Monte Carlo Search

Tristan Cazenave

LAMSADE CNRS

Université Paris-Dauphine

PSL

Tristan.Cazenave@dauphine.fr

Outline

- Monte Carlo Tree Search
- Nested Monte Carlo Search
- Nested Rollout Policy Adaptation
- Playout Policy Adaptation
- Imperfect Information Games
- Zero Learning (Deep RL)


INTELLIGENCE ARTIFICIELLE
Une approche ludique
Tristan Cazenave
ellipses

Monte Carlo Tree Search

Monte Carlo Go

- 1993 : first Monte Carlo Go program
  - Gobble, Bernd Bruegmann.
  - How nature would play Go ?
  - Simulated annealing on two lists of moves.
  - Statistics on moves.
  - Only one rule : do not fill eyes.
  - Result = average program for 9x9 Go.
  - Advantage : much more simple than alternative approaches.

Monte Carlo Go

- 1998 : first master course on Monte Carlo Go.
- 2000 : sampling based algorithm instead of simulated annealing.
- 2001 : Computer Go an AI Oriented Survey.
- 2002 : Bernard Helmstetter.
- 2003 : Bernard Helmstetter, Bruno Bouzy, Developments on Monte Carlo Go.

Monte Carlo Phantom Go

- Phantom Go is Go when you cannot see the opponent's moves.
- A referee tells you illegal moves.
- 2005 : Monte Carlo Phantom Go program.
- Many Gold medals at computer Olympiad since then using flat Monte Carlo.
- 2011 : Exhibition against human players at European Go Congress.

UCT

- UCT : Exploration/Exploitation dilemma for trees [Kocsis and Szepesvari 2006].
- Play random random games (playouts).
- Exploitation : choose the move that maximizes the mean of the playouts starting with the move.
- Exploration : add a regret term (UCB).

UCT

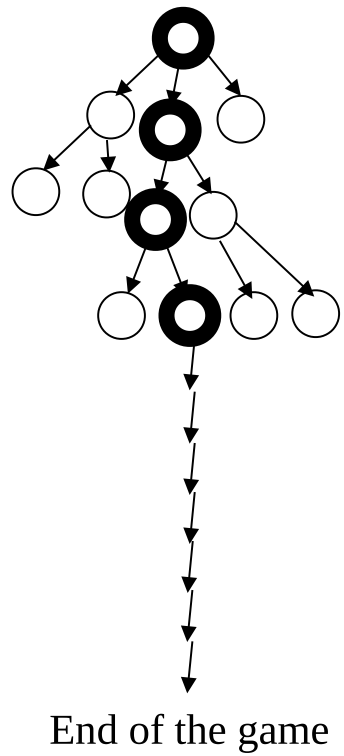
- UCT : exploration/exploitation dilemma.
- Play the move that maximizes

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

In which

- w_i = number of wins after the i-th move
- n_i = number of simulations after the i-th move
- c = exploration parameter (theoretically equal to √2)
- t = total number of simulations for the parent node

UCT

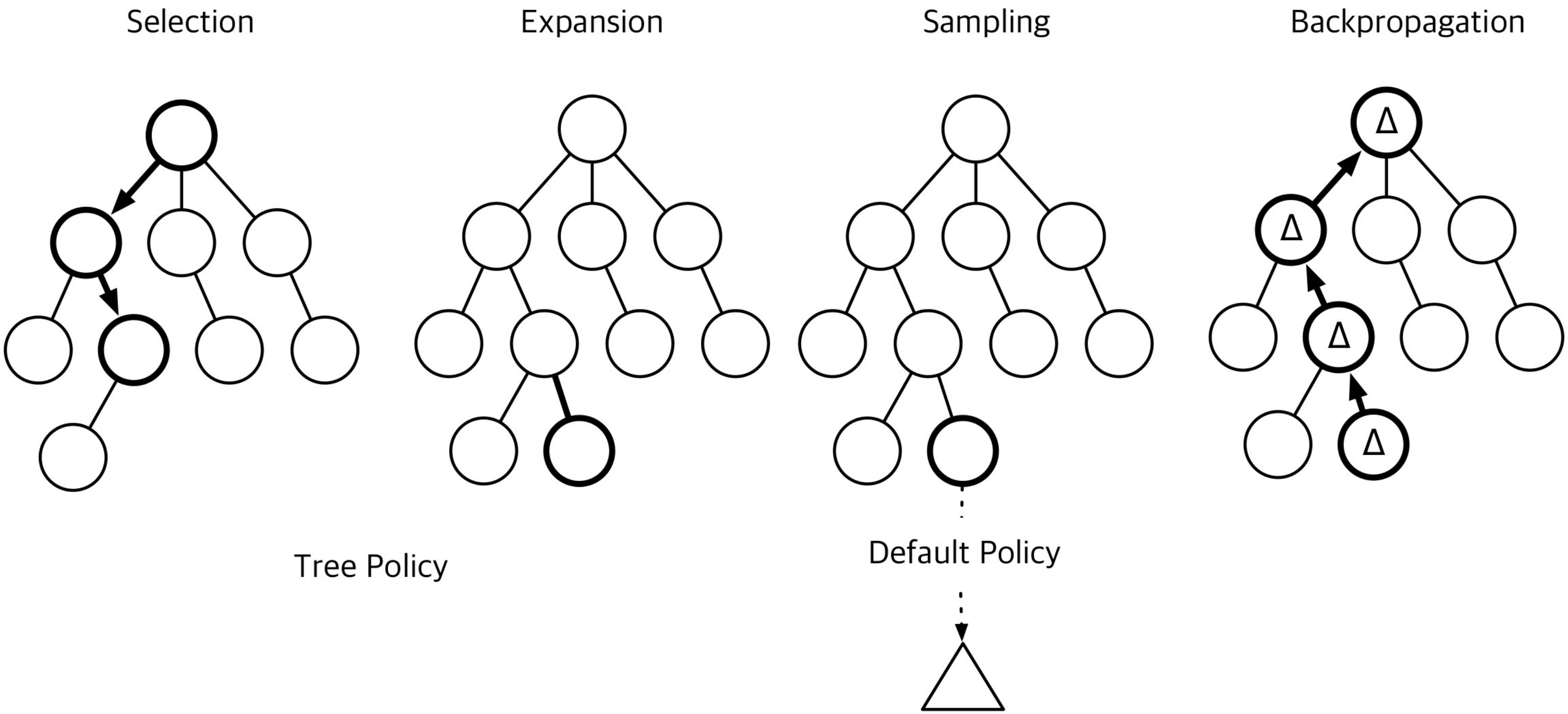


1) descent of the tree

2) playout

3) update the tree

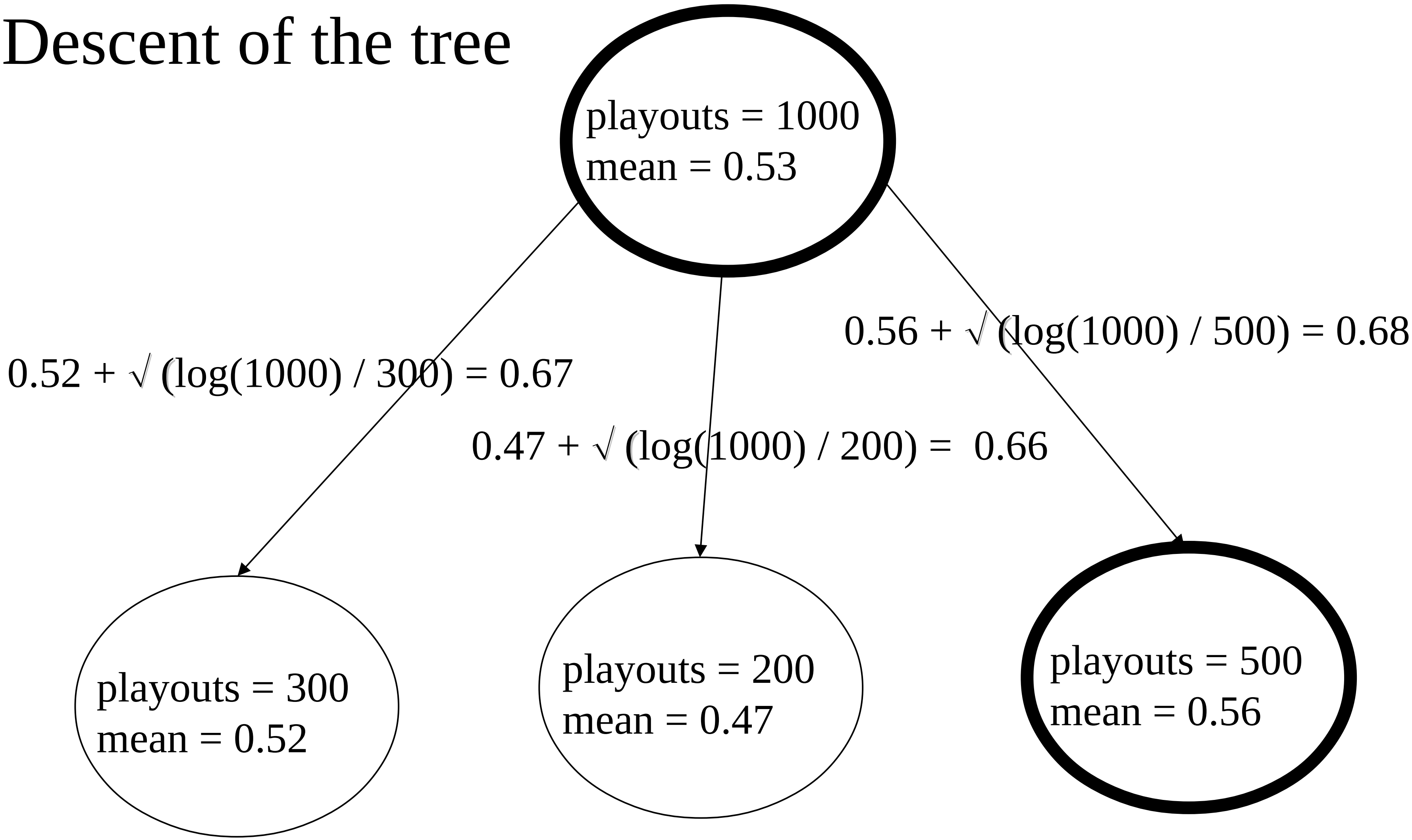
UCT



Selection
Expansion
Sampling
Backpropagation
Δ
Δ
Δ
Δ
Tree Policy
Default Policy

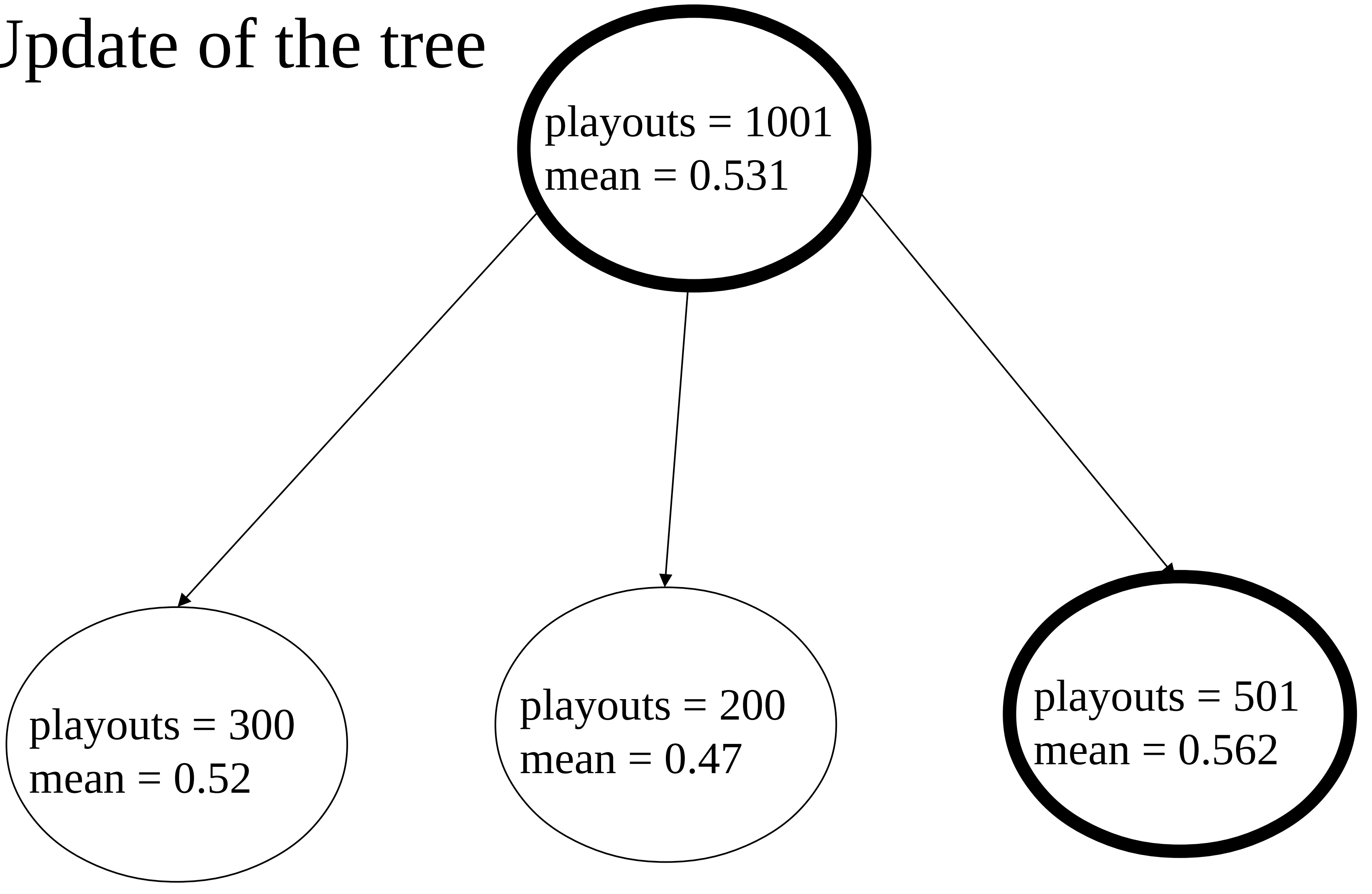
UCT

Descent of the tree



UCT

Update of the tree



UCT

function $\text{MCTSSEARCH}(s_0)$
    create root node v_0 with state s_0
    while within computational budget **do**
        $v_l \leftarrow \text{TREEPOLICY}(v_0)$
        $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$
        $\text{BACKUP}(v_l, \Delta)$
    return $a(\text{BESTCHILD}(v_0, 0))$

UCT

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

UCT

function $\text{Expand}(v)$
  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'

UCT

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

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

UCT

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

UCT

function $\text{Backup}(v, \Delta)$
  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

AMAF

- All Moves As First (AMAF).
- AMAF calculates for each possible move of a state the average of the playouts that contain this move.

RAVE

- A big improvement for Go, Hex and other games is Rapid Action Value Estimation (RAVE) [Gelly and Silver 2007].

- RAVE combines the mean of the playouts that start with the move and the mean of the playouts that contain the move.

RAVE

- Parameter β_m for move m is :

  $\beta_m \leftarrow pAMAF_m /$

  $(pAMAF_m + p_m + bias \times pAMAF_m \times p_m)$

- β_m starts at 1 when no playouts and decreases as more playouts are played.
- Selection of moves in the tree :

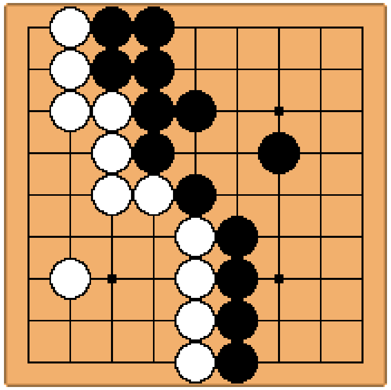
  $argmax_m((1.0 - \beta_m) \times mean_m + \beta_m \times AMAF_m)$

GRAVE

- Generalized Rapid Action Value Estimation (GRAVE) is a simple modification of RAVE.
- It consists in using the first ancestor node with more than n playouts to compute the RAVE values.
- It is a big improvement over RAVE for Go, Atarigo, Knightthrough and Domineering [Cazenave 2015].

Atarigo

- Atarigo is a simplification of the game of Go.
- The winner of a game is the first player to capture a string of stones.



White to play and win.

Atarigo

- Atarigo 8x8 with 10,000 playouts :

  RAVE wins 94.2 % against UCT.

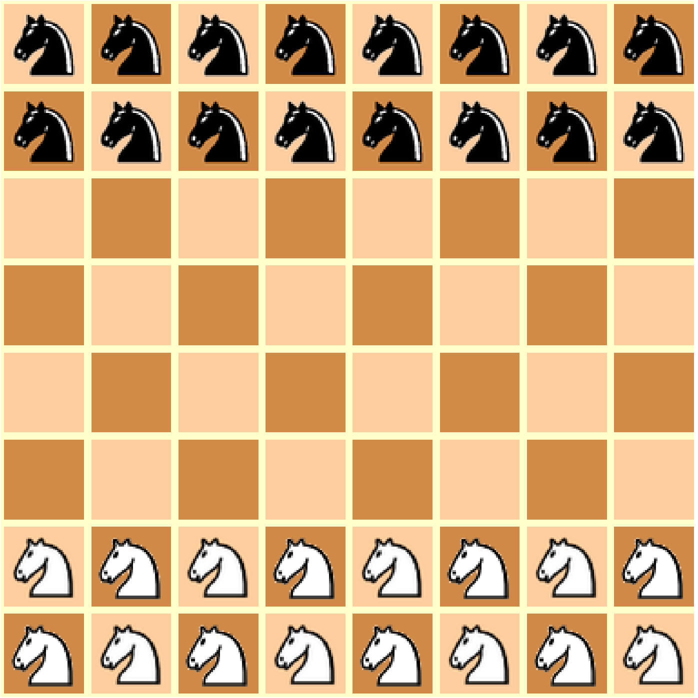
  GRAVE wins 88.4 % against RAVE.

- Atarigo 19x19 with 1,000 playouts :

  RAVE wins 72.4 % against UCT.

  GRAVE wins 78.2 % against RAVE.

Knightthrough



Knightthrough

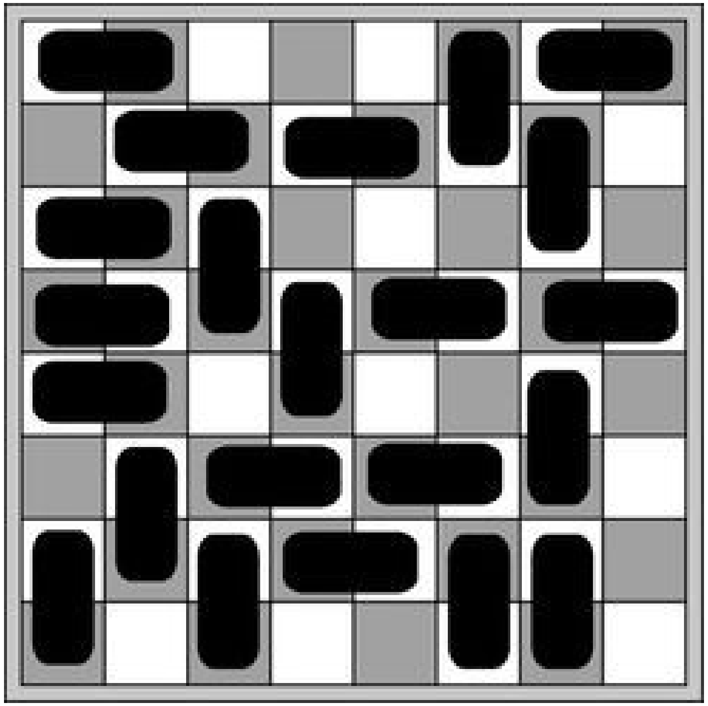
- RAVE wins 69.4 % against UCT with 1,000 playouts.
- GRAVE wins 67.8 % against RAVE with 1,000 playouts.
- RAVE wins 56.2 % against UCT with 10,000 playouts.
- GRAVE wins 67.2 % against RAVE with 10,000 playouts.

Domineering

- Domineering is a two player combinatorial game usually played on an 8x8 board.
- It consists in playing 2x1 dominoes on the board.
- The first player put the dominoes vertically and the second player put them horizontally.
- If a player cannot play anymore, he loses the game.

Domineering

- The last to play has won.



Domineering

- Domineering 8x8 with 10,000 playouts :

  RAVE wins 72.6 % against UCT.

  GRAVE wins 62.4 % against RAVE.

- Domineering 19x19 with 1,000 playouts :

  RAVE wins 63.8 % against UCT.

  GRAVE wins 56.4 % against RAVE.

Go

- Go is an ancient oriental game of strategy that originated in China thousands of years ago.
- It is usually played on a 19x19 grid.
- AlphaGo is the current best (computer) Go player.
- MCTS is the best algorithm for the game of Go.
- The RAVE algorithm was originally designed for computer Go.

Go 9x9

- Go 9x9 with 1,000 playouts :

  RAVE wins 89.6 % against UCT.

  GRAVE wins 66.0 % against RAVE.

- Go 9x9 with 10,000 playouts :

  RAVE wins 73.2 % against UCT.

  GRAVE wins 54.4 % against RAVE.

Go 19x19

- Go 19x19 with 1,000 playouts :

  GRAVE wins 81.8 % against RAVE.

- Go 19x19 with 10,000 playouts :

  GRAVE wins 62.4 % against RAVE.

Three Color Go

- Multicolor Go is Go with more than two players.
- Three Color Go is played with stones of three different colors.
- Chinese rules are used to score games.
- The winner is the player that has the greatest score at the end.
- The average winning rate is 33.33 %.

Three Color Go

- Three Color Go 9x9 with 1,000 playouts :
  RAVE wins 70.83 % against UCT.
  GRAVE wins 57.17 % against RAVE.

- Three Color Go 19x19 with 1,000 playouts :
  RAVE wins 18.50 % against two GRAVE.

Parallelization of MCTS

- Root Parallelization.
- Tree Parallelization (virtual loss).
- Leaf Parallelization.

MCTS

- Great success for the game of Go since 2007.
- Much better than all previous approaches to computer Go.

AlphaGo

Lee Sedol is among the strongest and most famous 9p Go player :

AlphaGo has won 4-1 against Lee Sedol in March 2016

AlphaGo Master wins 3-0 against Ke Jie, 60-0 against pros.

AlphaGo Zero wins 89-11 against AlphaGo Master in 2017.

General Game Playing

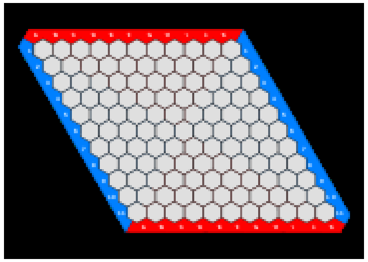
- General Game Playing = play a new game just given the rules.
- Competition organized every year by Stanford.
- Ary world champion in 2009 and 2010.
- All world champions since 2007 use MCTS.

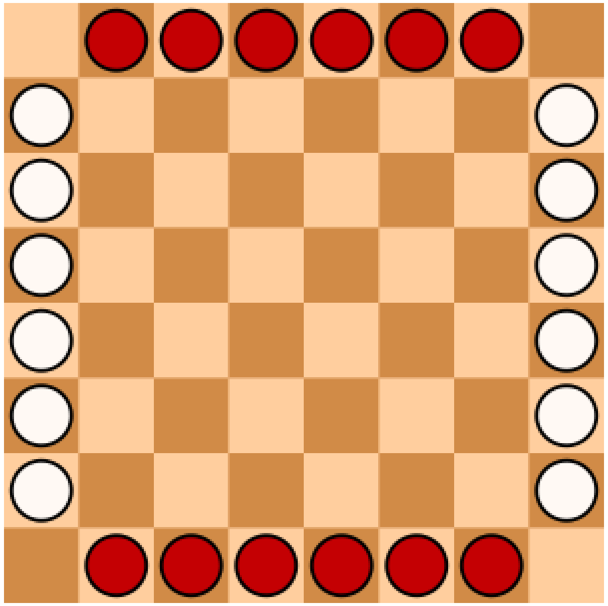
General Game Playing

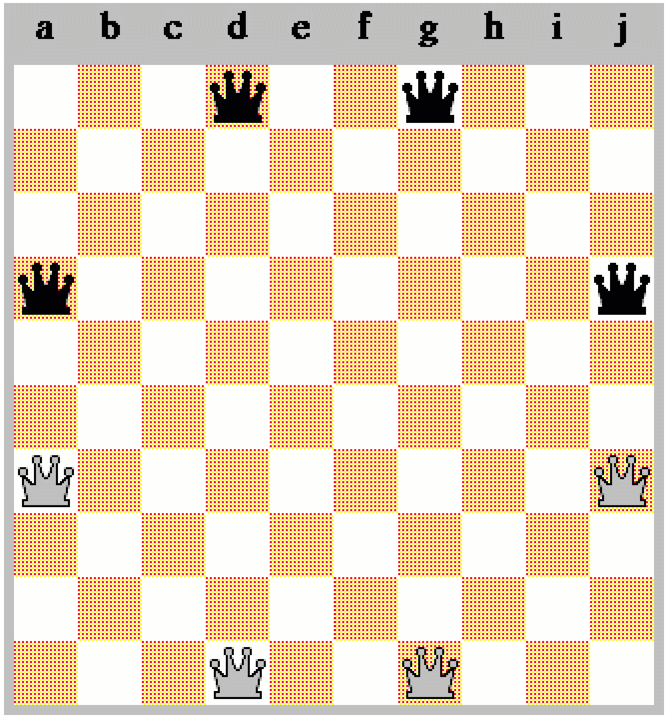
- Eric Piette combined Stochastic Constraint Programming with Monte Carlo in WoodStock.
- World champion in 2016 (MAC-UCB-SYM).
- Detection of symmetries in the states.

Other two-player games

- Hex : 2009
- Amazons : 2009
- Lines of Action : 2009



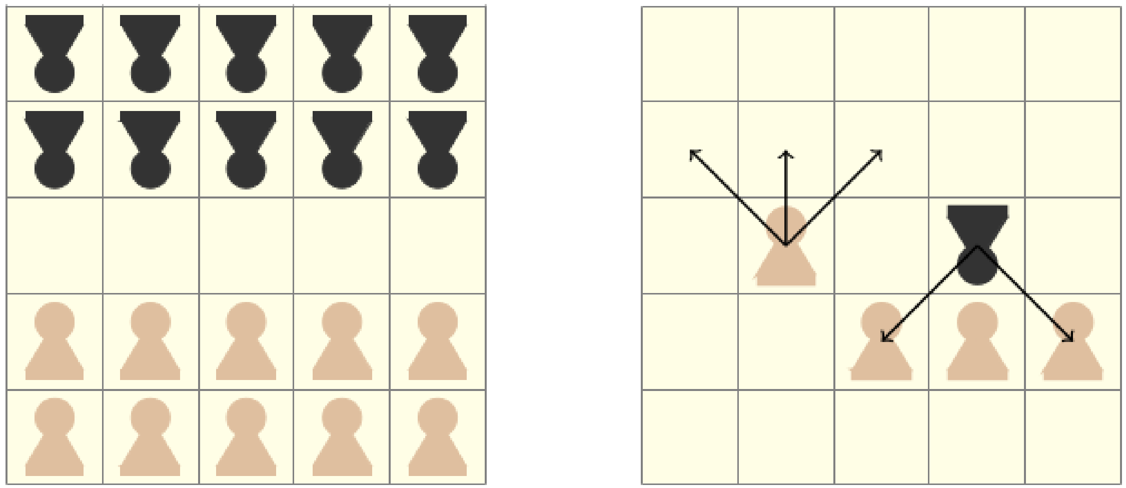




MCTS Solver

- When a subtree has been completely explored the exact result is known.
- MCTS can solve games.
- Score Bounded MCTS is the extension of pruning to solving games with multiple outcomes.

Breakthrough



(a) Starting position on size 5×5. (b) Possible movements.

- Write the Board and Move classes for Breakthrough 5x5.
- Write the function for the possible moves.
- Write a program to play random games at Breakthrough 5x5.

Breakthrough

- The Move class contains the color, the starting and arriving locations of a pawn.

```
class Move(object):
    def __init__(self, color, x1, y1, x2, y2):
        self.color = color
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
```

Breakthrough

- The Bord class initializes the board with two rows of Black and two rows of White pawns:

```
Dx = 5
Dy = 5
Empty = 0
White = 1
Black = 2
class Board(object):
    def __init__(self):
        self.h = 0
        self.turn = White
        self.board = np.zeros ((Dx, Dy))
        for i in range (0, 2):
            for j in range (0, Dy):
                self.board [i] [j] = White
        for i in range (Dx - 2, Dx):
            for j in range (0, Dy):
                self.board [i] [j] = Black
```

Breakthrough

- Test if a move is valid for a given board:

```
def valid (self, board):
    if self.x2 >= Dx or self.y2 >= Dy or self.x2 < 0 or self.y2 < 0:
        return False
    if self.color == White:
        if self.x2 != self.x1 + 1:
            return False
        if board.board [self.x2] [self.y2] == Black:
            if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1:
                return True
            return False
        elif board.board [self.x2] [self.y2] == Empty:
            if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1 or self.y2 == self.y1:
                return True
            return False
    elif self.color == Black:
        ...
```

Breakthrough

- Generate the legal moves:

```
def legalMoves(self, color):
    moves = []
    for i in range (0, Dx):
        for j in range (0, Dy):
            if self.board [i] [j] == color:
                for k in [-1, 0, 1]:
                    for l in [-1, 0, 1]:
                        m = Move (color, i, j, i + k, j + l)
                        if m.valid (self):
                            moves.append (m)
    return moves
```

Flat Monte Carlo

- Write a won function to detect when a game is won.
- Write a playout function that plays a random game from the current state and returns the result of the random game (1.0 if White wins, 0.0 else).

Flat Monte Carlo

- Keep statistics for all the moves of the initial state.
- For each move of the initial state, keep the number of playouts starting with the move and the number of playouts starting with the move that have been won.
- Play the move with the greatest mean when all the playouts are finished.

UCB

Choose the first move at the root according to UCB before each playout:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

In which

- w_i = number of wins after the i-th move
- n_i = number of simulations after the i-th move
- c = exploration parameter (theoretically equal to √2)
- t = total number of simulations for the parent node

UCB vs Flat

- Make UCB with 1 000 playouts play 100 games against Flat with 1 000 playouts.
- Tune the UCB constant (hint 0.4).