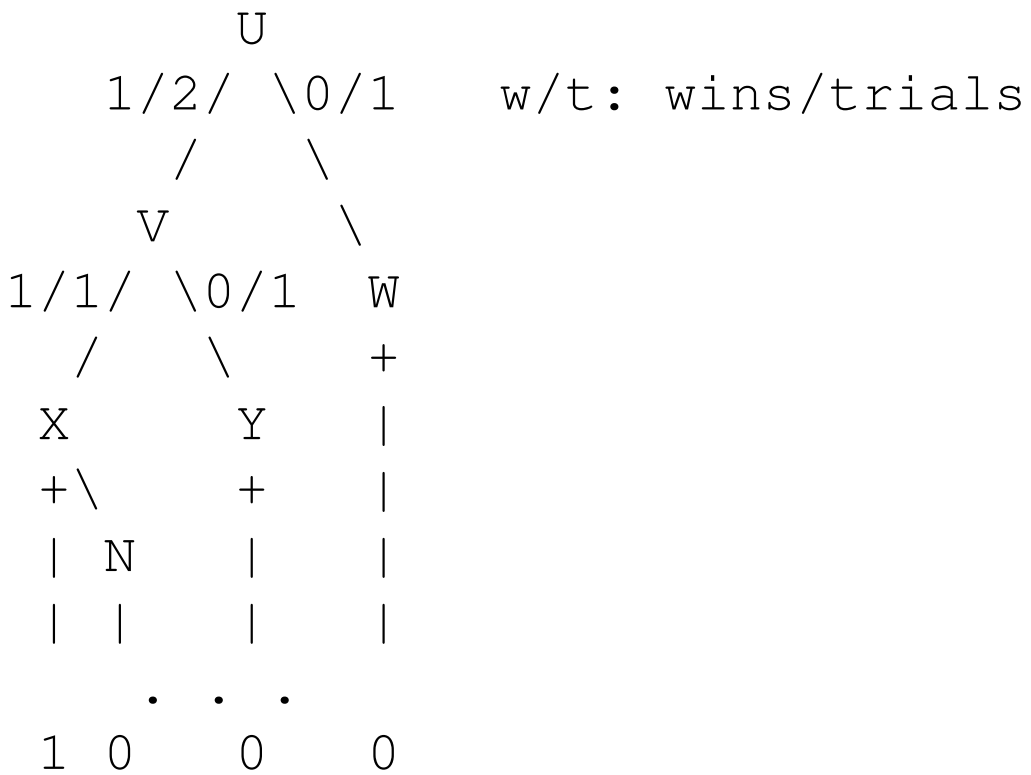

1. Choose move according to UCB starting at U (assume C = 1)

V: $(1/2 + \sqrt{(log3)/2} = 1.24$ or W: $(0/1 + \sqrt{(log3)/1} = 1.05$ ? $\Rightarrow$ go to V

X: $(1/1 + \sqrt{(log2)/1} = 1.83$ or Y: $(0/1 + \sqrt{(log2)/1} = 0.83$ ? $\Rightarrow$ go to X

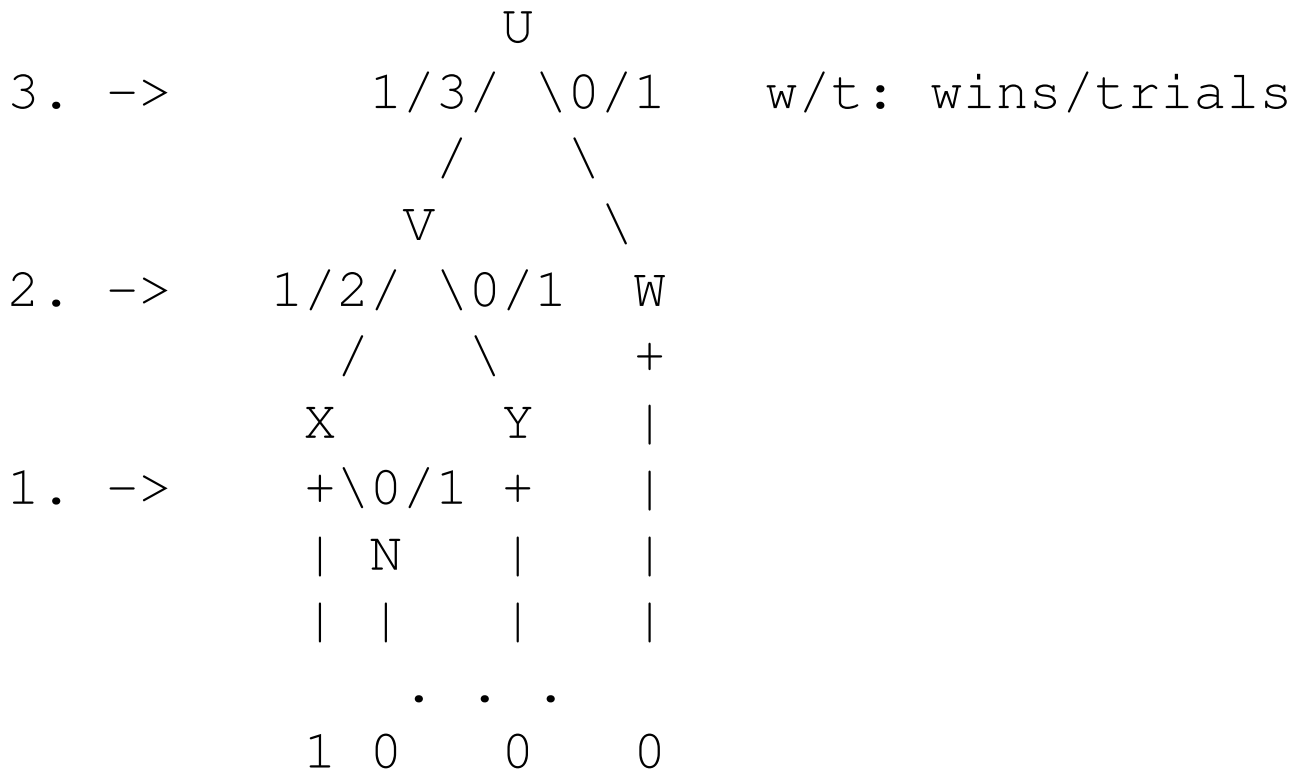
# UCT Example (continued)

2. pick (random) move at X, add node, finish game randomly: result 0



# UCT Example (continued)

3. update values on path to root




repeat ...

# UCT Improvements

In recent years the original UCT search framework has been improved in various ways by using game-specific heuristics such as

- **RAVE (Rapid Action Value Estimation)**

  In games with transpositions it is often the case that values of moves played now or a bit later are highly correlated. UCT can take advantage of this by maintaining statistics for unseen moves based on their performance in subtrees

- **Prior Knowledge**

  The original UCB rule requires us to pull each arm at least once. This can be wasteful in games, especially if the chance of visiting a node again is low. Instead, modern MCTS game programs use policy networks to evaluate all possible moves — pretending they have run a simulation for each child — and then pick the best child for running an actual simulation

# Results

- The development of UCT in 2006 sparked a new area in heuristic search
- MCTS-based programs dominate computer Go and blew away Alpha-Beta search based Go programs. (e.g., 2007-2014 MoGo [7,9], Fuego, Crazy Stone, 2015-17 AlphaGo [11,14])
- AlphaZero-Go and AlphaZero-Chess [14] are now much stronger than any human player. The latest versions don't even use simulations anymore: leaf values are evaluated by a deep neural network
- MCTS has also been successfully applied to other abstract games (Chess, Shogi, Hex, Havannah, Amazons, Poker [12] ...), to single-agent domains, and even to video games [13]

MCTS is the new heuristic search hammer, and we are looking for nails ...

# References

[1] D. Billings et al, Using Probabilistic Knowledge and Simulation to Play Poker , AAAI, pp. 697-703, 1999

[2] M. Ginsberg, GIB: Steps Toward an Expert-Level Bridge-Playing Program. In Proceedings of IJCAI, pp.584-593, 1999

[3] B. Sheppard, "Towards Perfect Play at Scrabble", Ph.D. thesis, 2002

[4] B. Sheppard, Brian Sheppard, World-championship-caliber Scrabble, Artificial Intelligence 134, pp. 241-275, 2002

[5] Auer et al., Finite-time Analysis of the Multiarmed Bandit Problem, Machine Learning, 47, pp. 235-256, 2002

[6] L. Kocsis, C. Szepesvari, "Bandit based Monte-Carlo Planning", ECML 2006, `https://www.researchgate.net/publication/221112399_Bandit_Based_Monte-Carlo_Planning`

# References (continued)

[7] S. Gelly, Y. Wang, “Exploration Exploitation in Go: UCT for Monte-Carlo Go”, https://www.researchgate.net/publication/228699422_Exploration_exploitation_in_Go_UCT_for_Monte-Carlo_Go

[8] M. Buro, J.R. Long, T. Furtak, and N. Sturtevant, Improving State Evaluation, Inference, and Search in Trick-Based Card Games, IJCAI, Pasadena USA, pp. 1407-1413, 2009

[9] S. Gelly and D. Silver, Combining online and offline knowledge in UCT. In Z. Ghahramani editor, ICML volume 227 of ACM International Conference Proceedings Series, pp 273-280, 2007

[11] D. Silver et al.: Mastering the game of Go with deep neural networks and tree search, Nature, VOL 529, pp. 484-501, 2016

[12] J. Heinrich and D. Silver, Smooth UCT Search in Computer Poker, AAAI 2015

# References (continued)

[13] D. Churchill and M. Buro, Portfolio Greedy Search and Simulation for Large-Scale Combat in Starcraft, CIG, Niagara Falls, Canada, 2013

[14] D. Silver et al., Mastering the game of Go without human knowledge, Nature, Vol. 550, pp. 354-359, 2017

FIN