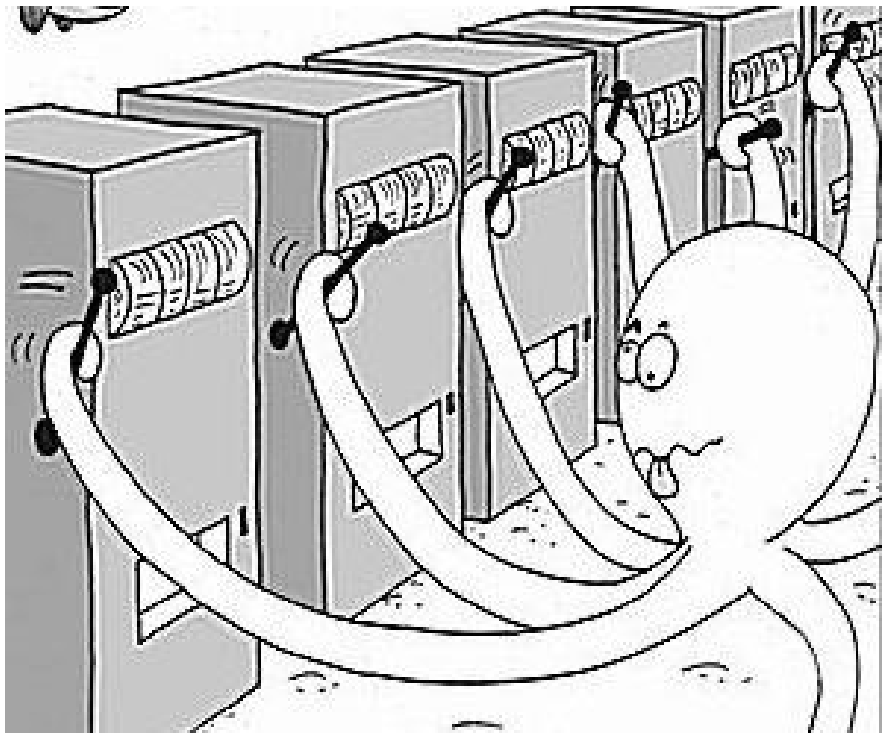
Monte Carlo Tree Search

(based on a tutorial by Simon Lucas and Peter Cowling)

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

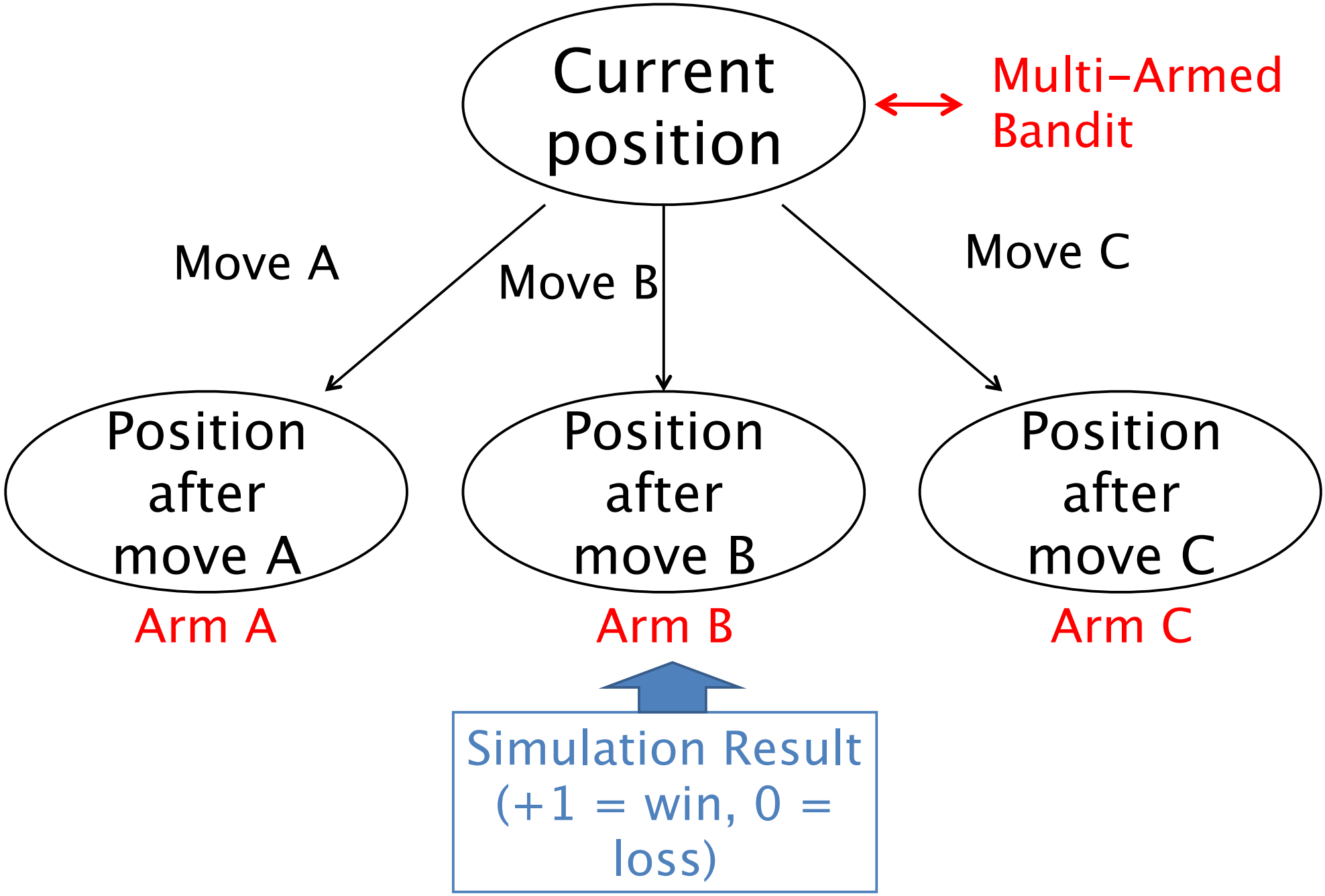
n = number of plays so far

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

- Monte Carlo Tree Search (MCTS),
- Upper Confidence Bounds (UCB),
- Upper Confidence Bounds for Trees (UCT)

- From making single decision to making a sequence of decisions with dependencies

Game decisions



UCB

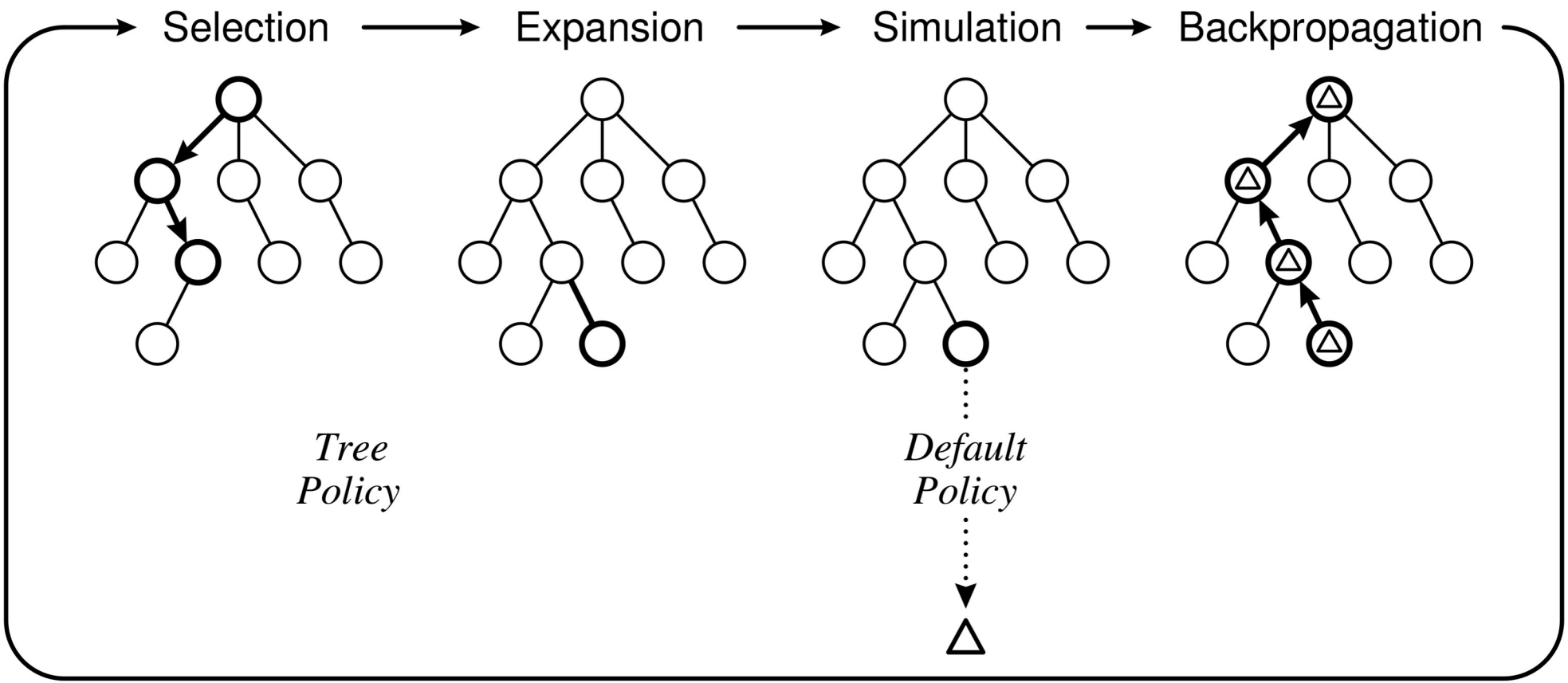
- Anytime – stop whenever you like
- UCB1 formula minimises regret
- Grows like log(n)
- Needs only game rules:
  - Move generation
  - Terminal state evaluation
- Surprisingly effective, but…
  … doesn't look ahead

- Monte Carlo Tree Search (MCTS),
- Upper Confidence Bounds (UCB),
- Upper Confidence Bounds for Trees (UCT)

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



- Tree policy: choose which node to expand (not necessarily leaf of tree)
- Default (simulation) policy: random playout until end of game

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 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'

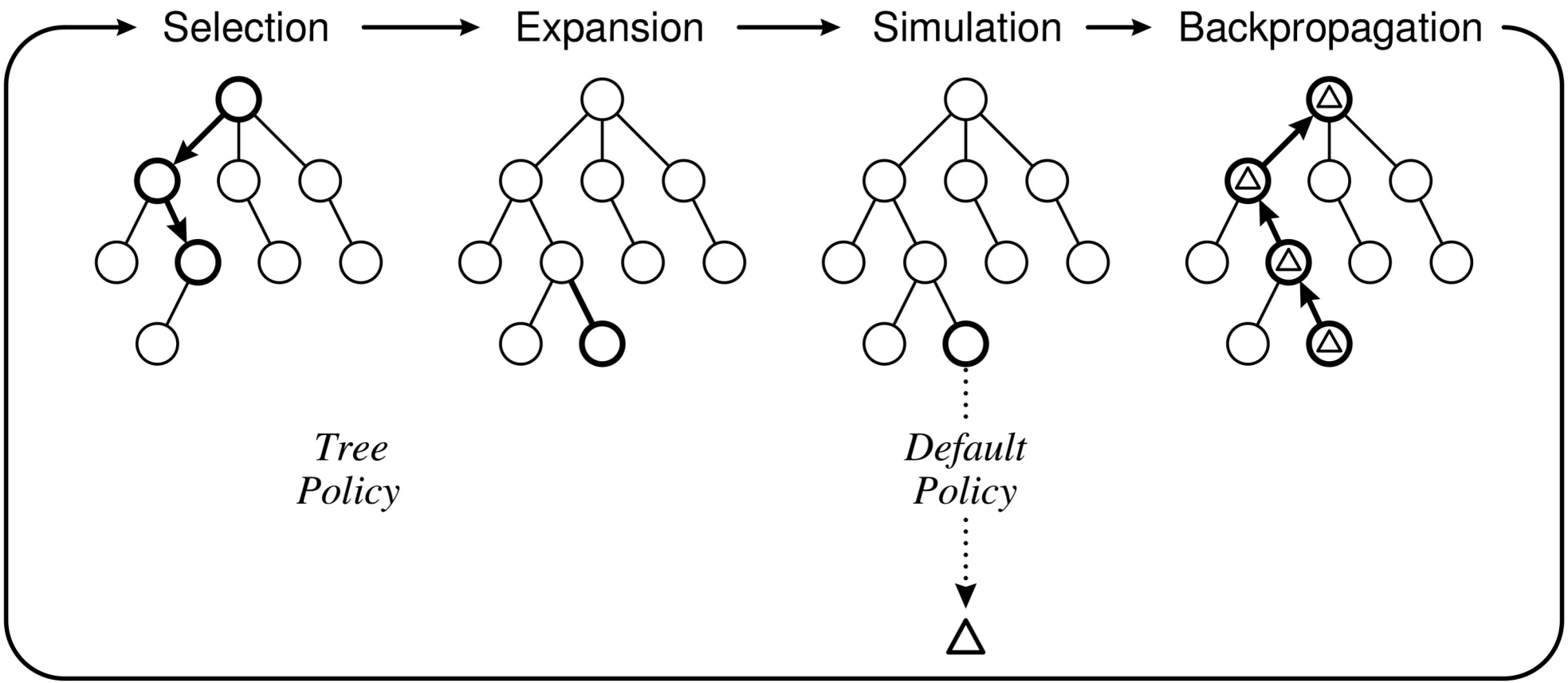
Best child (UCT)

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

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

- 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 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

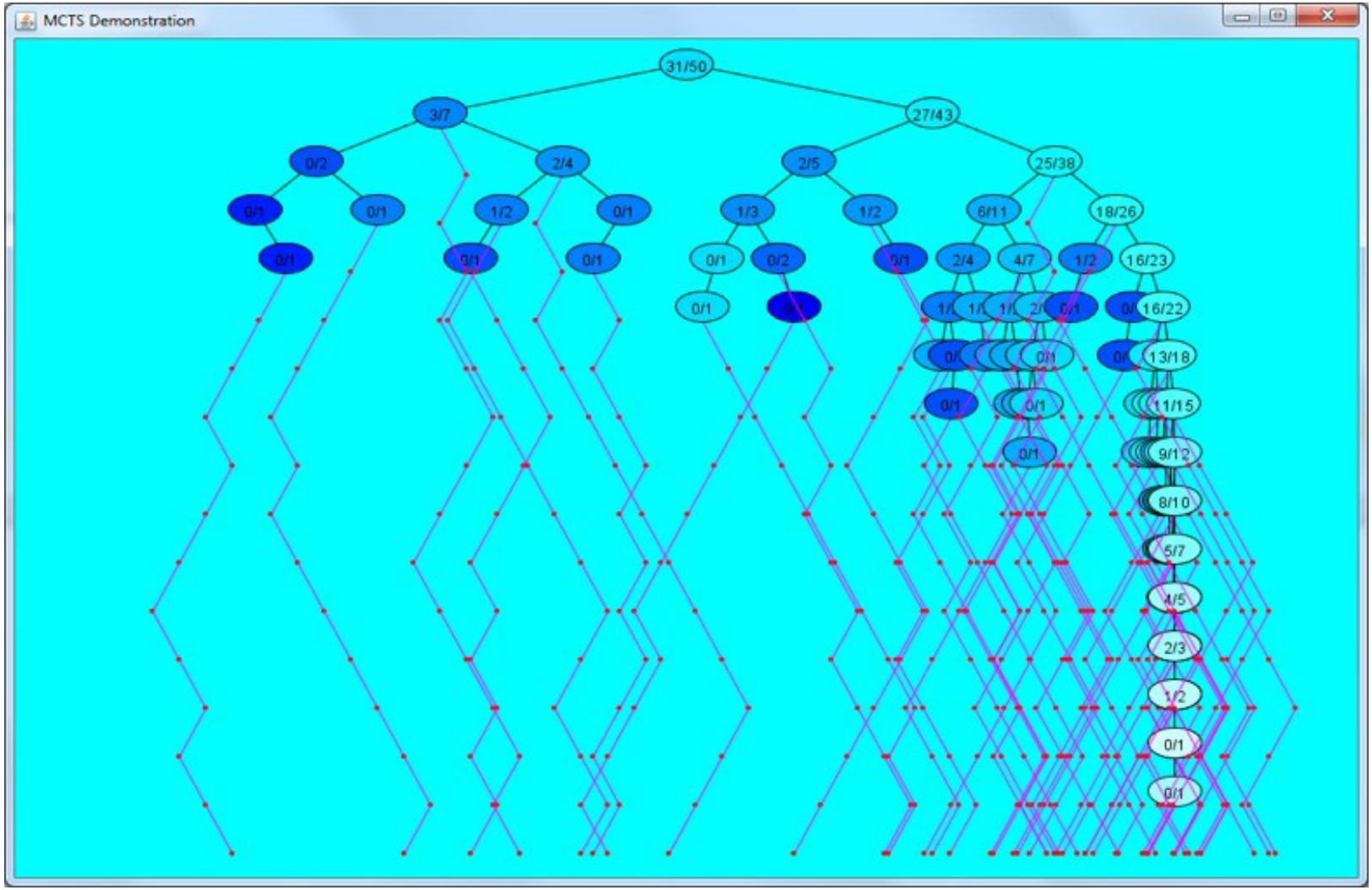
- Each time a node is added to the tree, the default policy plays out until the terminal state of the game
- The standard is to do this uniformly randomly

Backup

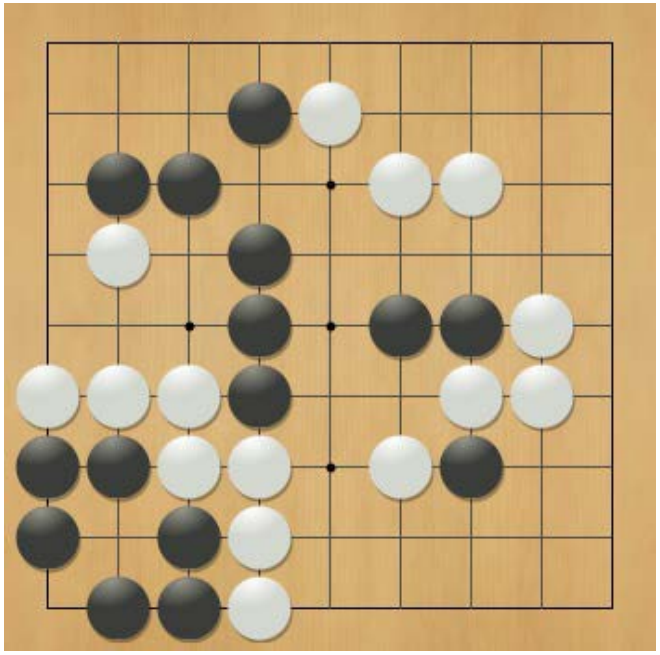
function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

- 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

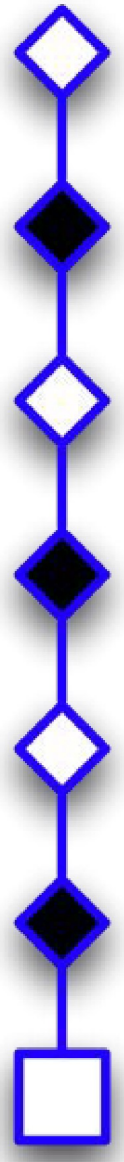


Computer Go



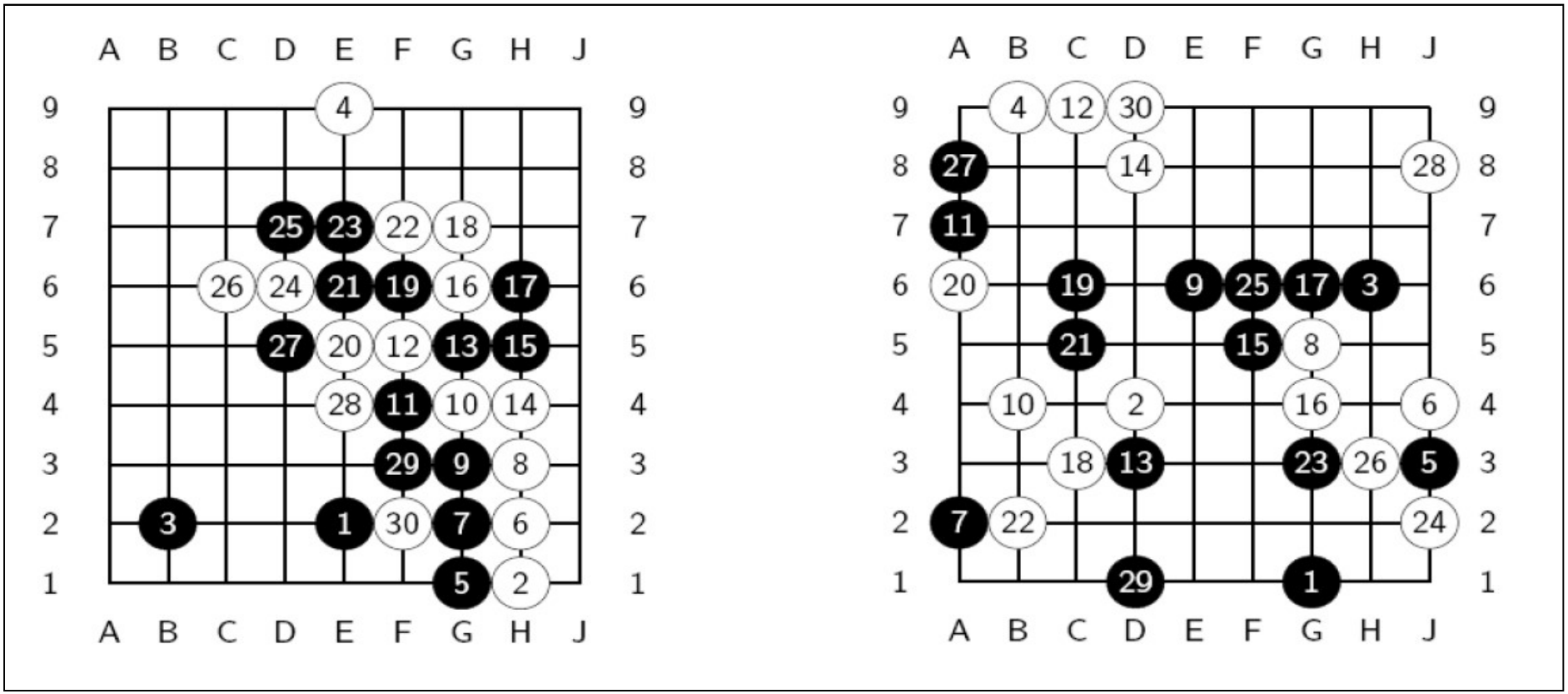
Default Policy

- The default policy is crucial to UCT
- Better default policy => better UCT (?)
- As hard as the overall problem
- Default policy must also be fast

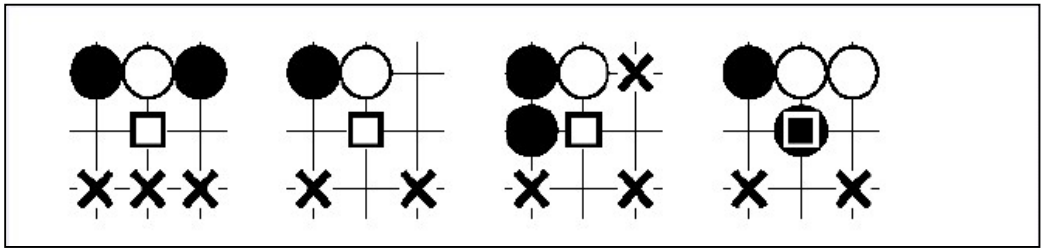


Educated simulations:
Sequence-like simulations

Sequences matter!



How it works in MoGo

- Look at the 8 intersections around the previous move
- For each such intersection, check the match of a pattern (including symetries)
- If at least one  matching intersections;
- Else play uniformly among legal moves

Random mode	Win. Rate for B. Games	Win. rate for W. Games	Total Win. Rate
Uniform	46% (250)	36% (250)	$41.2\% \pm 2.2\%$
Sequence-like	77% (400)	82% (400)	$80\% \pm 1.4\%$

Default Policy (continued)

- The default policy is crucial to UCT
- **Better default policy => better UCT (?)**
- As hard as the overall problem
- Default policy must also be fast