Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Monte Carlo Tree Search

CMU 10-703

Katerina Fragkiadaki

Part of slides inspired by Sebag, Gaudel

Definitions

Learning: the acquisition of knowledge or skills through experience, study, or by being taught.

Planning: any computational process that uses a model to create or improve a policy

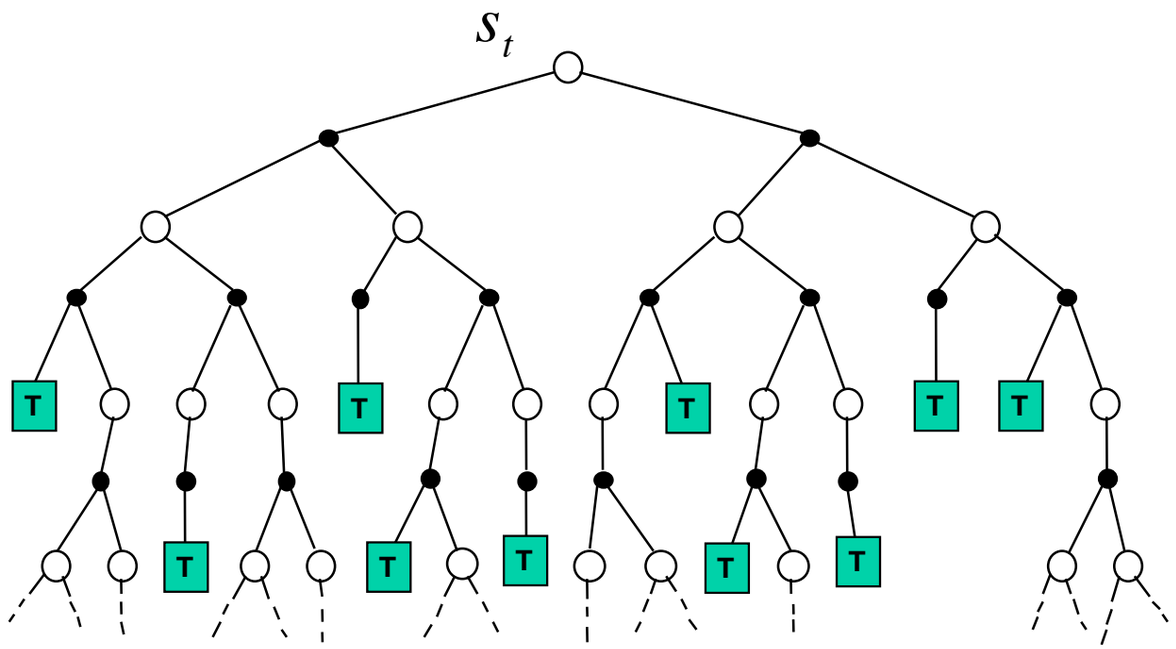
This lecture

Computing value functions combining learning and planning using Monte Carlo Tree Search

Computing value functions combining learning and planning in other ways will be revisited in later lectures

Online Planning with Search

1. Build a search tree **with the current state of the agent** at the root
2. Compute value functions using simulated episodes (reward usually only on final state, e.g., win or loose)
3. Select the next move to execute
4. Execute it
5. GOTO 1



Why online planning?

Why don't we *just* learn a value function directly for every state offline, so that we do not waste time online?

- Because the environment has many many states (consider Go 10^170, Chess 10^48, real world ….)
- Very hard to compute a good value function for each one of them, most you will never visit
- Thus, condition on the current state you are in, try to estimate the value function of the relevant part of the state space online
- Focus your resources on sub-MDP starting from now, often dramatically easier than solving the whole MDP

Curse of dimensionality

- The sub-MDP rooted at the current state the agent is in may still be very large (too many states are reachable), despite much smaller than the original one.
- Too many actions possible: large tree branching factor
- Too many steps: large tree depth

I cannot exhaustively search the full tree

Curse of dimensionality

Consider hex on an NxN board.

branching factor ≤ N^2

2N ≤ depth ≤ N^2

board size	max branching factor	min depth	tree size	depth of 10^{10} nodes
6x6	36	12	$>10^{17}$	7
8x8	64	16	$>10^{28}$	6
11x11	121	22	$>10^{44}$	5
19x19	361	38	$>10^{96}$	4

Goal of HEX: to make a connected line that links two antipodal points of the grid

How to handle the curse of dimensionality?

Intelligent instead of exhaustive search

1. The depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function v(s)=v*(s) that predicts the outcome from state s.
2. The breadth of the search may be reduced by sampling actions from a policy p(a|s), that is, a probability distribution over plausible moves a in position s, instead of trying every action.

Position evaluation

We can estimate values for states in two ways:

- Engineering them using human experts (DeepBlue)
- Learning them from self-play (TD-gammon)

Problems with human engineering:

- tiring
- non transferrable to other domains.

YET: that's how Kasparov was first beaten.

http://stanford.edu/~cpiech/cs221/apps/deepBlue.html

Monte-Carlo position evaluation

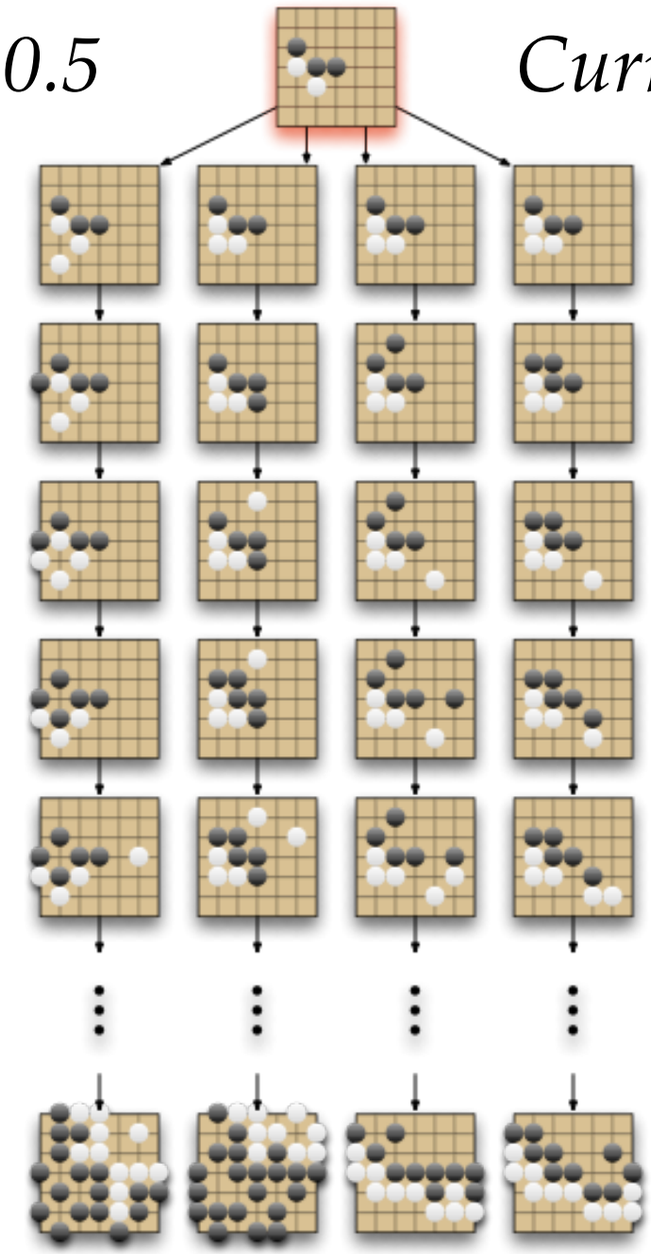
```
function MC_BoardEval(state):
    wins = 0
    losses = 0
    for i=1:NUM_SAMPLES
        next_state = state
        while non_terminal(next_state):
            next_state = random_legal_move(next_state)
        if next_state.winner == state.turn: wins++
        else: losses++ #needs slight modification if draws possible
    return (wins - losses) / (wins + losses)
```

What policy shall we use to draw our simulations?

The cheapest one is random..

Monte-Carlo position evaluation in Go

$V(s) = 2/4 = 0.5$



Current position s

Simulation

1 1 0 0

Outcomes

Averaging sampled returns..

Simplest Monte-Carlo Search

- For action selection, I need to be estimating not state but rather state-action values.
- But! Since we assume dynamics given, we can simply use one step look-ahead!

Simplest Monte-Carlo Search

Given a deterministic transition function T, a root state s and a simulation policy π (potentially random)

For each action $a \in \mathcal{A}$

$$Q(s,a) = \text{MC-boardEval}(s'), \quad s' = \mathrm{T}(s,a)$$

Select root action: $a = \mathrm{argmax}_{a \in \mathcal{A}} Q(s,a)$

Simplest Monte-Carlo Search

Given a deterministic transition function T, a root state s and a simulation policy π (potentially random)

Simulate K episodes from current (real) state:

$$\{s, a, R_1^k, S_1^k, A_1^k, R_2^k, S_2^k, A_2^k, \ldots, S_T^k\}_{k=1}^K \sim \mathrm{T}, \pi$$

Evaluate action value function of the root by mean return:

$$Q(s,a) = \frac{1}{K} \sum_{k=1}^{K} G_k \overset{\mathrm{P}}{\rightarrow} q_\pi(s,a)$$

Select root action: $a = \mathrm{argmax}_{a \in \mathcal{A}} Q(s,a)$

https://en.wikipedia.org/wiki/Convergence_of_random_variables

Can we do better?

- Could we be **improving our simulation policy** the more simulations we obtain?
- Yes we can! We can have two policies:
  1. Internal to the tree: keep track of action values Q not only for the root but also for nodes internal to a tree we are expanding, and use ϵ – greedy(Q) to improve the simulation policy over time
  2. External to the tree: we do not have Q estimates and thus we use a random policy

In MCTS, the simulation policy improves

- Any better ideas for the simulation policy?

Upper Confidence Bound (UCB)

$$A_t \sim \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

t : parent node visits

$N_t(a)$: parent node visits

- Probability of choosing an action:
  - decreases with the number of visits (explore)
  - increases with a node's value (exploit)
- Always tries every option once.

A better exploration-exploitation than ϵ — greedy

Finite-time Analysis of the Multiarmed Bandit Problem, Auer, Cesa-Bianchi, Fischer, 2002

Monte-Carlo Tree Search

1. **Selection**
   - Used for nodes we have seen before
   - Pick according to UCB
2. **Expansion**
   - Used when we reach the frontier
   - Add one node per playout
3. **Simulation**
   - Used beyond the search frontier
   - Don’t bother with UCB, just play randomly
4. **Backpropagation**
   - After reaching a terminal node
   - Update value and visits for states expanded in selection and expansion

Bandit based Monte-Carlo Planning, Kocsis and Szepesvari, 2006

Monte-Carlo Tree Search

Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
    winner = MCTS_sample(next_state)
else:
    if some children of state expanded:
        next_state = expand(random unexpanded child)
    else:
        next_state = state
    winner = random_playout(next_state)
update_value(state, winner)
```

For every state within the search tree we bookkeep # of visits and # of wins

Monte-Carlo Tree Search

Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
    winner = MCTS_sample(next_state)
else:
    if some children of state expanded:
        next_state = expand(random unexpanded child)
    else:
        next_state = state
    winner = random_playout(next_state)
update_value(state, winner)
```

Monte-Carlo Tree Search

MCTS helper functions

Sample actions based on UCB score

```
function UCB_sample(state):
    weights = []
    for child of state:
        w = child.value + C * sqrt(ln(state.visits) / child.visits)
        weights.append(w)
    distribution = [w / sum(weights) for w in weights]
    return child sampled according to distribution
```

(unrolling)

```
function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```

Monte-Carlo Tree Search

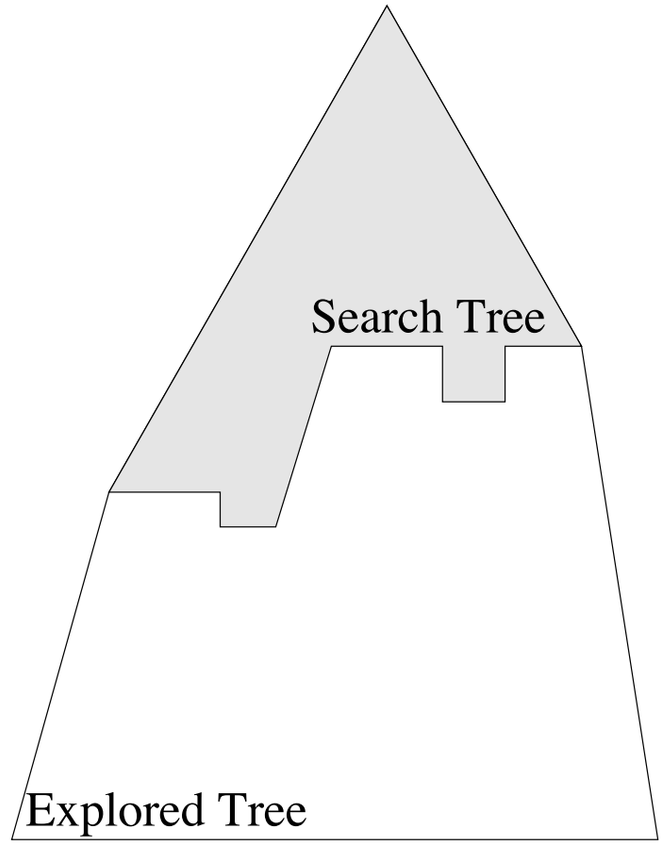
MCTS helper functions

```
function expand(state):
    state.visits = 1
    state.value = 0
```

```
function update value(state, winner):

    if winner == state.turn:
        state.value += 1
    else:
        state.value -= 1
```

Basic MCTS pseudocode

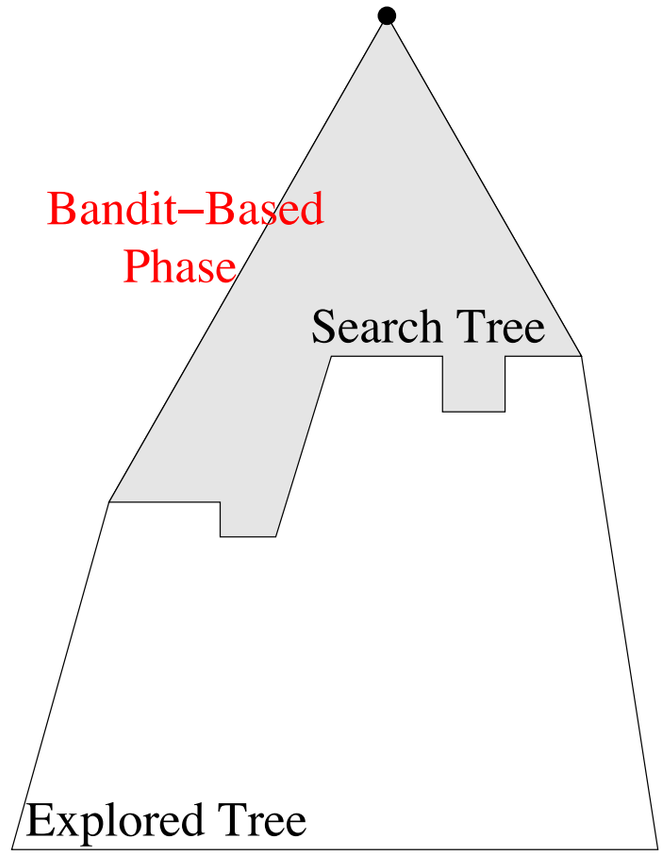
```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```



Seacrh tree contains states whose all children have been tried at least once

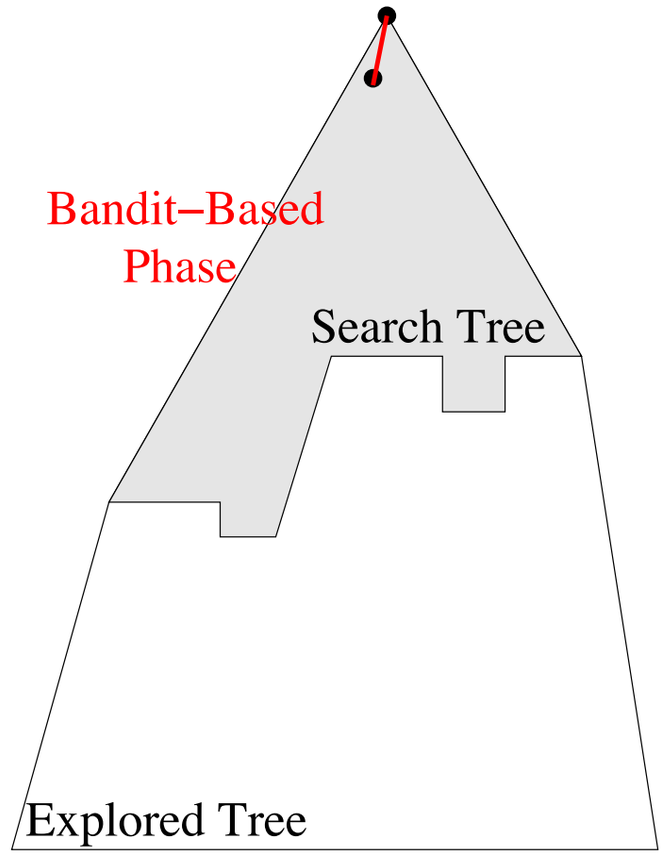
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```



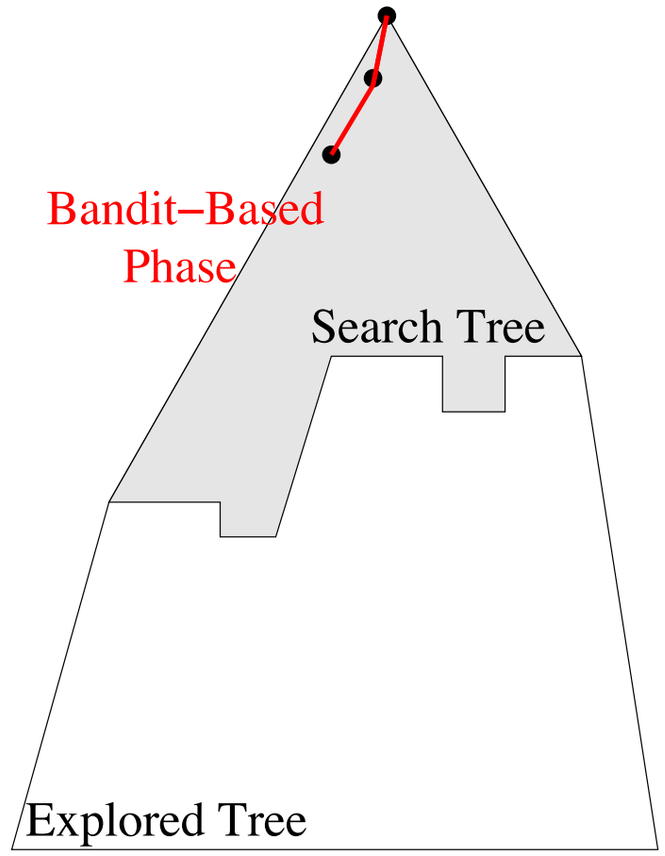
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```



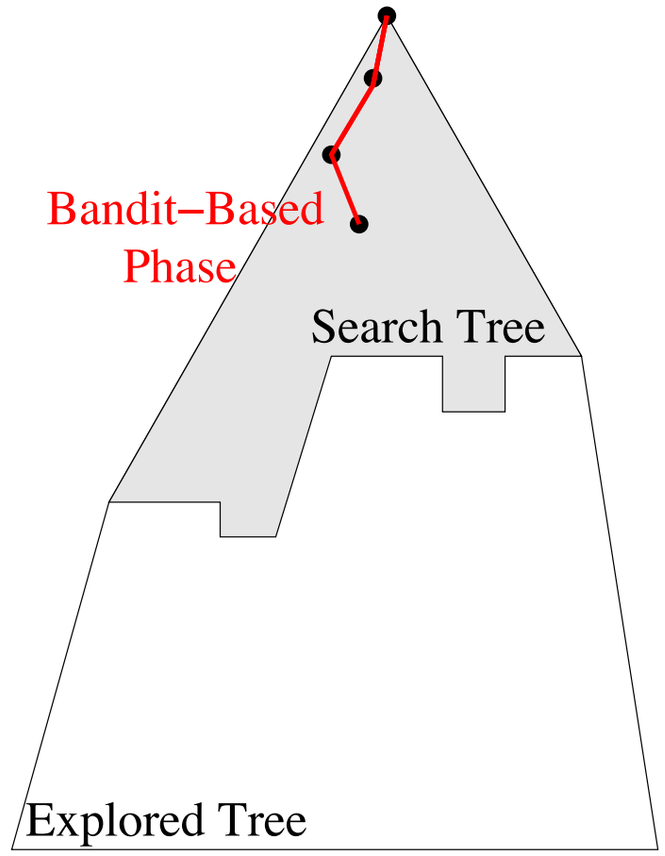
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```



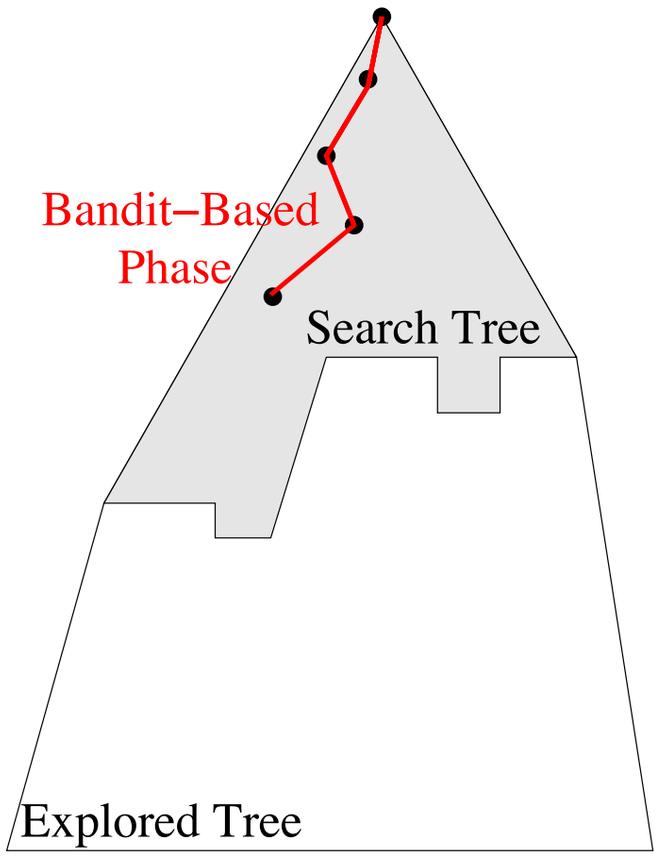
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
else:
    if some children of state expanded:
        next_state = expand(random unexpanded child)
    else:
        next_state = state
    winner = random_playout(next_state)
update_value(state, winner)
```



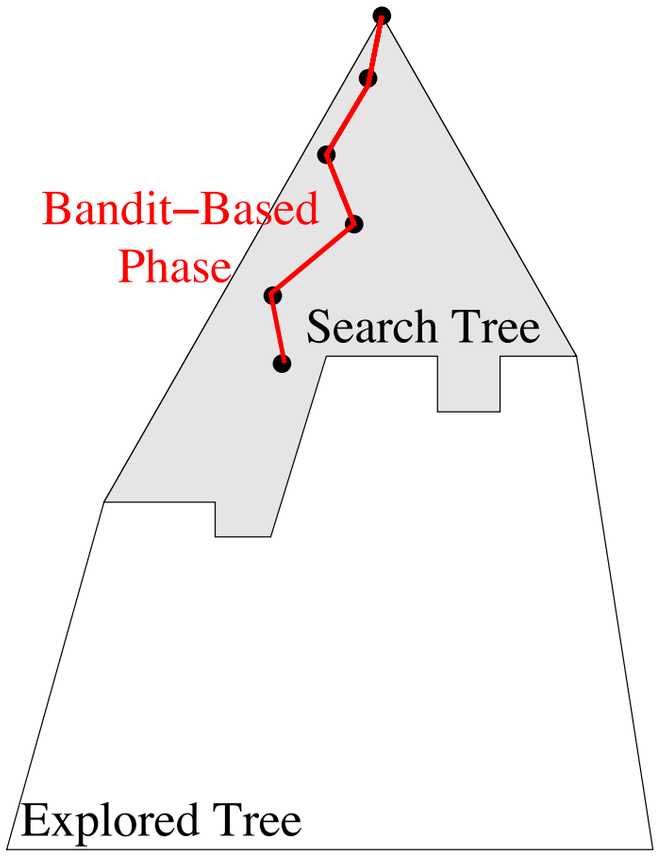
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



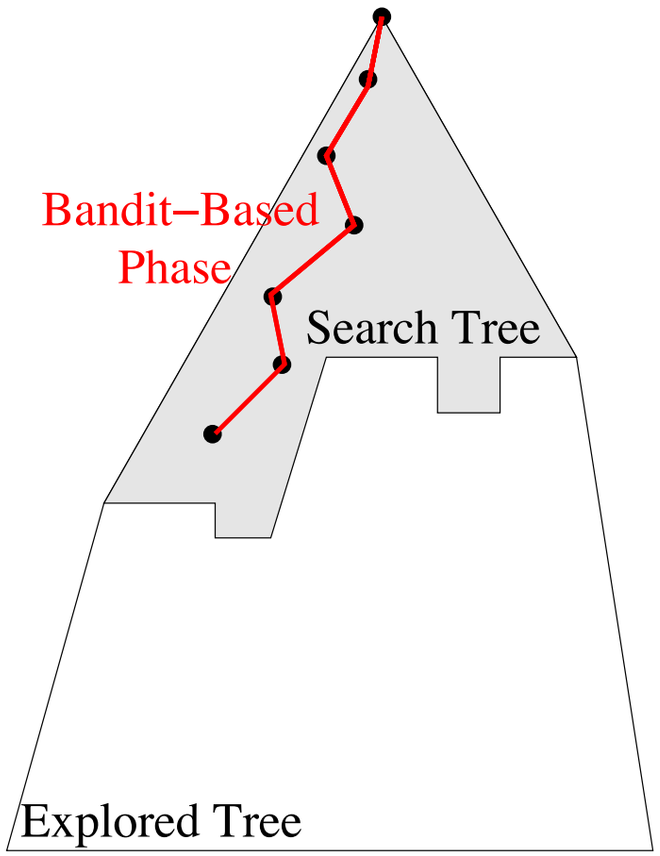
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



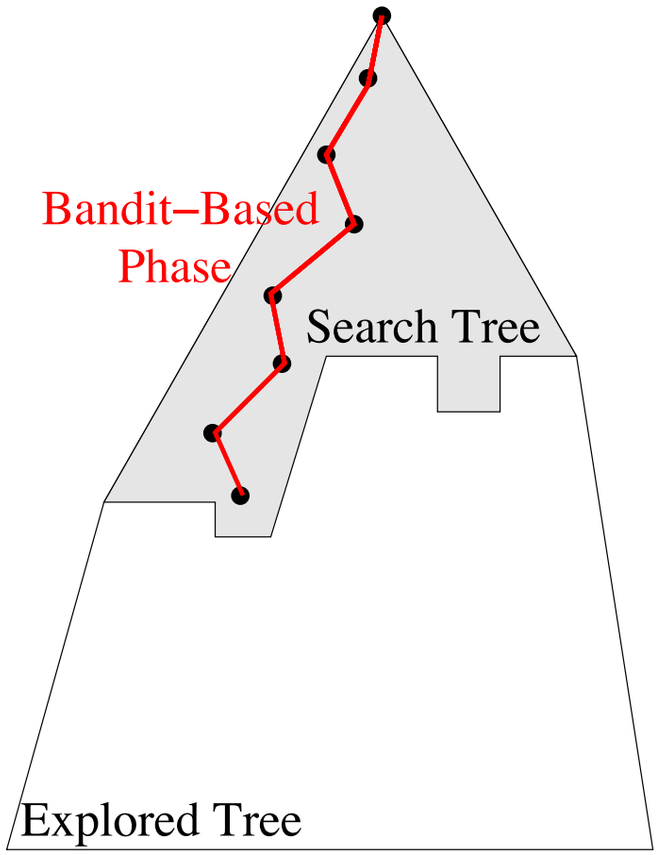
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



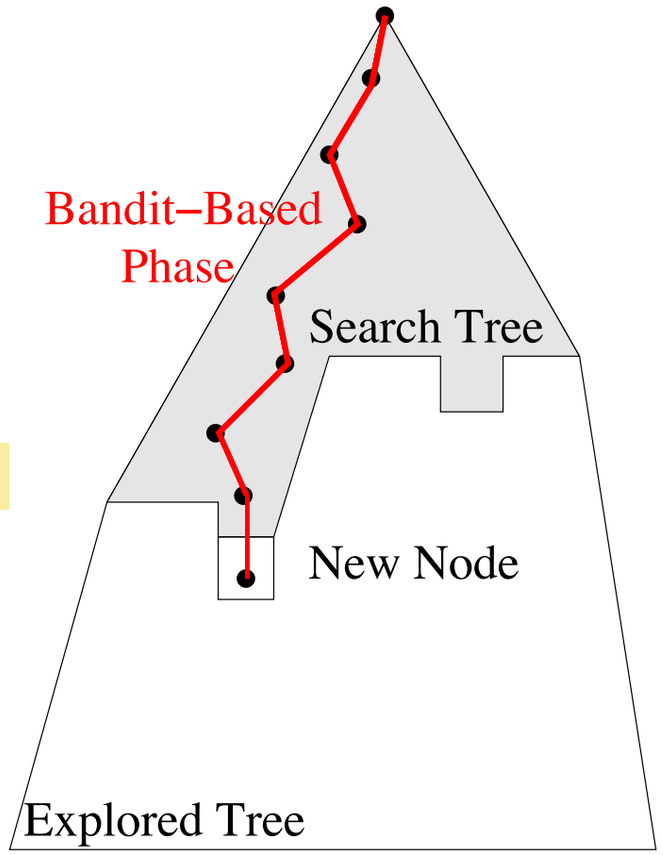
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



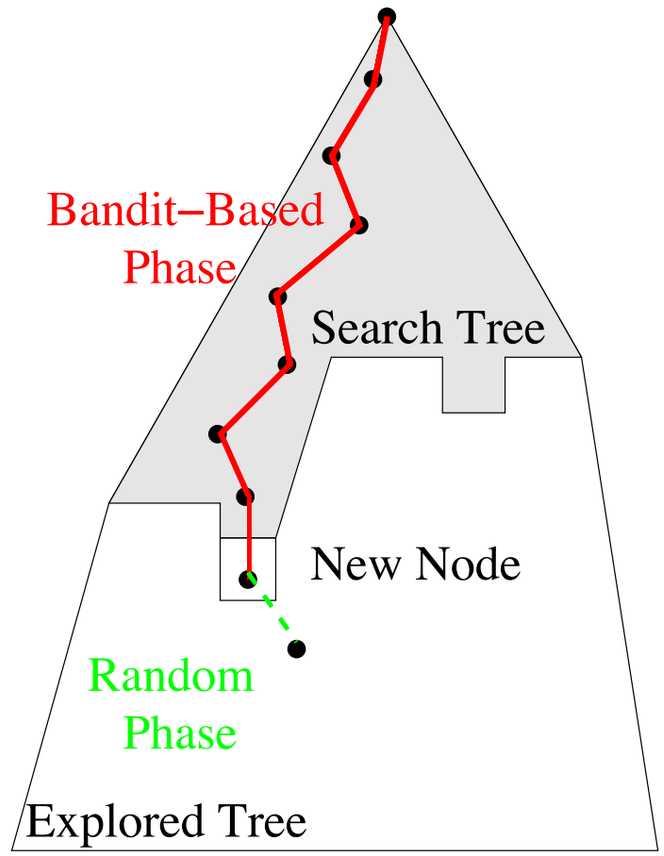
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



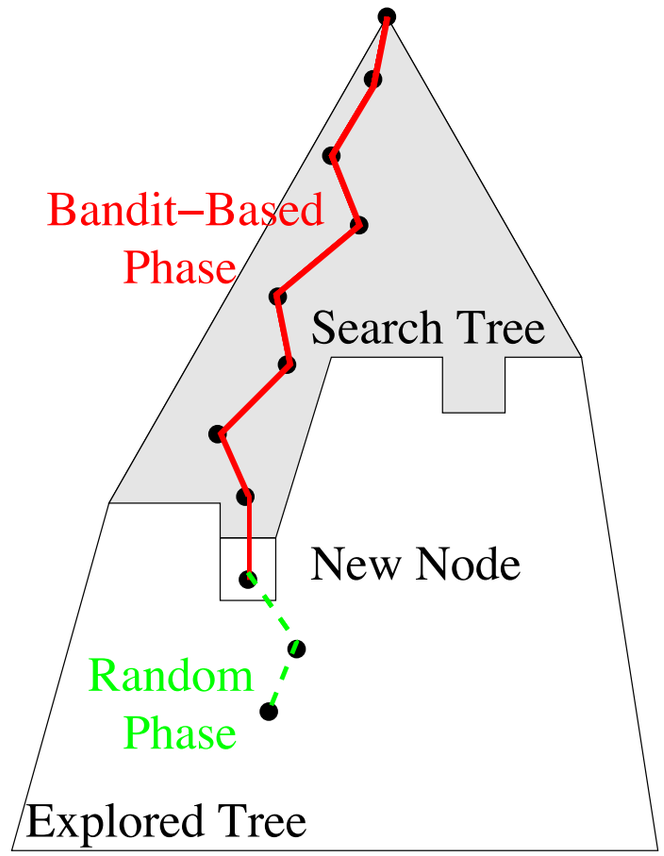
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)

function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```



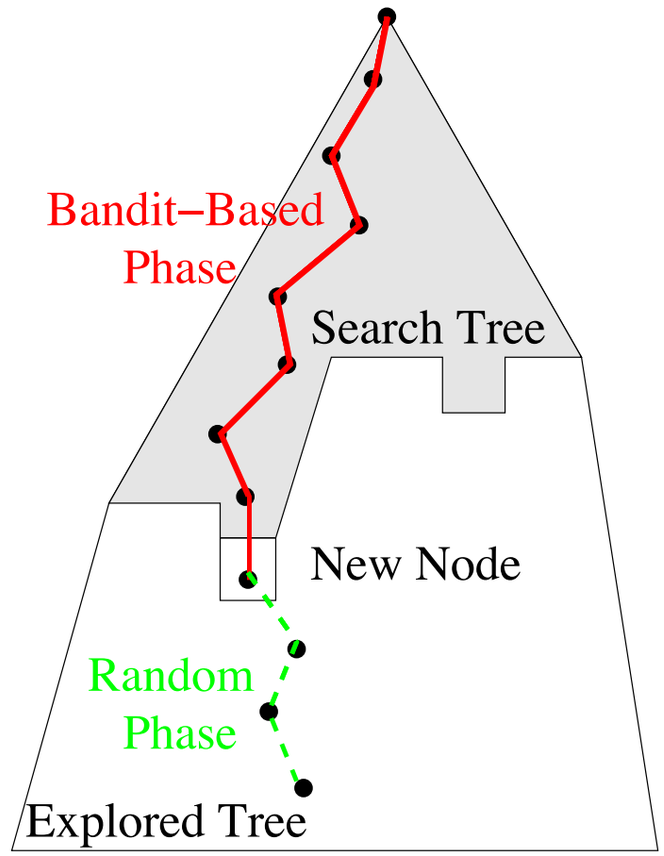
Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)

function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```



Basic MCTS pseudocode

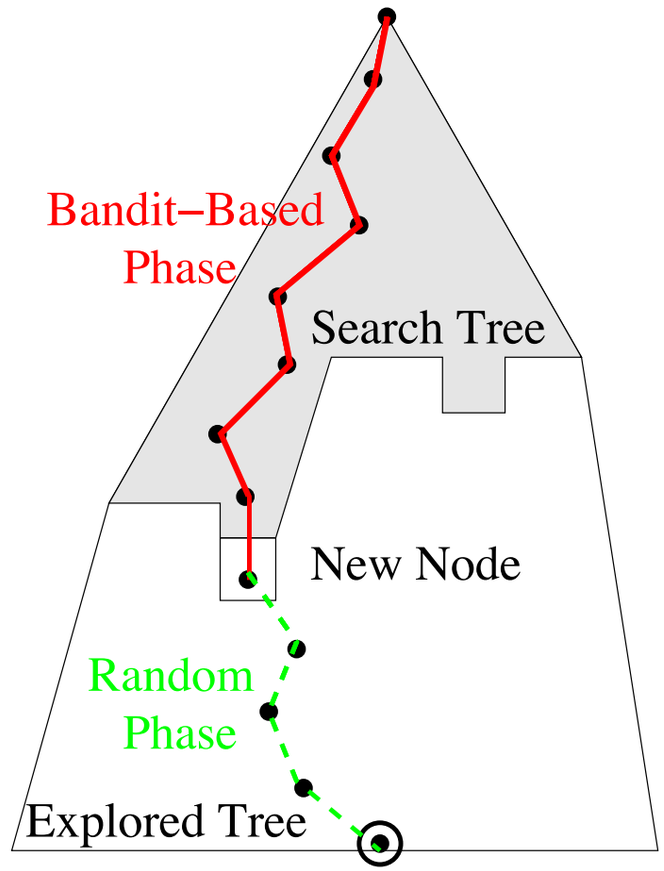
```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```




```
function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```

Basic MCTS pseudocode

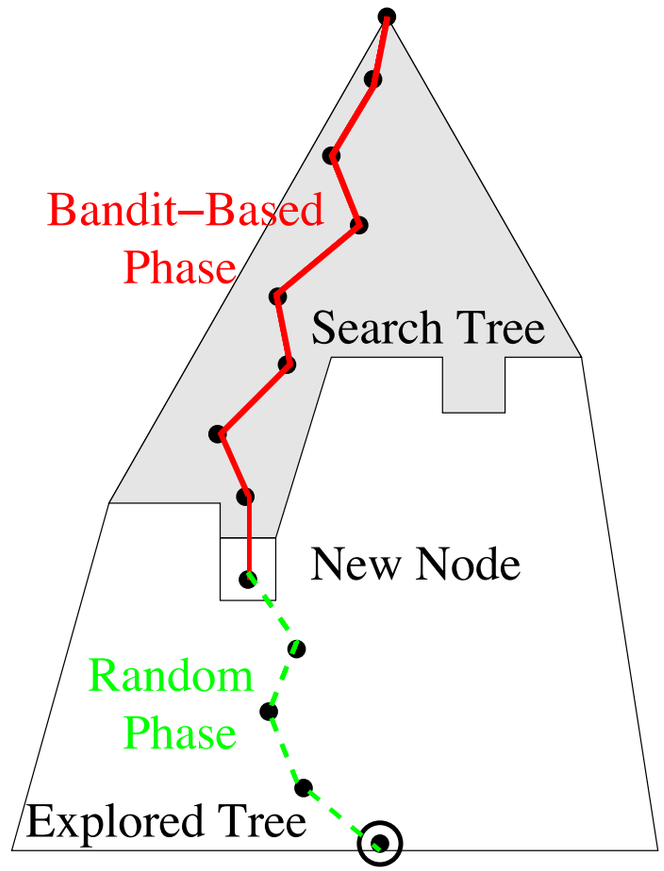
```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
    update_value(state, winner)
```




```
function random_playout(state):
    if is_terminal(state):
        return winner
    else: return random_playout(random_move(state))
```

Basic MCTS pseudocode

```
function MCTS_sample(state)
    state.visits++
    if all children of state expanded:
        next_state = UCB_sample(state)
        winner = MCTS_sample(next_state)
    else:
        if some children of state expanded:
            next_state = expand(random unexpanded child)
        else:
            next_state = state
        winner = random_playout(next_state)
update_value(state, winner)
```



Monte-Carlo Tree Search