

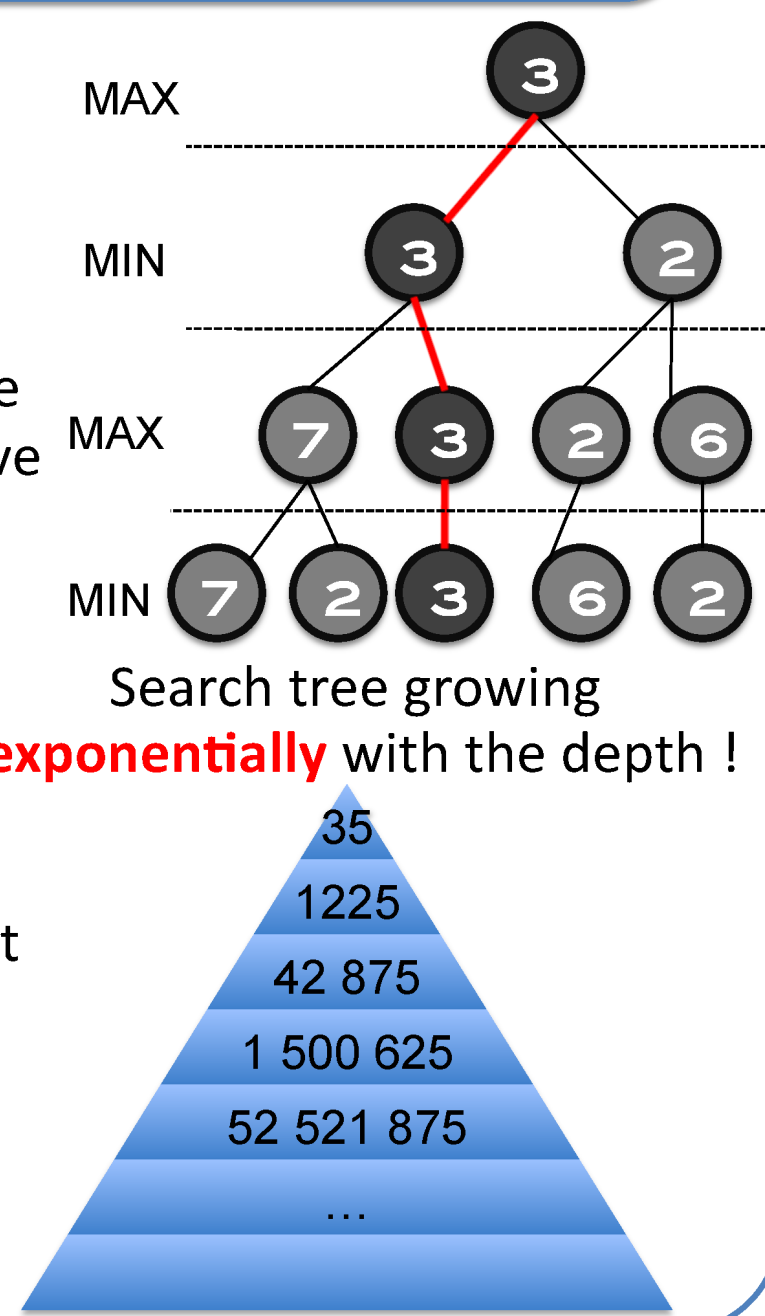


MONTE-CARLO TREE SEARCH

Olivier Teytaud, Hassen Doghmen
Tao, INRIA Saclay-IDF

Minimax Algorithm: the brute force solution

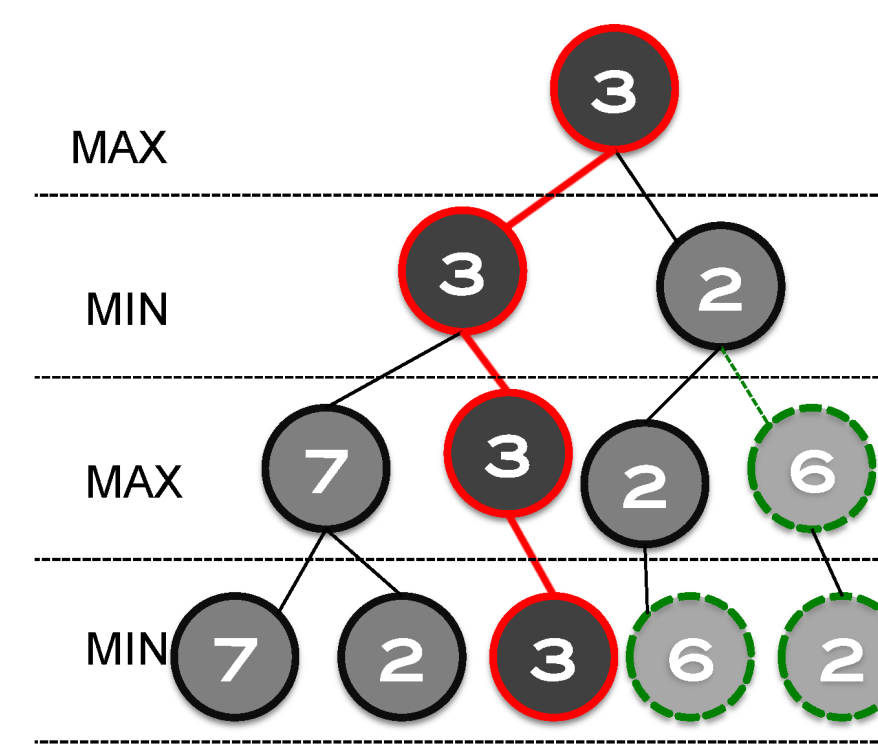
- For the MAX player
1. Generate the game to terminal states
 2. Apply the utility function to the terminal states
 3. Back-up values
 - At MIN ply assign minimum payoff move
 - At MAX ply assign maximum payoff move
 4. At root, MAX chooses the operator that led to the highest payoff



- ✓ Perfect play for deterministic, perfect-information games
- ✗ Assume that the opponent will select its best move choice and make no mistakes
- ✗ Totally impractical
 - Time complexity is $O(b^d)$
 - Space complexity is $O(b^d)$

AlphaBeta: an improvement of Minimax

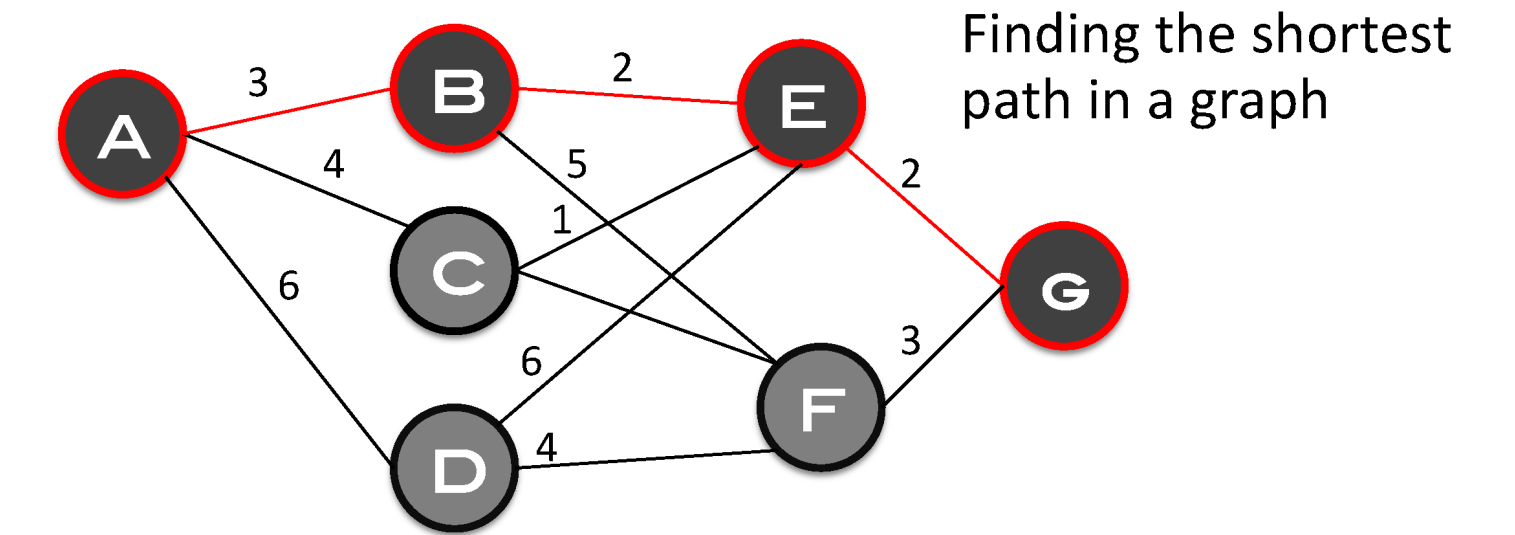
- Algorithm : a MiniMax with **pruning**.
- ALPHABETA takes advantage of the calculated states when determining whether to continue searching a particular subtree or prune it.
- Pruning does not affect final result.
- Good move ordering improves effectiveness of pruning.
- Time complexity
 - $O(b^{d/2})$
 - means we go from an effective branching factor of b to \sqrt{b} (e.g. 35 -> 6 for chess).



ALPHABETA twice faster

Dynamic programming

- Dynamic Programming is a method used to **optimize** the objective function in a planning problem.
- Recursively **decompose** the problem into subproblems, solve subproblems and construct the global solution.
- However, the algorithm is **not anytime**; if stopped before then, there is no result.
- Can be seen as one player game against a **random** player.



Monte-Carlo Tree Search: Algorithm sketch

Input: a situation s
Output: a (hopefully good) move m for situation s
Memory = empty; statistics = empty mapping

```

While (time left > 0)
  s' = s, game = ()
  While (s' in memory, not terminal)
    s' = reachable situation such that nbWins(s') / nbSims(s') max
  endwhile
  s'' = s'
  While (s' not terminal)
    s' = random reachable situation from s'
  endwhile
  result = result(s')
  memory = memory + s''
  for all states in the simulation
    statistics(s) = statistics(s) + statistics(game, result)
  endfor
endwhile
    
```

Restart games

Choose move to be explored
Using statistics

Choose move to be explored

Update statistics

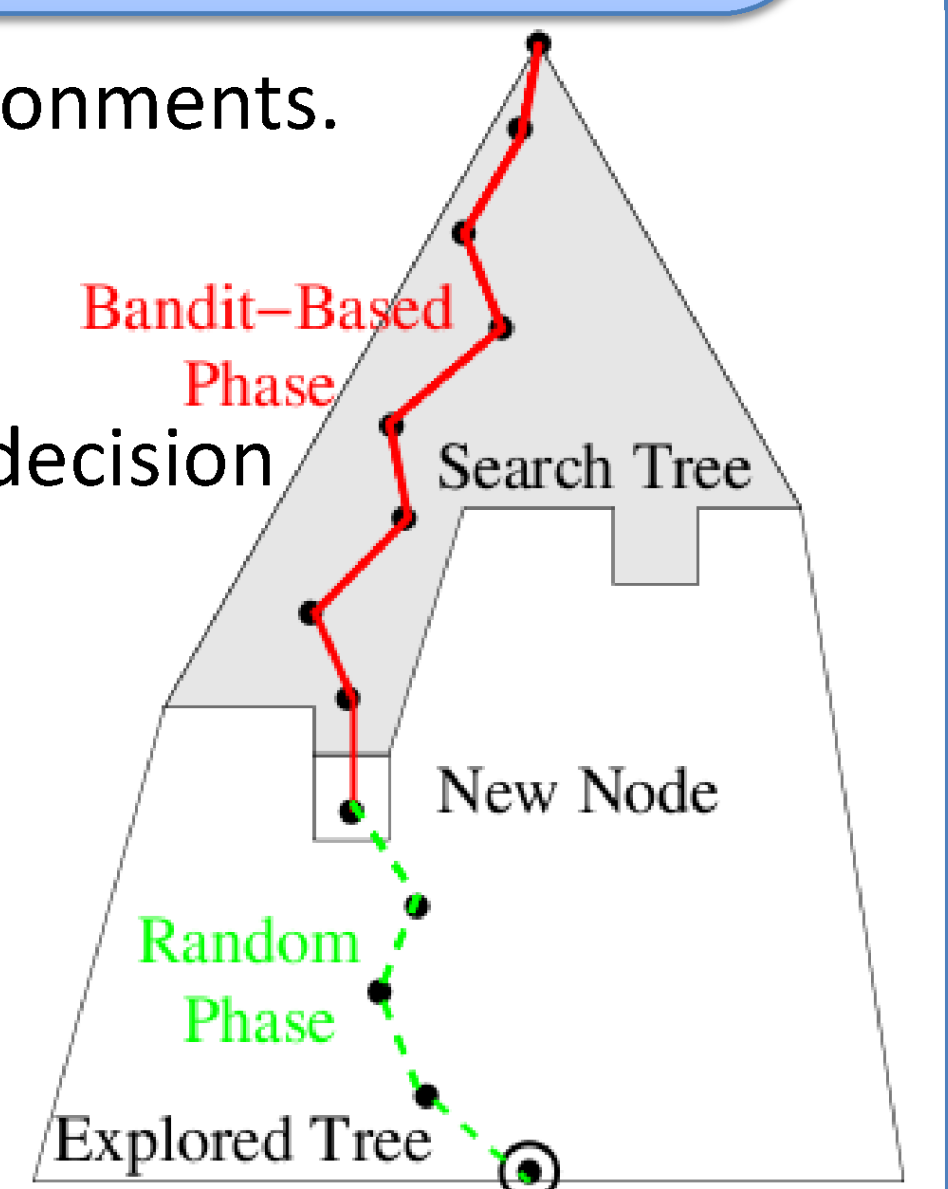
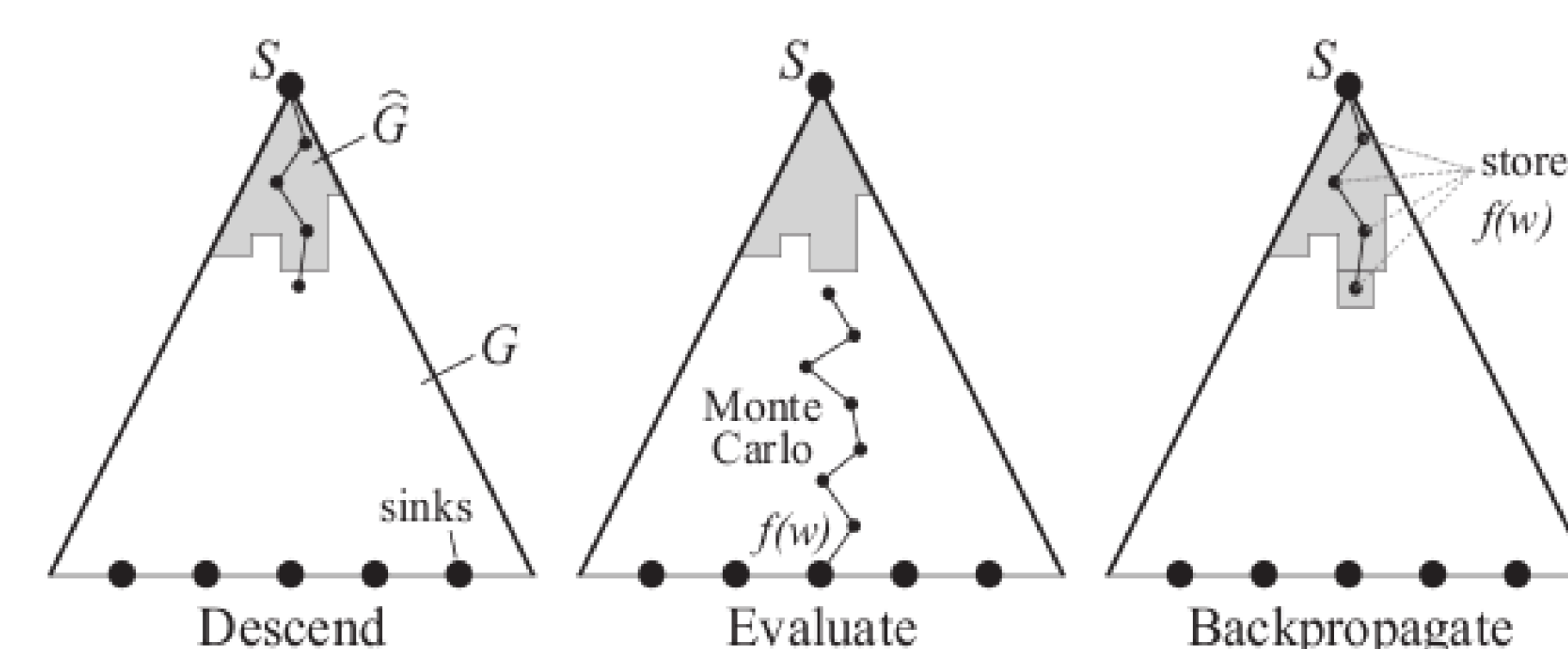
MCTS: how it works ?

- Algorithms for taking decisions in uncertain environments.

Principles:

- Iteratively growing **unbalanced** tree (choosing a decision is seen as bandit problems – UCT).
- Random simulations (Monte-Carlo simulations as an **evaluation** function).

3 steps loop while time budget > 0



K-armed bandit problem

- Problem Description :
 - p_1, \dots, p_N unknown probabilities $\in [0, 1]$
 - At each time step $t \in \{1, \dots, N\}$
 - Choose arm $a_t \in \{1, \dots, N\}$ (as a function of a_j and r_t)
 - With probability $p_{j,t}$
 - win ($r_t = 1$)
 - loose ($r_t = 0$)

- Goal: minimize a Regret: $R_N = N \max\{p_j\} - \sum r_t$ ($t < N$)

- ➔ Classical solution: UCB: Choose arm j maximizing the compromise:

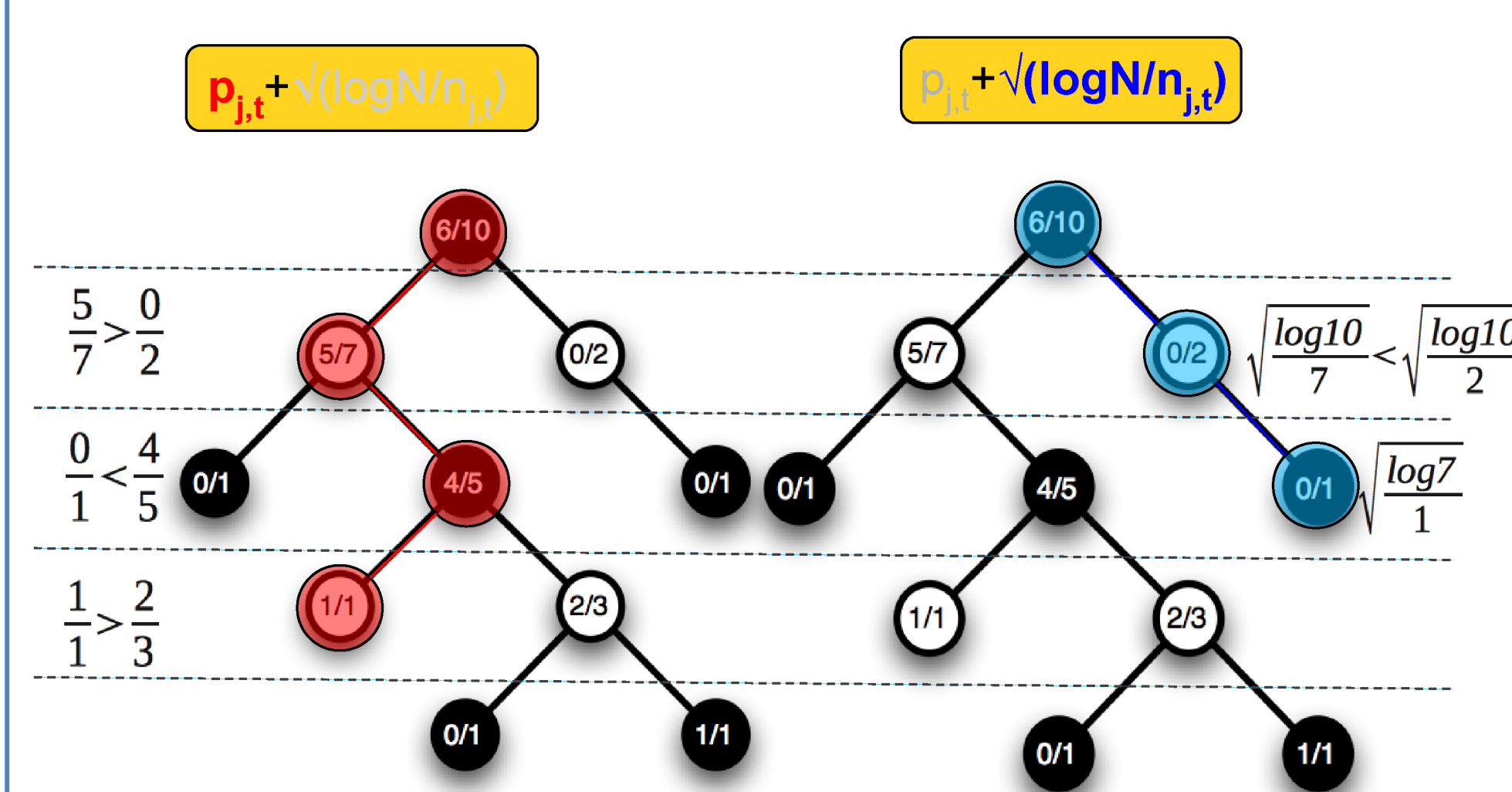
$$p_{j,t} + \sqrt{\frac{\log N}{n_{j,t}}}$$

(Lai et al; Auer et al)

$p_{j,t}$: Empirical average for decision j
 N : Total number of trials
 $n_{j,t}$: number of trials with decision j

- ➔ Optimal regret $O(\log(n))$

Exploitation vs Exploration dilemma



K-armed bandit problem

- simulations to approximate π
- Simulations to find the optimal move

```

double nbTotalPoints = 0;
double nbPointsInSquare = 0;
double PI;
int timeLeft = 10000; // time budget
double x, y;

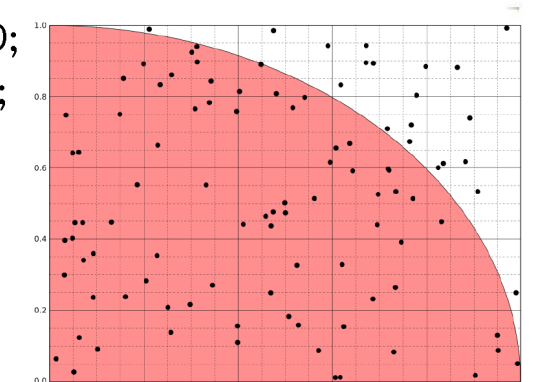
srand(time(NULL));
cout << "Estimating PI by MC Simulations" << endl;

while (timeLeft > 0) {
  x = (double)(rand() % 10) / 10;
  y = (double)(rand() % 10) / 10;

  nbTotalPoints++;

  if ((sqrt(x * x + y * y)) < 1) {
    nbPointsInSquare++;
  }
  timeLeft--;
}

PI = 4 * (nbPointsInSquare / nbTotalPoints);
    
```



Monte-Carlo evaluation function consists in starting from the position and playing with a **simulation policy** until the end of the game.

MoGo: record of achievement

- 2011 : First win against a pro (6D) H2 in 13x13 MoGoTW
- 2011 : First win against a pro (9P) H2.5 in 13x13 MoGoTW
- 2011 : First win against a pro in Blind-Go in 9x9 MoGoTW
- 2010 : Gold medalist in TAAI 2010 in three categories: 9x9 Go, 13x13 Go, 19x19 Go. MoGoTW
- 2007: win against a pro (5p) in 9x9 (blitz) MoGo
- 2008: win against a pro (5p) in 9x9 white MoGo
- 2009: win against a pro (5p) in 9x9 black MoGo
- 2009: win against a pro (9p) in 9x9 black MoGoTW
- 2008: win against a pro (8p) 19x19, H9 MoGo
- 2009: win against a pro (9p) 19x19, H7 MoGo
- 2009: win against a pro (1p) 19x19, H6 MoGo

MCTS = an efficient approach for taking decisions under uncertainty

Application to energy

- MCTS in the continuous domain
- Energy plant management
- Financial market
- Application to partially observable problems

