



MONTE-CARLO TREE SEARCH

Olivier Teytaud, Hassen Doghmen
Tao, INRIA Saclay-IDF

ABSTRACT HERE.....

Multi-Armed Bandit Problem

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
- ✗ The program must assume that the opponent will select its best move choice and make no mistakes
- ✗ Totally impractical since it generates the whole tree
 - Time complexity is $O(b^d)$
 - Space complexity is $O(b^d)$

Search tree growing exponentially with the depth

Minimax algo

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
- ✗ The program must assume that the opponent will select its best move choice and make no mistakes

- ✗ Totally impractical since it generates the whole tree
 - Time complexity is $O(b^d)$
 - Space complexity is $O(b^d)$

Search tree growing **exponential** with the depth !

Minimax algo

EXP3 Algorithm

Parameter : real $\gamma \in]0; 1]$

Initialization : define the weight $w_i(t) = 1$ for $t = 1$ and all $i = 1, \dots, k$

For each $t = 1, 2, \dots$

1. Set

$$p_i(t) = (1 - \gamma) \frac{w_i(t)}{\sum_{j=1}^k w_j(t)} + \frac{\gamma}{K}$$
 for $i = 1, \dots, K$.
2. Select randomly an arm i_t according to the probabilities $p_1(t), \dots, p_K(t)$
3. Observe the reward $r_{i_t}(t)$
4. Update the weight of i_t by

$$w_{i_t}(t+1) = w_{i_t}(t) \exp\left(\frac{\gamma r_{i_t}(t)}{K p_{i_t}(t)}\right)$$
 and set $w_j(t+1) = w_j(t)$ for other arms.

• In order to find good actions EXP3 maintains a balance between

- **exploration:** weighting actions according to the reward: $1 - \gamma$ term in the probability, and
- **exploration:** the uniform term $\frac{\gamma}{K}$ ensures that unsuffi-

EXP3 Algorithm

Parameter : real $\gamma \in]0; 1]$

Initialization : define the weight $w_i(t) = 1$ for $t = 1$ and all $i = 1, \dots, K$

For each $t = 1, 2, \dots$

1. Set

$$p_i(t) = (1 - \gamma) \frac{w_i(t)}{\sum_{j=1}^K w_j(t)} + \frac{\gamma}{K}$$
 for $i = 1, \dots, K$.
2. Select randomly an arm i_t according the the probabilities $p_1(t), \dots, p_K(t)$
3. Observe the reward $r_{i_t}(t)$
4. Update the weight of i_t by

$$w_{i_t}(t+1) = w_{i_t}(t) \exp\left(\frac{\gamma}{K} \frac{r_{i_t}(t)}{p_{i_t}(t)}\right)$$
 and set $w_j(t+1) = w_j(t)$ for other arms.

- In order to find good actions EXP3 maintains a balance between
 - exploration:** weighting actions according to the reward: $1 - \gamma$ term in the probability, and
 - exploration:** the uniform term $\frac{\gamma}{K}$ ensures that unsuffi-

EXP3 Algorithm

Parameter : real $\gamma \in]0; 1]$

Initialization : define the weight $w_i(t) = 1$ for $t = 1$ and all $i = 1, \dots, K$

For each $t = 1, 2, \dots$

1. Set

$$p_i(t) = (1 - \gamma) \frac{w_i(t)}{\sum_{j=1}^K w_j(t)} + \frac{\gamma}{K}$$
 for $i = 1, \dots, K$.
2. Select randomly an arm i_t according the the probabilities $p_1(t), \dots, p_K(t)$
3. Observe the reward $r_{i_t}(t)$
4. Update the weight of i_t by

$$w_{i_t}(t+1) = w_{i_t}(t) \exp\left(\frac{\gamma r_{i_t}(t)}{K p_{i_t}(t)}\right)$$
 and set $w_j(t+1) = w_j(t)$ for other arms.

• In order to find good actions EXP3 maintains a balance between

- **exploration:** weighting actions according to the reward: $1 - \gamma$ term in the probability, and
- **exploration:** the uniform term $\frac{\gamma}{K}$ ensures that unsuffi-

External regret and Zero-Sum Matrix Games

- Two players simultaneously choose a column i and a line j of a given matrix M
- The line player receives $M_{i,j}$ and the column player receives $-M_{i,j}$.

Example : the Rock-Paper-Scissors game

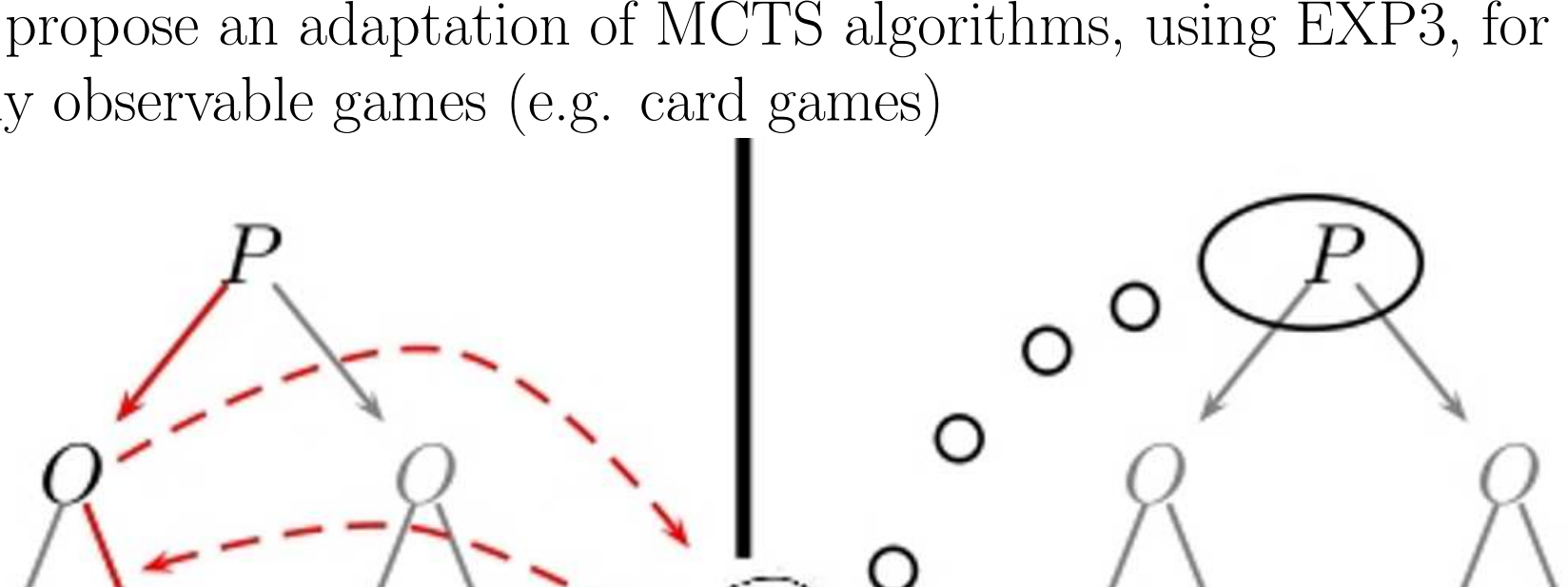
	Rock	Cissors	Paper
Rock	0	+1	-1
Cissors	-1	0	+1
Paper	+1	0	-1

If both players repeatedly play, selecting their strategies with an algorithm minimizing external regret, then the empirical frequencies of play converge to optimal strategies (“Nash Equilibrium”).

- ## Monte-Carlo Tree Search Algorithms
- MCTS Algorithms are efficient algorithms designed for tree-searching with huge inputs.
 - They have been used to design computer players in games with full observation, e.g. Go
 - Mogo was the first algorithm to win against professional Go players
-
- The diagram illustrates the four phases of Monte-Carlo Tree Search (MCTS) within a triangular search space:
- Selection:** A red solid line with black dots at each node, representing the path chosen based on bandit-based selection from the root of the **Search Tree** down to a **New Node**.
 - Expansion:** A green dashed line starting from the **New Node** and extending to a new node at the bottom, representing the addition of a new node to the **Explored Tree**.
 - Simulation:** The green dashed line continues from the new node to a terminal state (a circle with a dot), representing a random simulation.
 - Backpropagation:** A red solid line starting from the terminal state and moving up through the nodes, representing the backpropagation of the simulation result.
- As in the multi-armed bandit problem we must balance *exploitation* and *exploration*
 - classical implementations [KS06] use the UCB bandit algorithm [LR85], designed for the stochastic setting.

Multiple Tree Monte-Carlo Tree Search

We propose an adaptation of MCTS algorithms, using EXP3, for partially observable games (e.g. card games)



The diagram illustrates the Multiple Tree Monte-Carlo Tree Search (MCTS) algorithm for partially observable games. It shows a player (Smurf) at the bottom, interacting with a game tree. The tree is divided into two parts by a vertical line. The left part shows a game tree with a player node (O) and a chance node (P). The right part shows a game tree with a player node (O) and a chance node (P). The player node (O) is highlighted with a red dashed line, indicating it is the current node being explored. The chance node (P) is highlighted with a red dashed line, indicating it is the current node being explored.

Monte-Carlo Tree Search Algorithms

- MCTS Algorithms are efficient algorithms designed for tree-searching with huge inputs.
- They have been used to design computer players in games with full observation, e.g. Go
- Mogo was the first algorithm to win against professional Go players

The diagram illustrates the four phases of Monte-Carlo Tree Search (MCTS) within a triangular tree structure:

- Selection:** A red line with black dots at each node, representing the path chosen for expansion from the root to a leaf node.
- Expansion:** A small white square labeled "New Node" is added to the end of the selected path.
- Simulation:** A green dashed line with black dots, representing a random walk from the new node to a terminal state (a circle with a dot).
- Backpropagation:** A green dashed line with black dots, representing the backpropagation of the simulation result from the terminal state up the tree.

- As in the multi-armed bandit problem we must balance *exploitation* and *exploration*
- classical implementations [KS06] use the UCB bandit algorithm [LR85], designed for the stochastic setting.

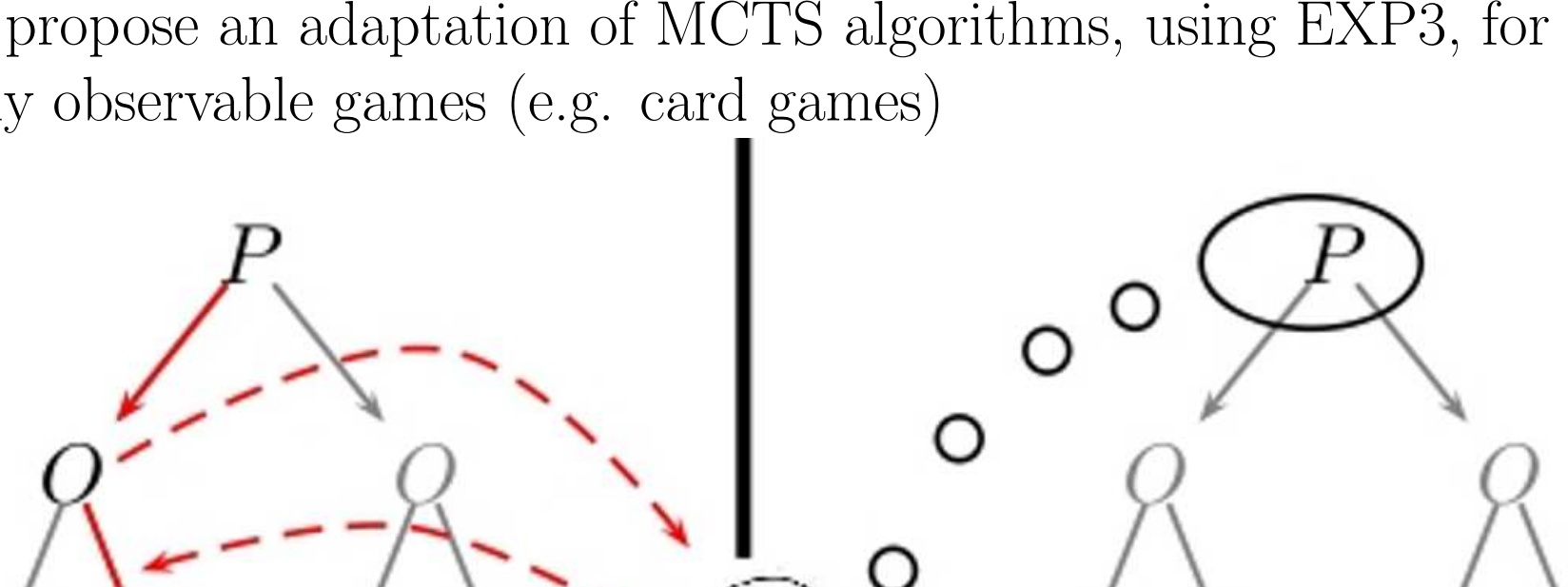
- MCTS Algorithms are efficient algorithms designed for tree-searching with huge inputs.
- They have been used to design computer players in games with full observation, e.g. Go
- Mogo was the first algorithm to win against professional Go players

The diagram illustrates the structure of a Monte Carlo Tree Search (MCTS) tree. It is represented as a large triangle divided into two main regions. The upper region, shaded gray, is labeled "Search Tree". Within this region, a path of nodes connected by red lines is shown, labeled "Bandit-Based Phase". This path starts from a root node at the top and descends through several intermediate nodes. The lower region, which is white, is labeled "Explored Tree". A dashed green line, labeled "Random Phase", connects a node from the "Bandit-Based Phase" path to a new node in the "Explored Tree", labeled "New Node". The "New Node" is a small circle, and the "Explored Tree" region contains several other nodes connected by dashed green lines, representing the part of the tree that has been fully explored.

- As in the multi-armed bandit problem we must balance *exploitation* and *exploration*
- classical implementations [KS06] use the UCB bandit algorithm [LR85], designed for the stochastic setting.

Multiple Tree Monte-Carlo Tree Search

We propose an adaptation of MCTS algorithms, using EXP3, for partially observable games (e.g. card games)



The diagram illustrates the Multiple Tree Monte-Carlo Tree Search (MCTS) algorithm for partially observable games. It shows a player (Smurf) at the bottom, interacting with a game tree. The tree is divided into two parts by a vertical line. The left part shows a game tree with a player node (O) and a chance node (P). The right part shows a game tree with a player node (O) and a chance node (P). The player node (O) is highlighted with a red dashed line, indicating it is the current node being explored. The chance node (P) is highlighted with a red dashed line, indicating it is the current node being explored.

- Multiple Tree Monte-Carlo Tree Search
- We propose an adaptation of MCTS algorithms, using EXP3, for partially observable games (e.g. card games)
-

We propose an adaptation of MCTS algorithms, using EXP3, for partially observable games (e.g. card games)

- Played like standard Tic-Tac-Toe but one does not see where the opponent plays.
- In case of an illegal move the player must play somewhere else

Whereas standard Tic-Tac-Toe is a draw, in Phantom Tic-Tac-Toe the first player can force 85 % of victories with only 4% of losses.

Simulations	wins1-wins2 (solid)	wins 1 (dashed)	wins 2 (dotted)
0	0.00	0.00	0.00
5e+06	0.85	0.70	0.10
1e+07	0.85	0.75	0.10
2e+07	0.85	0.75	0.10
3e+07	0.85	0.75	0.10
4e+07	0.85	0.75	0.10
5e+07	0.85	0.75	0.10

- ## A Phantom Tic-Tac-Toe personal Olympiad

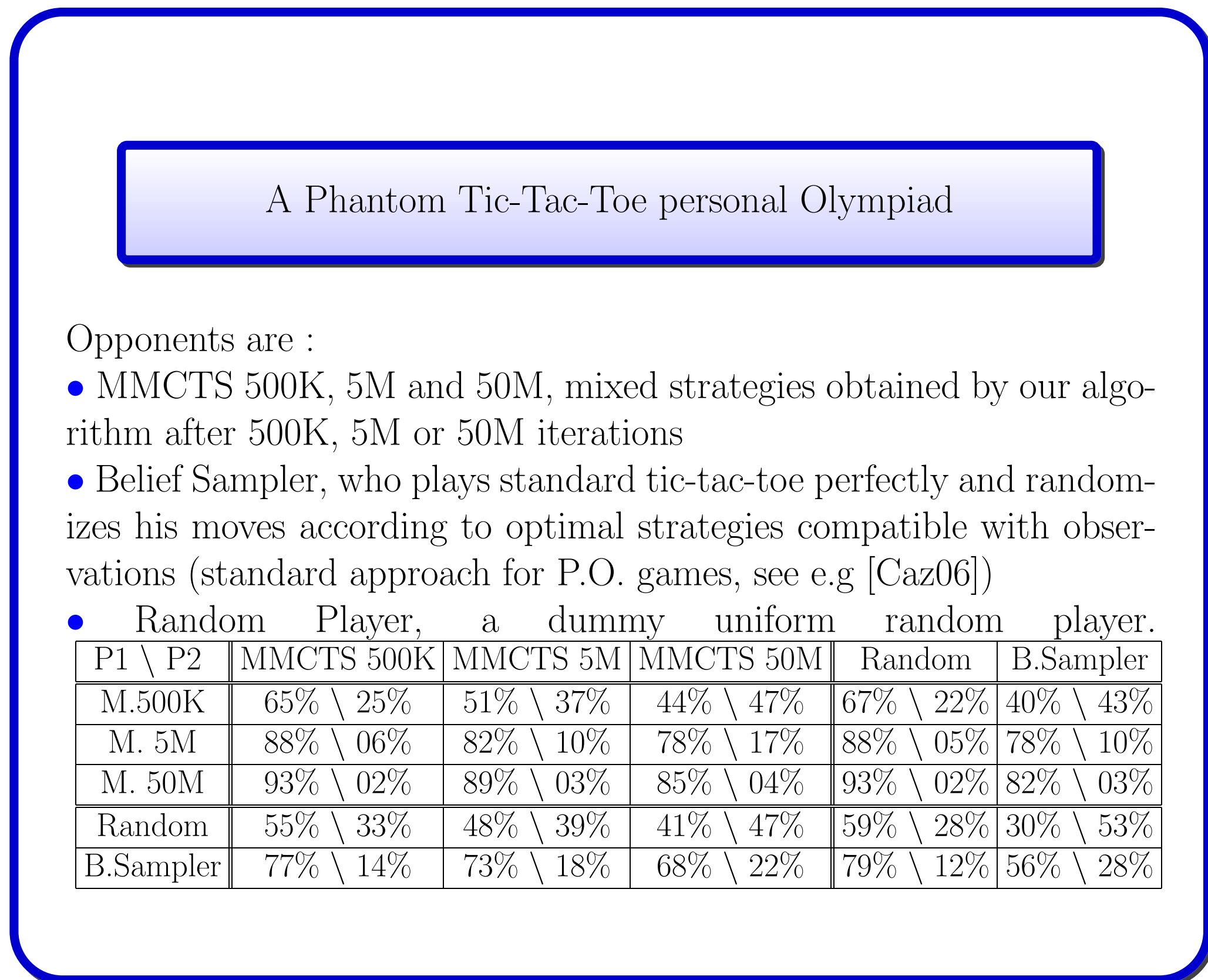
Opponents are :

 - MMCTS 500K, 5M and 50M, mixed strategies obtained by our algorithm after 500K, 5M or 50M iterations
 - Belief Sampler, who plays standard tic-tac-toe perfectly and randomizes his moves according to optimal strategies compatible with observations (standard approach for P.O. games, see e.g [Caz06])
 - Random Player, a dummy uniform random player.

P1 \ P2	MMCTS 500K	MMCTS 5M	MMCTS 50M	Random	B.Sampler
M.500K	65% \ 25%	51% \ 37%	44% \ 47%	67% \ 22%	40% \ 43%
M. 5M	88% \ 06%	82% \ 10%	78% \ 17%	88% \ 05%	78% \ 10%
M. 50M	93% \ 02%	89% \ 03%	85% \ 04%	93% \ 02%	82% \ 03%
Random	55% \ 33%	48% \ 39%	41% \ 47%	59% \ 28%	30% \ 53%
B.Sampler	77% \ 14%	73% \ 18%	68% \ 22%	79% \ 12%	56% \ 28%



The graph displays the results of a simulation over 50 million iterations. The y-axis represents the number of wins, ranging from 0 to 0.9. The x-axis represents the number of simulations, ranging from 0 to 5e+07. Three data series are plotted: 'wins1-wins2' (solid line), 'wins1' (dashed line), and 'wins2' (dotted line). The 'wins1-wins2' series starts at 0 and rises sharply to approximately 0.8 by 5e+06 simulations, then fluctuates around that level. The 'wins1' series starts at 0 and rises to approximately 0.7 by 5e+06 simulations, then fluctuates around that level. The 'wins2' series starts at 0 and falls sharply to approximately 0.05 by 5e+06 simulations, then fluctuates around that level.



Opponents are :

- MMCTS 500K, 5M and 50M, mixed strategies obtained by our algorithm after 500K, 5M or 50M iterations
- Belief Sampler, who plays standard tic-tac-toe perfectly and randomizes his moves according to optimal strategies compatible with observations (standard approach for P.O. games, see e.g [Caz06])
- Random Player, a dummy uniform random player.

P1 \ P2	MMCTS 500K	MMCTS 5M	MMCTS 50M	Random	B.Sampler
M.500K	65% \ 25%	51% \ 37%	44% \ 47%	67% \ 22%	40% \ 43%
M. 5M	88% \ 06%	82% \ 10%	78% \ 17%	88% \ 05%	78% \ 10%
M. 50M	93% \ 02%	89% \ 03%	85% \ 04%	93% \ 02%	82% \ 03%
Random	55% \ 33%	48% \ 39%	41% \ 47%	59% \ 28%	30% \ 53%
B.Sampler	77% \ 14%	73% \ 18%	68% \ 22%	79% \ 12%	56% \ 28%

- MMCTS 500K, 5M and 50M, mixed strategies obtained by our algorithm after 500K, 5M or 50M iterations
- Belief Sampler, who plays standard tic-tac-toe perfectly and randomizes his moves according to optimal strategies compatible with observations (standard approach for P.O. games, see e.g [Caz06])
- Random Player, a dummy uniform random player.

P1 \ P2	MMCTS 500K	MMCTS 5M	MMCTS 50M	Random	B.Sampler
M.500K	65% \ 25%	51% \ 37%	44% \ 47%	67% \ 22%	40% \ 43%
M. 5M	88% \ 06%	82% \ 10%	78% \ 17%	88% \ 05%	78% \ 10%
M. 50M	93% \ 02%	89% \ 03%	85% \ 04%	93% \ 02%	82% \ 03%
Random	55% \ 33%	48% \ 39%	41% \ 47%	59% \ 28%	30% \ 53%
B.Sampler	77% \ 14%	73% \ 18%	68% \ 22%	79% \ 12%	56% \ 28%

References

[ACBFS03] P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

[Caz06] T. Cazenave. A Phantom-Go program. *Advances in Computer Games*, pages 120–125, 2006.

[KS06] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–

[ACBFS03] P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

[Caz06] T. Cazenave. A Phantom-Go program. *Advances in Computer Games*, pages 120–125, 2006.

[KS06] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–

[Caz06] T. Cazenave. A Phantom-Go program. *Advances in Computer Games*, pages 120–125, 2006.

[KS06] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–

[KS06] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–