Lecture 13: Monte Carlo Tree Search

Artificial Intelligence
CS-GY-6613
Julian Togelius
julian.togelius@nyu.edu

Let’s Go!

Rules

- Players take turns to place stones at intersections
- Every stone remaining on the board must have at least one open "point" (an intersection, called a "liberty") directly next to it (up, down, left, or right)
- The stones on the board must never repeat a previous position of stones

Branching factors

- Chess: ~35
- Go: ~350
- Hex: ~100
- Arimaa: ~13000
- Pac-Man: 4
- Halo: ...infinite?
- Most games: lots

Heuristic evaluation functions

- Chess: count the number of white vs black pieces, who holds key positions etc.
- Go: ???
- Pac-Man: number of pills eaten, proximity to ghosts?
- Halo: ???

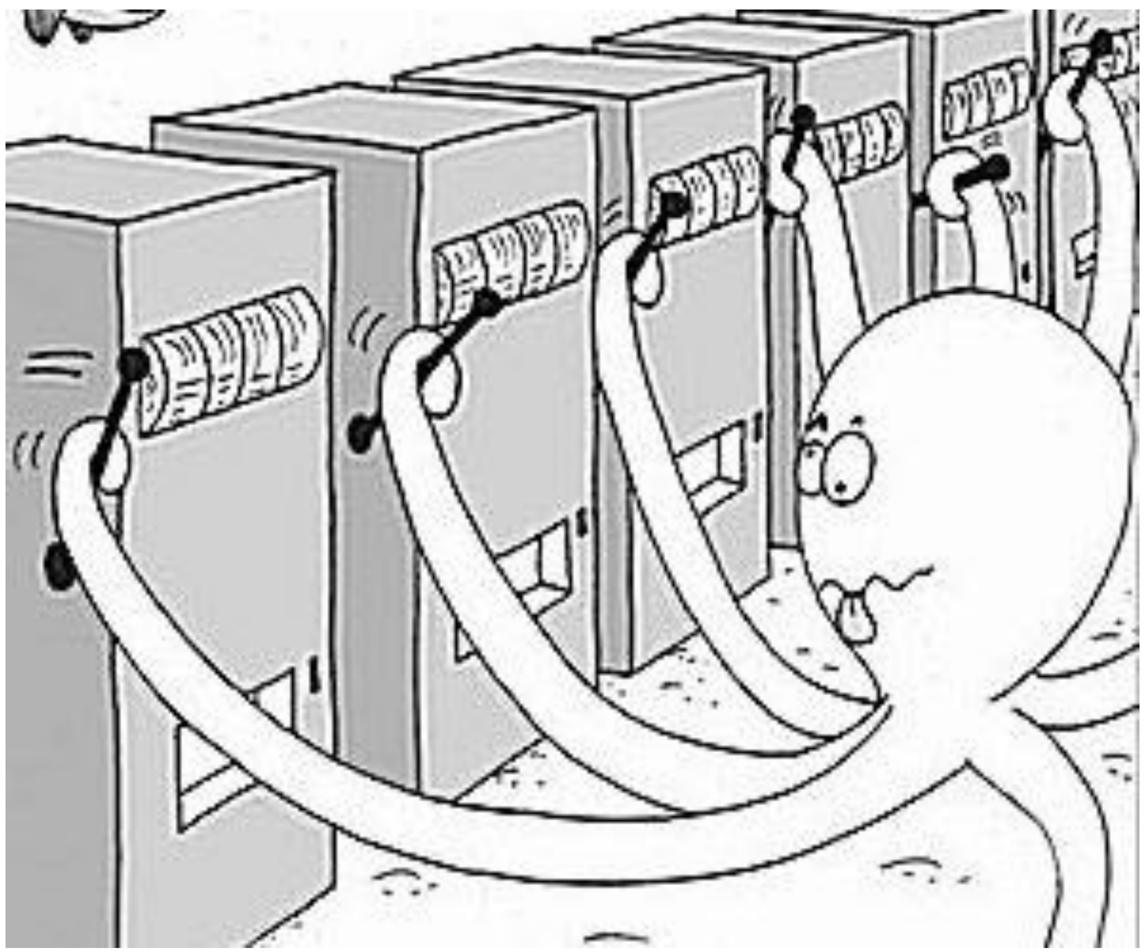
Go MCTS!

Year	Program	Description	Elo
2006	INDIGO	Pattern database, Monte Carlo simulation	1400
2006	GNU GO	Pattern database, α-β search	1800
2006	MANY FACES	Pattern database, α-β search	1800
2006	NEUROGO	TDL, neural network	1850
2007	RLGO	TD search	2100
2007	MOGO	MCTS with RAVE	2500
2007	CRAZY STONE	MCTS with RAVE	2500
2008	FUEGO	MCTS with RAVE	2700
2010	MANY FACES	MCTS with RAVE	2700
2010	ZEN	MCTS with RAVE	2700

What is MCTS?

- A way of selecting the next action
- A statistical tree-search method with rollouts rather than function evaluations
- Builds unbalanced trees

Bandit problems



- At each step, pull one arm
- Noisy/random reward signal in the range [0..1]
- Different average reward
- Task: maximise reward (minimise regret)

Which arm to pull?

Which arm to pull?

- Pull all arms equally often?
- Only pull the arm that has given the best results so far?
- Mostly pull the “best” arm, but sometimes one of the others?
- An example of the exploration/exploitation dilemma
- Principled solution?

Which arm to pull?

Flat Monte Carlo
Share trials uniformly between arms

***ε*-Greedy**
P(1- *ε*) – Best arm so far
P(*ε*) – Random arm

UCB1 (Auer et al (2002)).
Choose arm *j* so as to maximise:

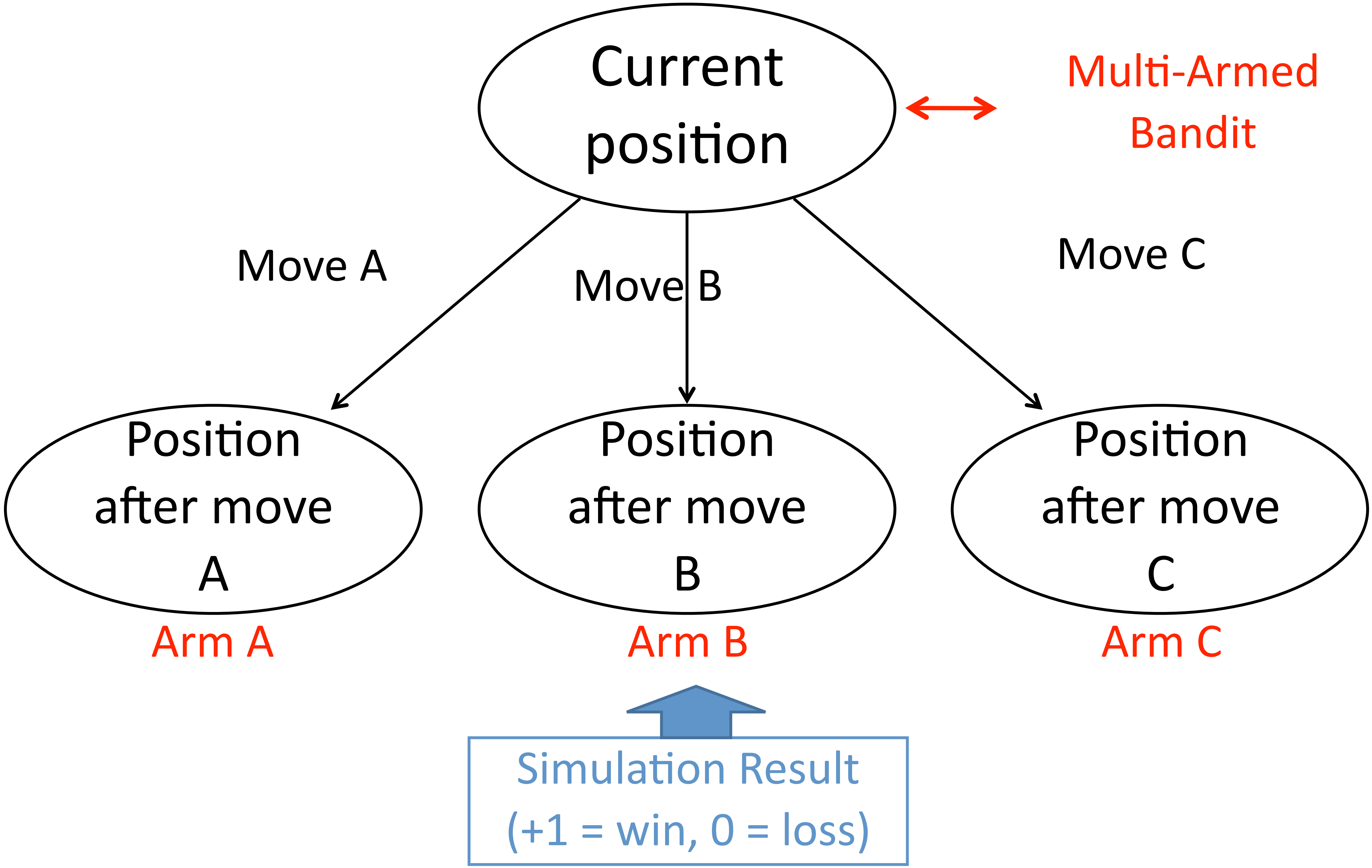
$$\bar{X}_j + \sqrt{\frac{2 \log n}{T_j(n)}}$$

Mean so far

Upper bound on variance

n = number of plays so far
Tj(n) = number of times arm j was pulled

Game Decisions



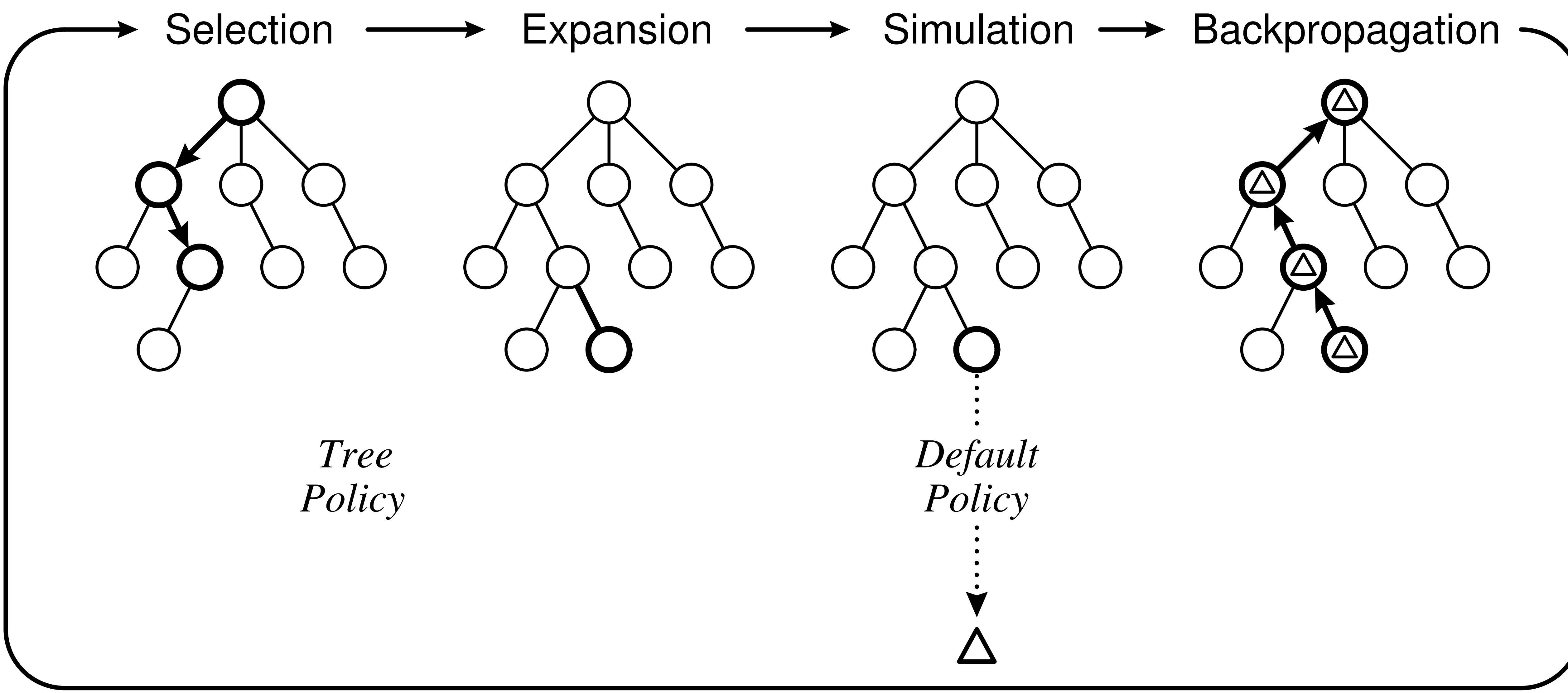
UCB

- Anytime – stop whenever you like
- UCB1 formula minimises regret
- Grows like log(n)
- Needs only game rules:
  - Move generation
  - Terminal state evaluation
- Surprisingly effective, but…
  … doesn’t look ahead

UCT (UCB on trees)

- Anytime
- Scalable
- Tackle complex games better than before
- May be logarithmically better with increased CPU
- No need for heuristic function
- Though often better with one

MCTS general idea



MCTS algorithm

- Tree policy
  - Expand
  - Best Child (UCT Formula)
- Default Policy
- Back-propagate

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

Tree policy

```
function TreePolicy(v)
    while v is nonterminal do
        if v not fully expanded then
            return Expand(v)
        else
            v ← BestChild(v, Cp)
    return v
```

Note that node selected for expansion does not need to be a leaf of the tree (the nonterminal test refers to the game state)

Tree expansion

function $\textsc{Expand}(v)$
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

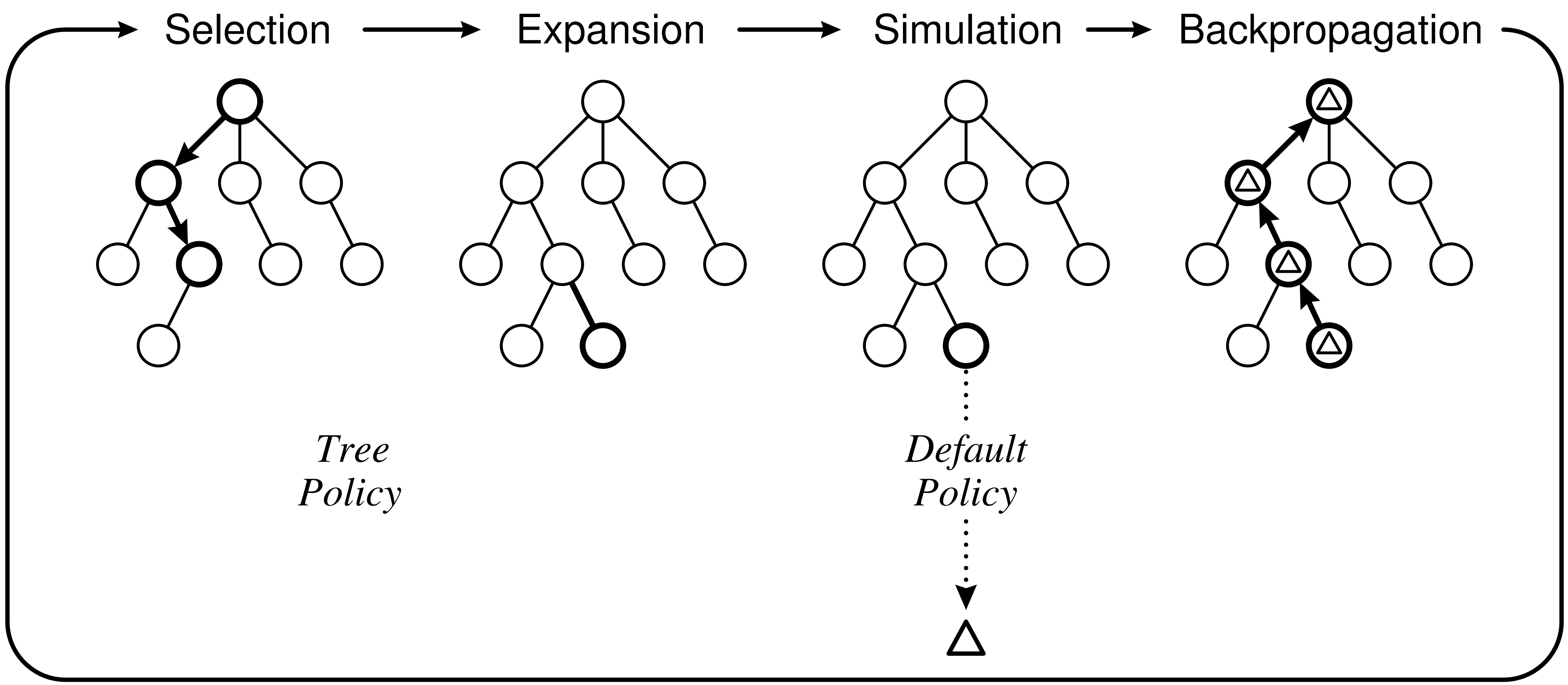
Best child (UCT)

$$\textbf{function } \textsc{BestChild}(v, c)$$

$$\textbf{return} \quad \mathop{\arg\max}_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

- Standard UCT equation (compare UCB)
- Higher values of c lead to more exploration

MCTS general idea



- Tree policy: choose which node to expand (not necessarily leaf of tree)

- Default (simulation) policy: random playout until end of game

Default policy (rollout)

function $\textsc{DefaultPolicy}(s)$
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

- Each time a node is added to the tree, the default policy plays out until the terminal state of the game
- The standard is to do this uniformly randomly

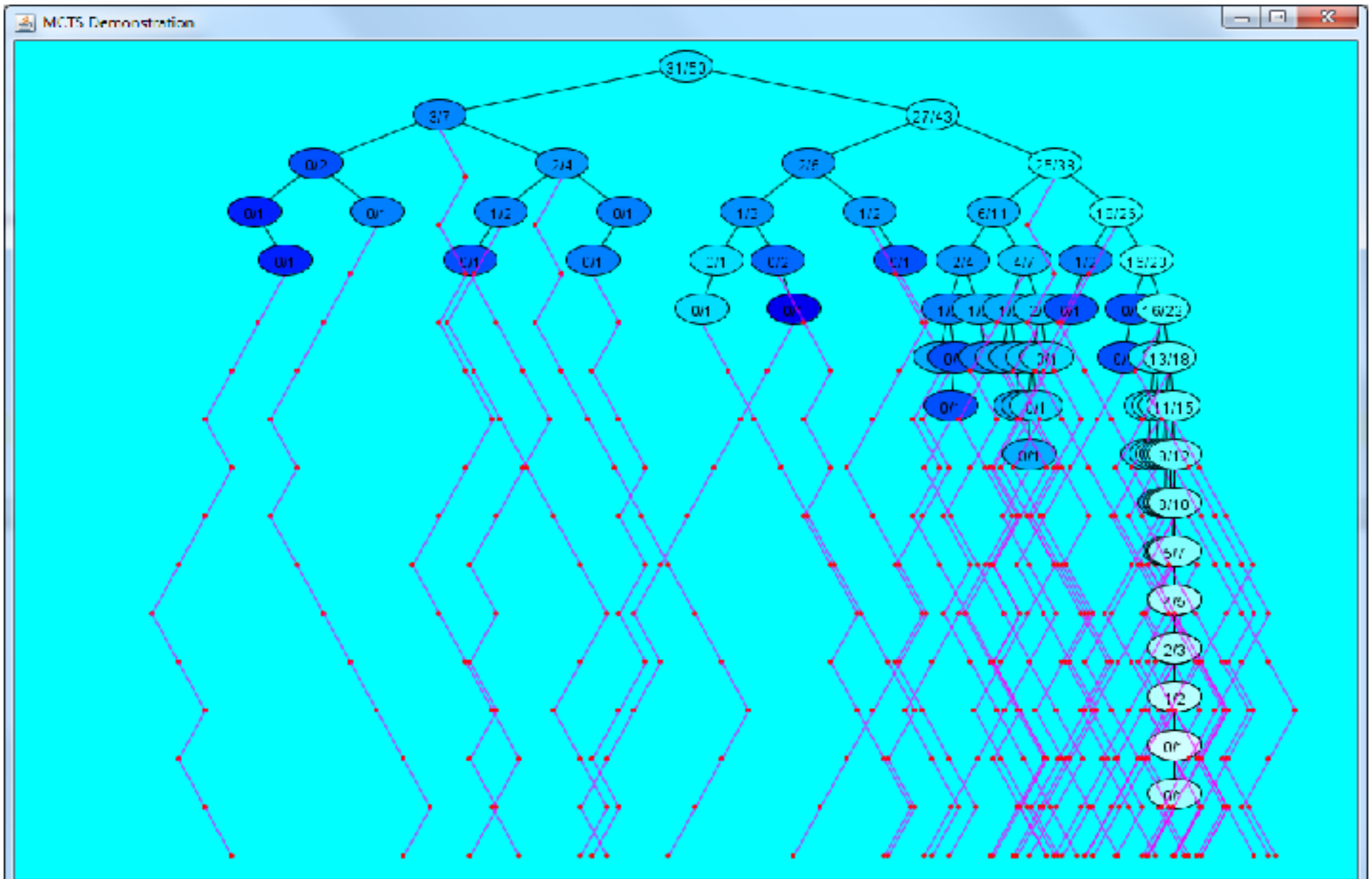
Backup

$$
\begin{array}{l}
\textbf{function } \textsc{Backup}(v, \Delta) \\
\quad \textbf{while } v \text{ is not null } \textbf{do} \\
\quad\quad N(v) \leftarrow N(v) + 1 \\
\quad\quad Q(v) \leftarrow Q(v) + \Delta(v, p) \\
\quad\quad v \leftarrow \text{parent of } v
\end{array}
$$

- v is the new node added to the tree by the tree policy

- Back up the values from the added node up the tree to the root

MCTS builds asymmetric trees



Enhancements

- Selection/expansion
  - All moves as first (AMAF) / RAVE
  - Bandit enhancements
  - Parameter tuning
- Simulation enhancements
  - Make the simulation more realistic (often costly and results may vary)

Non-determinism and incomplete information

- Many games are not deterministic
- Many games are only partially observable
- Determinization: create separate nodes for each random outcome
    - Huge branching factor
- Cheat?