

Monte Carlo Permutation Search

Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. We propose Monte Carlo Permutation Search (MCPS), a general-purpose Monte Carlo Tree Search (MCTS) algorithm that improves on the GRAVE algorithm. MCPS is relevant when deep reinforcement learning is not an option, or when the computing power available before play is not substantial, such as in General Game Playing, for example. The principle of MCPS is to include in the exploration term of a node the statistics on all the playouts that contain all the moves on the path from the root to the node. We extensively test MCPS on a variety of games: board games, wargame, investment game, video game and multi-player games. MCPS has better results than GRAVE in all the two-player games. It has equivalent results for multi-player games because these games are inherently balanced even when players have different strengths. We also show that using abstract codes for moves instead of exact codes can be beneficial to both MCPS and GRAVE, as they improve the permutation statistics and the AMAF statistics. We also provide a mathematical derivation of the formulas used for weighting the three sources of statistics. These formulas are an improvement on the GRAVE formula since they no longer use the bias hyperparameter of GRAVE. Moreover, MCPS is not sensitive to the ref hyperparameter.

1 Introduction

Monte Carlo Tree Search (MCTS) [20, 9] has been successfully applied to many games and problems [4, 35]. It originates from the computer game of Go [3] with a method based on simulated annealing [5]. The principle underlying MCTS is to learn which move to play using statistics from random games. In the early times of MCTS, random games were played with a uniform policy. Computer Go programs soon used non-uniform playout policies, learning the policy with optimization algorithms [10]. Playout policies were replaced with neural network evaluations for computer Go with the AlphaGo program [32], and then for other games such as Chess and Shogi with the AlphaZero program [33]. There have been numerous applications of MCTS following its notorious success in Computer Go. Applications to problems other than games include interplanetary trajectory planning [17], real-world disaster response [36], agile legged locomotion [8], multi-robot active perception [1], security games [19], chemical retrosynthesis [30, 16, 26, 2], fluid-structure topology optimization [13], task scheduling [18], vehicle routing [23, 31], and general problem solving [28, 7], among others.

Algorithms that use Monte Carlo Tree Search (MCTS) with playouts instead of evaluation by a trained neural network are useful for domains such as General

Game Playing [25, 15] when the game is not known in advance, or for other applications, such as modeling biological systems where the biological system varies with each run [24]. We aim to develop a general MCTS algorithm that performs well across many problems without parameter tuning, without training a model as in AlphaZero, and without problem-specific modifications. GRAVE [7] is such an algorithm, and we propose to compare it to others in our experiments.

In a recent study [34] related to a Kaggle competition, 268,386 plays were performed among 61 different agents across 1,494 distinct games. The goal of the Kaggle competition was to design the best machine learning model to predict the strength of general MCTS algorithms using different optimizations. A result of the Kaggle competition is that the GRAVE algorithm is the best feature for predicting the strength of general MCTS agents.

We propose to improve on GRAVE using three sources of statistics instead of two as in GRAVE. The first two sources are the same as in GRAVE: the statistics on the playouts that start with a move and the AMAF statistics on the playouts below a node that contain a move. The third source of statistics is on the playouts that contain a sequence of moves in any order. The algorithm is named Monte Carlo Permutation Search (MCPS) as it uses statistics on playouts for which a permutation of the moves of the playout starts with the same sequence of moves as the sequence of moves that reaches the state associated with the node in the search tree. For some games such as Hex, any permutation of the moves of a playout gives the same state as if we directly play the permuted sequence. For other games such as Atarigo, if we replay a permutation, it can lead to a state different from the state at the end of the original playout with an opposite score. Nevertheless, we have found that even for Atarigo, the use of statistics on permutations improves the win rate.

MCPS does not have hyperparameters in the formulas for the coefficients of the three sources of statistics it uses, whereas GRAVE has one hyperparameter in its formula for the coefficient on the two sources of statistics it uses, as well as one hyperparameter for the selection of the ancestor node. MCPS behaves similarly well for multiple different games across various values of the hyperparameter used to select the ancestor node. MCPS has two advantages over GRAVE: it performs better for the same number of playouts as GRAVE, and it does not require optimizing hyperparameters. MCPS requires calculating the statistics on playouts that contain a sequence of moves. We provide code optimizations that enable MCPS to calculate them efficiently.

This paper is organized as follows. The second section describes previous work. The third section presents the combination of statistics used in MCPS. The fourth section details the experimental results. The last section concludes.

2 Previous Work

We start by defining notations for the different statistics used in the paper. We then describe previous algorithms related to MCPS and especially the GRAVE algorithm.

2.1 Notations

Let a state s be defined as the sequence of d actions from the root that reaches s :

$$s = a_0, a_1, a_2, \dots, a_d$$

Let a playout p be defined as the sequence of its actions until the terminal state:

$$p = p_0, p_1, p_2, \dots, p_t$$

Let $P(s)$ be the set of playouts that start from s . Let $avg(P(s))$ be the function that returns the average of the rewards of the playouts in the set of playouts $P(s)$.

We can now define:

$$Q(s, a) = avg(\{p \in P(s) \mid p_0 = a\})$$

$$N(s, a) = |\{p \in P(s) \mid p_0 = a\}|$$

$$N(s) = |\{p \in P(s)\}|$$

and $\tilde{Q}(s, a)$, the All Moves As First (AMAF) statistics at node s :

$$\tilde{Q}(s, a) = avg(\{p \in P(s) \mid a \in p\})$$

$$\tilde{N}(s, a) = |\{p \in P(s) \mid a \in p\}|$$

Let $\hat{Q}(s, a)$ be the permutation statistics at node s that take into account all the playouts that contain the moves of s in any order, with $root$ as the root of the search tree:

$$\hat{Q}(s, a) = avg(\{p \in P(root) \mid a \in p, \forall a_i \in s : a_i \in p\})$$

$$\hat{N}(s, a) = |\{p \in P(root) \mid a \in p, \forall a_i \in s : a_i \in p\}|$$

2.2 Previous Algorithms

The first program to produce statistics on random games in the game of Go was the Gobble program [5]. It also defined the AMAF statistics as a heuristic to use when there are not enough playouts.

MCTS [9] was then designed by memorizing the visited states in a tree and using them to direct the search with the UCB exploration term, leading to the UCT algorithm [20]. It was a revolution in computer game playing.

The same year, Virtual Global Search [6] was proposed. It used statistics on playouts containing a sequence of moves played in any order. These statistics are similar to the $\hat{Q}(s, a)$ statistics we propose to use in combination with usual statistics and AMAF statistics.

An important improvement to UCT was the RAVE algorithm [14]. It combined AMAF statistics with the usual statistics. It was a significant improvement over UCT for computer Go.

The use of transposition tables in UCT was also studied with the Upper Confidence Bound for rooted Directed acyclic graphs (UCD) algorithm [29].

The GRAVE algorithm [7] is a simple modification of RAVE that makes it much better than RAVE for many games. It uses the AMAF statistics of an ancestor node instead of the AMAF statistics of the node, as in RAVE. It is an appropriate algorithm for General Game Playing. It was recently adapted to the continuous case for modeling biological systems [24]. The GRAVE algorithm is presented in Algorithm 1.

MCTS is often used in combination with Deep Learning since the success of the AlphaGo system [32] and subsequent versions such as AlphaZero [33]. The exploration term used by AlphaGo is PUCT which makes use of the prior on the moves given by the policy head. AlphaZero has been applied to numerous problems, including algorithm discovery for matrix multiplication [12] and quantum circuit optimization [27], for example.

3 Combining Statistics

In this section we first detail the mathematical derivation of the formulas for the weights applied to the different statistics. We then adapt these formulas in order to mimic the behavior of RAVE and GRAVE for a large number of playouts. We continue with the description of the MCPS algorithm and finish with code optimization to calculate the permutation statistics efficiently and the design of abstract codes.

3.1 Derivation of the Weights for the Statistics

The mathematical derivation of the formulas for α_* , β_* , and γ_* was conducted by ChatGPT on the 7th of August 2025.

Let $Q_*(s, a)$ be the weighted sum of $Q(s, a)$, $\tilde{Q}(s, a)$, and $\hat{Q}(s, a)$:

$$Q_*(s, a) = \alpha Q(s, a) + \beta \tilde{Q}(s, a) + \gamma \hat{Q}(s, a), \text{ with } \alpha + \beta + \gamma = 1$$

We start as in RAVE [14] by decomposing the mean squared error of the combined value into the bias and variance of the $Q(s, a)$, $\tilde{Q}(s, a)$, and $\hat{Q}(s, a)$ values, respectively (the bias of $\hat{Q}(s, a)$ is $\hat{b}$, and the variance $\hat{\sigma}^2$), making the assumption that these values are independently distributed:

$$e_*^2 = \sigma_*^2 + b_*^2 = \alpha^2 \sigma^2 + \beta^2 \tilde{\sigma}^2 + \gamma^2 \hat{\sigma}^2 + (\alpha b + \beta \tilde{b} + \gamma \hat{b})^2$$

Algorithm 1 The GRAVE algorithm. The number of playouts below state s that start with move a is $N(s, a)$. The number of playouts below state s that contain move a is $\tilde{N}(s, a)$. The average of the scores of the playouts below state s that start with move a is $Q(s, a)$. The average of the scores of the playouts below state s that contain move a is $\tilde{Q}(s, a)$.

Input: N tree-walks, initial state $root$, reference state constant ref
Output: A search tree

```

1: Initialize an empty transposition table
2: for  $i = 1$  to  $N$  do
3:    $s \leftarrow root, s_{ref} \leftarrow root$ 
4:   while  $s$  is in the tree and is not terminal do
5:     if  $n(s) > ref$  then
6:        $s_{ref} \leftarrow s$ 
7:     end if
8:     for each  $a \in$  possible moves do
9:        $\beta \leftarrow \frac{\tilde{N}(s_{ref}, a)}{\tilde{N}(s_{ref}, a) + N(s, a) + bias \times \tilde{N}(s_{ref}, a) \times N(s, a)}$ 
10:       $Q_*(s, a) \leftarrow (1 - \beta) \times Q(s, a) + \beta \times \tilde{Q}(s_{ref}, a)$ 
11:    end for
12:    Select  $a_* \leftarrow \operatorname{argmax}\{Q_*(s, a) \mid a \in \text{possible moves}\}$ 
13:     $s \leftarrow s \cup \{a_*\}$ 
14:  end while
15:  Add  $s$  to the tree as the new leaf
16:  while  $s$  is not a terminal state do
17:    Sample  $a$  from the available moves of  $s$  based on the default policy
18:     $s \leftarrow s \cup \{a\}$ 
19:  end while
20:   $score \leftarrow \text{evaluate}(s)$ 
21:  Update  $Q, N, \tilde{Q}$ , and  $\tilde{N}$  for nodes on the path from the root to the new leaf
22: end for

```

We now consider the problem of minimizing the quadratic function

$$f(\alpha, \beta, \gamma) = \alpha^2 \sigma^2 + \beta^2 \tilde{\sigma}^2 + \gamma^2 \hat{\sigma}^2 + (\alpha b + \beta \tilde{b} + \gamma \hat{b})^2,$$

subject to the constraint

$$\alpha + \beta + \gamma = 1,$$

where $\alpha, \beta, \gamma \in \mathbb{R}$.

Assumptions:

$$b = \tilde{b} = \hat{b}, \quad \text{and} \quad \sigma^2 = \frac{\mu(1-\mu)}{n}, \quad \tilde{\sigma}^2 = \frac{\mu(1-\mu)}{\tilde{n}}, \quad \hat{\sigma}^2 = \frac{\mu(1-\mu)}{\hat{n}}.$$

with

$$n = N(s, a), \quad \tilde{n} = \tilde{N}(s, a), \quad \hat{n} = \hat{N}(s, a)$$

Additionally, assume

$$\mu(1 - \mu) = 0.25.$$

We then look for the minimum of:

$$f(\alpha, \beta, \gamma) = \alpha^2\left(\frac{1}{4n} + b^2\right) + \beta^2\left(\frac{1}{4\tilde{n}} + b^2\right) + \gamma^2\left(\frac{1}{4\hat{n}} + b^2\right) + 2\alpha\beta b^2 + 2\alpha\gamma b^2 + 2\beta\gamma b^2$$

Let $\mathbf{x} = (\alpha, \beta, \gamma)^\top$ and define

$$D = \text{diag}\left(\frac{1}{4n}, \frac{1}{4\tilde{n}}, \frac{1}{4\hat{n}}\right), \quad \mathbf{1} = (1, 1, 1)^\top.$$

The quadratic objective can be written as

$$F(\mathbf{x}) = \mathbf{x}^\top Q \mathbf{x}, \quad Q = D + b^2 \mathbf{1} \mathbf{1}^\top,$$

because the diagonal entries of D produce the single-variable terms, and the off-diagonal entries b^2 are given by the rank-one matrix $b^2 \mathbf{1} \mathbf{1}^\top$.

We minimize $F(\mathbf{x})$ subject to the affine constraint $\mathbf{1}^\top \mathbf{x} = 1$. Form the Lagrangian

$$\mathcal{L}(\mathbf{x}, \mu) = \mathbf{x}^\top Q \mathbf{x} - \mu(\mathbf{1}^\top \mathbf{x} - 1).$$

The stationarity condition $\nabla_{\mathbf{x}} \mathcal{L} = 0$ gives

$$2Q\mathbf{x} - \mu\mathbf{1} = 0 \quad \implies \quad Q\mathbf{x} = \frac{\mu}{2} \mathbf{1}.$$

Hence $\mathbf{x}$ is proportional to $Q^{-1}\mathbf{1}$; imposing the normalization $\mathbf{1}^\top \mathbf{x} = 1$ yields the explicit formula

$$\mathbf{x}^* = \frac{Q^{-1}\mathbf{1}}{\mathbf{1}^\top Q^{-1}\mathbf{1}}.$$

To compute $Q^{-1}\mathbf{1}$, use the Sherman–Morrison formula for the rank-one update $Q = D + uu^\top$ with $u = b\mathbf{1}$. Set $y := D^{-1}\mathbf{1}$. Then

$$Q^{-1}\mathbf{1} = (D + uu^\top)^{-1}\mathbf{1} = y - D^{-1}u \frac{u^\top y}{1 + u^\top D^{-1}u}.$$

But $D^{-1}u = bD^{-1}\mathbf{1} = by$ and $u^\top y = b\mathbf{1}^\top y = bS$ where $S := \mathbf{1}^\top y$. Therefore

$$Q^{-1}\mathbf{1} = y - by \frac{bS}{1 + b^2 S} = y \left(1 - \frac{b^2 S}{1 + b^2 S}\right) = \frac{y}{1 + b^2 S}.$$

Thus $Q^{-1}\mathbf{1}$ is proportional to $y = D^{-1}\mathbf{1}$.

Since $D^{-1}\mathbf{1} = (4n, 4\tilde{n}, 4\hat{n})^\top$, we obtain after normalization

$$\alpha^* = \frac{n}{n + \tilde{n} + \hat{n}}, \quad \beta^* = \frac{\tilde{n}}{n + \tilde{n} + \hat{n}}, \quad \gamma^* = \frac{\hat{n}}{n + \tilde{n} + \hat{n}}.$$

Finally, Q is positive definite because D is diagonal with positive entries, and $b^2 \mathbf{1} \mathbf{1}^\top$ is positive semidefinite; hence, the quadratic is strictly convex, and the stationary point above is the unique global minimizer. The positivity of the weights follows immediately from $n, \tilde{n}, \hat{n} > 0$.

3.2 Behavior for a Large Number of Playouts

Using directly the above formulas for the parameters leads to a different behavior than RAVE and GRAVE for nodes with a large number of playouts. In order to recover behavior similar to RAVE and GRAVE, we use a weight c_1 for the α parameter. For large numbers of playouts, α should tend to $\frac{n}{n+\tilde{n}}$ behave similarly to RAVE and GRAVE. So we pose the equation:

$$\frac{c_1 \times n}{c_1 \times n + \tilde{n} + \hat{n}} = \frac{n}{n + \tilde{n}}$$

thus,

$$c_1 \times (n + \tilde{n}) = c_1 \times n + \tilde{n} + \hat{n}$$

$$(c_1 - 1) \times \tilde{n} = \hat{n}$$

and,

$$c_1 = \frac{\tilde{n} + \hat{n}}{\tilde{n}}$$

In the algorithm, we use the following formulas for the weights:

$$\alpha = \frac{c_1 \times n}{c_1 \times n + \tilde{n} + \hat{n}}, \quad \beta = \frac{\tilde{n}}{c_1 \times n + \tilde{n} + \hat{n}}, \quad \gamma = \frac{\hat{n}}{c_1 \times n + \tilde{n} + \hat{n}}$$

3.3 The MCPS Algorithm

The MCPS algorithm is given in Algorithm 2. The difference from GRAVE is primarily the exploration term used in line 9. At line 21, it also maintains the set of playouts that have been played in order to compute the $\hat{Q}$ statistics and the associated $\hat{n}$.

3.4 Computing $\hat{Q}(s, a)$

A naive implementation of the $\hat{Q}(s, a)$ statistics can be inefficient. We now describe implementation details that enable us to efficiently compute these statistics.

We keep as a global variable a matrix of boolean of size (number of possible moves, maximum number of playouts). For each possible move, we then have a boolean array of the size of the maximum number of playouts. After each playout, we dynamically assign for each move of the playout the True value at the index of the playout.

Before each tree descent, a vector of booleans is initialized at true. It has the size of the maximum number of playouts. During tree descent, for each move

Algorithm 2 The MCPS algorithm. The main differences with the GRAVE algorithm are lines 9 and 21. An array of booleans that contains True for all the indices of the code of moves played during a playout is kept for each playout. The $\hat{Q}(s, a)$ statistics are computed using the set of playouts that contain the sequence of moves from the root. This set of playouts is efficiently computed using the indices of the codes of the moves from the root that are True in the arrays.

Input: N tree-walks, initial state $root$, reference state constant ref
Output: A search tree

```

1: Initialize an empty transposition table
2: for  $i = 1$  to  $N$  do
3:    $s \leftarrow root, s_{ref} \leftarrow root$ 
4:   while  $s$  is in the tree and is not terminal do
5:     if  $n(s) > ref$  then
6:        $s_{ref} \leftarrow s$ 
7:     end if
8:     for each  $a \in$  possible moves do
9:        $Q_*(s, a) \leftarrow \alpha Q(s, a) + \beta \hat{Q}(s_{ref}, a) + \gamma \hat{Q}(s, a)$ 
10:    end for
11:    Select  $a_* \leftarrow \operatorname{argmax}\{Q_*(s, a) \mid a \in \text{possible moves}\}$ 
12:     $s \leftarrow s \cup \{a_*\}$ 
13:  end while
14:  Add  $s$  to the tree as the new leaf
15:  while  $s$  is not a terminal state do
16:    Sample  $a$  from the available moves of  $s$  based on the default policy
17:     $s \leftarrow s \cup \{a\}$ 
18:  end while
19:  score  $\leftarrow \text{evaluate}(s)$ 
20:  Update  $Q, N, \hat{Q}$ , and  $\tilde{N}$  for nodes on the path from the root to the new leaf
21:   $P(root) \leftarrow P(root) \cup s$ 
22: end for
```

played, a logical and is applied between this vector and the vector of playouts that contain the move. Then $\hat{n}$ at a node is the sum of this vector and $\hat{Q}(s, a)$ is the sum of the scores of the playouts in the vector, divided by $\hat{n}$.

Another optimization is to keep for each state the last playout number that was used to calculate the statistics and start from this number at the next visit to update the statistics.

3.5 Codes for the Moves

In order to make statistics on moves, each move is associated with an integer. The integer is used as a code for the move, and the number of playouts that contain the move, as well as the sum of rewards of these playouts, are associated with the code of the move. At each node, we make statistics on codes for all players so that the AMAF statistics can be reused in descendants.

Exact codes use a bijection between moves and integers. We also use abstract codes for moves in some games. An abstract code encodes only part of the move. Using abstract codes is faster since the number of possible codes is smaller than the number of exact codes, and it also uses less memory. Moreover, an abstract code appears more often in playouts than an exact code. For MCPS and GRAVE, it means that $\tilde{n}$ and $\hat{n}$ are greater, which reinforces the weights of $\tilde{Q}(s, a)$ and $\hat{Q}(s, a)$.

When designing an abstract code there can be many possibilities. The principle is to retain the main information from the move and to remove the unnecessary ones. If the code is too abstract it can appear in almost all playouts, making the statistics useless. Abstract codes can be useful both for GRAVE and MCPS. MCPS makes a heavier use of abstract codes since they are used both for $\tilde{Q}(s, a)$ and for $\hat{Q}(s, a)$.

Table 1. MCPS for Atarigo, Breakthrough, Gomoku, Hex, Knightthrough and Nogo. The opponent is GRAVE with the same number of playouts as MCPS. The table gives the win rates for the different games with 1000 and 5000 playouts. Each win rate is calculated using 800 games between MCPS and GRAVE with alternating colors. For the All games lines, each win rate is the average of 4800 games. MCPS is better than GRAVE or has at least a similar level for all games. It improves more than GRAVE when increasing the number of playouts from 1000 to 5000. In Gomoku there are possible draws, they are counted as 0.5 for both sides. MCPS is particularly good at Hex where permutations are natural. The experiments are made by running 200 processes in parallel.

Game	N	ref=50	ref=100	ref=200	ref=400
Atarigo 6x6	1000	56.50 ± 3.44%	55.62 ± 3.44%	58.50 ± 3.41%	59.50 ± 3.40%
Atarigo 6x6	5000	60.25 ± 3.39%	59.38 ± 3.40%	60.25 ± 3.39%	65.25 ± 3.30%
Breakthrough 8x8	1000	54.75 ± 3.45%	54.75 ± 3.45%	56.12 ± 3.44%	53.12 ± 3.46%
Breakthrough 8x8	5000	57.00 ± 3.43%	58.25 ± 3.42%	57.25 ± 3.43%	57.38 ± 3.43%
Gomoku 9x9	1000	53.31 ± 3.46%	50.25 ± 3.46%	52.94 ± 3.46%	52.75 ± 3.46%
Gomoku 9x9	5000	61.19 ± 3.38%	60.50 ± 3.39%	58.88 ± 3.41%	56.69 ± 3.43%
Hex 7x7	1000	65.88 ± 3.29%	68.50 ± 3.22%	71.75 ± 3.12%	70.38 ± 3.16%
Hex 7x7	5000	74.12 ± 3.03%	75.38 ± 2.99%	77.12 ± 2.91%	77.50 ± 2.89%
Knightthrough 8x8	1000	52.62 ± 3.46%	53.50 ± 3.46%	52.75 ± 3.46%	47.88 ± 3.46%
Knightthrough 8x8	5000	56.38 ± 3.44%	53.87 ± 3.45%	56.00 ± 3.44%	55.00 ± 3.45%
Nogo 5x5	1000	59.62 ± 3.40%	59.38 ± 3.40%	58.38 ± 3.42%	55.62 ± 3.44%
Nogo 5x5	5000	63.25 ± 3.34%	64.62 ± 3.31%	61.75 ± 3.37%	57.12 ± 3.43%
All games	1000	57.11 ± 1.40%	57.00 ± 1.40%	58.41 ± 1.39%	56.54 ± 1.40%
All games	5000	62.03 ± 1.37%	62.00 ± 1.37%	61.88 ± 1.37%	61.49 ± 1.38%

4 Experimental Results

In our implementation of GRAVE and MCPS, we use a transposition table to store information about the states in the search tree [29]. It means that the permutations of moves in the tree are already naturally handled. The permutations taken into account with MCPS deal with the permutations of the entire play-out, not only the permutations of the path to the node that have already been accounted for with the transposition table.

For two-player games and for each parameter configuration, we run 800 games between MCPS and GRAVE: 400 games playing first and 400 games playing second. MCPS and GRAVE always use the same numbers of playouts during their search. For three-player games, we also run 800 games: one third of the games playing first, one third of the games playing second, and one third of the games playing third.

All of the experiments are made by running processes in parallel. Each process is assigned a random seed. The random seeds range from 0 to 799. Experiments are made with 1 000 and 5 000 playouts in order to see whether MCPS scales better than GRAVE. The number of playouts used by both MCPS and GRAVE is denoted by N in the tables. The games are coded in Python, so a match with 5 000 playouts already takes half a day running 200 processes in parallel.

4.1 Board Games

In order to test the generality of MCPS and to evaluate it in different contexts, we make it play against GRAVE for six different board games. The first board game is Atarigo 6x6. This is a simplification of the game of Go that is often used to teach beginners the game of Go. It is played on a 6x6 Go board, and the first player to capture wins. The second game is Breakthrough 8x8. It won the 2001 8x8 game design competition, and it has very often been used in General Game Playing competitions. Each player has two rows of pawns at the beginning of the game, and the winner is the player who first reaches the opposite side of the board. The third game is Gomoku. It is usually played on a 15x15 board. We compare the algorithms for a 9x9 board. The fourth game is Hex. We use the 7x7 board. The goal of Hex is to connect the two opposite sides of the board. The pie rule is used: in our case we force the first move to be in cell (2,2) and assume the second player does not swap in this case. This makes the game balanced. If we do not use the pie rule the first player has a great advantage. It is even proved that the first player wins in this case. The fifth game is Knightthrough 8x8. This game was invented for the General Game Playing competition. It is similar to Breakthrough except that pawns are replaced with knights that can only move forward. The last game is Nogo 5x5. It is the misere version of Atarigo. Suicides and captures are forbidden. The first player who cannot move has lost.

We compare MCPS to GRAVE with the same number of playouts as MCPS. The parameters for GRAVE are $ref = 50$ and $bias = 10^{-5}$. These standard values for ref and $bias$ are the values that work well for many games [7]. For MCPS, we use $ref \in [50, 100, 200, 400]$ for all games. We also vary the number

of playouts from 1000 to 5000. Each result is the average outcome of 800 games. Table 1 presents the results of games between MCPS and GRAVE for these six distinct games. MCPS beats GRAVE on average and improves more than GRAVE when the number of playouts is increased for both GRAVE and MCPS. The win rate does not appear to be highly dependent on the ref parameter. It means a standard ref parameter is fine for MCPS for all games and that MCPS does not require hyperparameter optimization. Moreover, the bias parameter of GRAVE and RAVE is removed in MCPS, which makes MCPS parameter-free while still being better than GRAVE.

Table 2. MCPS for a simple target selection sequential Wargame. For 20 units (10 units against 10 opposing units), MCPS wins 52.50% of the time using 1000 playouts against GRAVE with the same number of playouts. Increasing the number of playouts to 5000 for both MCPS and GRAVE makes MCPS wins 55.88% of the time against GRAVE. Increasing the number of units to 40 gives similar results. The experiments are made by running 800 games, the random seeds range from 0 to 799.

Game	Units	N	ref = 50
Wargame Abstract Codes	20	1000	52.50 $\pm$ 3.46%
Wargame Abstract Codes	20	5000	55.88 $\pm$ 3.44%
Wargame Abstract Codes	20	10000	58.50 $\pm$ 3.41%
Wargame Abstract Codes	40	1000	49.00 $\pm$ 3.46%
Wargame Abstract Codes	40	5000	55.00 $\pm$ 3.45%
Wargame Abstract Codes	40	10000	54.50 $\pm$ 3.45%

4.2 Wargame

We define a simple target selection sequential Wargame. The game is similar to the combat of groups of units in RTS games. The order in which the units attack is fixed, and players alternate turns. The game starts with unit 1 of player 1 attacking a target, followed by unit 1 of player 2 attacking a target, and then unit 2 of player 1 attacking a target, and so on. Each attack deals one point of damage to the target unit. All units start with 3 points of health. Both players have the same number of units at the start. The game is over when one of the players has lost all of their units. The winner receives a reward of 1, while the loser receives 0. Table 2 gives the results for different numbers of units and different numbers of playouts per move.

Both GRAVE and MCPS use an integer as a code for moves in order to make statistics on moves independent of the states. In the Wargame, an abstract code encodes the attacking unit and the target unit as an integer.

Table 3. MCPS versus GRAVE for the Investment Pair Game. MCPS is better than GRAVE and it scales better than GRAVE with the size of the game and the number of playouts. Experiments use both exact codes for moves and abstract codes for moves. For the more complex 21x21 game MCPS using abstract codes gets better results against GRAVE with abstract codes than MCPS with exact codes against GRAVE with exact codes. The experiments are made by running 800 games, the random seeds range from 0 to 799.

Game	N	ref = 50
Investment Pair Game 11x11	1000	56.38 $\pm$ 3.44%
Investment Pair Game 11x11	5000	60.12 $\pm$ 3.39%
Investment Pair Game 21x21	1000	56.12 $\pm$ 3.44%
Investment Pair Game 21x21	5000	58.63 $\pm$ 3.41%
Investment Pair Game 11x11 Abstract Codes	1000	55.12 $\pm$ 3.45%
Investment Pair Game 11x11 Abstract Codes	5000	58.25 $\pm$ 3.42%
Investment Pair Game 21x21 Abstract Codes	1000	56.38 $\pm$ 3.44%
Investment Pair Game 21x21 Abstract Codes	5000	65.25 $\pm$ 3.30%

This game is related to micro management in Real Time Strategy (RTS) games. Micro management has been studied in the context of Genetic Algorithms [21, 22], by imitating experts [37] or with competing approaches [11].

4.3 The Investment Pair Game

We designed an investment game that has a structure inspired by the previous Wargame. However, players are now paired together. There are two teams containing the same number of players. Each player has a corresponding opponent who plays just after or just before them. The game is sequential, with teams alternating their plays. A move consists of producing one unit of wealth and giving it to a player on the team, which includes oneself. When one of the players in a pair has reached a wealth of 3, he has won the sub-game against the paired opponent. This opponent can no longer play and must pass when it is their turn. The winner of the game is the team with the most non-passing players when all sub-games are terminated.

Experimental results for the Investment Pair Game are given in table 3. We use 11x11 and 21x21 for the number of players on the two teams since odd numbers avoid draws. We can observe from the first four lines of the table that MCPS has better results than GRAVE and that MCPS benefits more from additional playouts than GRAVE. The last four lines utilize abstract codes for moves. In the previous results, the code for a move involved coding players and their wealth. Here, the abstract code of a move consists only of the player who is to move and the target player. This code still detects the permutations of moves that reach the same state of the game, but it enables us to collect more statistics on the permutations used in MCPS. We can observe that it is beneficial for the

21x21 game, where it achieves a 65.25% winning rate against GRAVE, while coding the wealth of the players only reaches 58.63%.

Table 4. MCPS against GRAVE for the video game inspired by TFT. MCPS and GRAVE are close to each other. Abstract codes encoding the attacker and the target units are used for both algorithms. The experiments are made by running 800 games, the random seeds range from 0 to 799.

Game	Size	Units	N	ref = 50
Video Game	5x5	10x10	1000	49.38 $\pm$ 3.46%
Video Game	5x5	10x10	5000	54.25 $\pm$ 3.45%

4.4 Video Game

Team Fight Tactics (TFT) is a popular video game which makes creatures from League of Legends fight against each other in tactical mode. Taking inspiration from TFT, we created a video game where agents fight on a small map. They have a range of attack, health, and they can move to the next cell at each turn. The game is sequential. There are two teams of agents, and play alternates between agents of the two teams.

Table 4 gives the result of MCPS against GRAVE, both with abstract codes. The abstract code encodes the attacker agent and the targeted agent. This abstract code has many fewer possible values than the exact code that also encodes the cell of the agent and its move. We tried even more abstract codes, such as only encoding the target unit and its health. However, these abstract codes appear in almost all payouts and are less beneficial to GRAVE and MCPS than encoding the attacker and the target.

MCPS and GRAVE are close to each other for this video game. MCPS improves slightly more than GRAVE when they both have an increased number of payouts.

4.5 Multi-player Games

We tested MCPS for three different multi-player games: Three-player Nogo, the Three-player Wargame, and the Three-player Investment Pair Game. The results are given in Table 5.

Three-player Nogo Three-player Nogo is Nogo played by three players using green stones for the third player. In designing Three-player Nogo, we had to make a decision on the reward at the end of a game. A first try was to give a 0.5 reward to the other players when a player has no moves left and loses. However,

Table 5. MCPS for multi-player games. The experiments are made by running 800 processes, the random seeds range from 0 to 799.

Game	N	ref = 50
Three Player Nogo 5x5	1000	$35.75 \pm 2.71\%$
Three Player Nogo 5x5	5000	$34.83 \pm 2.70\%$
Three Player Wargame 10x10x10	1000	$33.50 \pm 3.27\%$
Three Player Wargame 10x10x10	5000	$32.88 \pm 3.26\%$
Three-player Investment Pair Game 10x10x10	1000	$35.08 \pm 2.70\%$
Three-player Investment Pair Game 10x10x10	5000	$33.12 \pm 2.86\%$

this makes a player’s statistics cap at 50% and it does not decide well between the players, as strong players still have to share their gains. So we decided to have a unique winner. When a player has no more moves, he has lost and passes, and the game continues for the other players. The games end when only one player has not lost, and he is the winner with a reward of 1.

In regular Nogo, a move has roughly one chance out of two to be played in a playout. A sequence of n moves has then roughly a chance $\frac{1}{2^n}$ of being present in a playout, taking into account the permutations. In Three-player Nogo, a move has roughly one chance out of three to be present in a playout and a sequence of moves $\frac{1}{3^n}$. This makes permutation statistics in the tree much less numerous than for regular Nogo. Moreover, the weaker players can also make an implicit coalition. This explains why MCPS behaves only slightly better than GRAVE for Three-player Nogo.

Three-player Wargame We extended the Wargame to three players. The abstract codes are used. The abstract code encodes the attacking unit and the target unit. We found that the Three-player Wargame is a balanced game. When a player takes some advantage, the two other players make an implicit coalition against him. This makes the game balanced even if a player is stronger than the two others.

Three-player Investment Pair Game In this game, the moves of different agents for a given company are not cumulative. Each agent can only reinforce a company of his team. So coalitions do not work as in the three-player Wargame, and when an agent is ahead in his pair, he can keep the advantage provided he plays as often as the other agents. The Wargame is a game of destruction, while the Investment Pair Game is a game of construction. The reward in this game is shared between the agents that win the greatest number of pairs. MCPS and GRAVE are close for this game.

Table 6. MCPS with abstract codes against GRAVE with exact codes. MCPS is much better than GRAVE with exact codes. The winrate increases with the budget of playouts. Comparing to Tables 2 and 4 shows that abstract codes improve both GRAVE and MCPS. Each winrate is calculated using 800 games between MCPS and GRAVE.

Game	Units	N	ref = 50
Wargame	10x10	1000	65.75 $\pm$ 3.29%
Wargame	10x10	5000	72.12 $\pm$ 3.11%
Wargame	20x20	1000	57.00 $\pm$ 3.43%
Wargame	20x20	5000	72.50 $\pm$ 3.09%
Video Game 5x5	10x10	1000	49.50 $\pm$ 3.46%
Video Game 5x5	10x10	5000	63.38 $\pm$ 3.34%
Video Game 5x5 specific codes	10x10	1000	72.75 $\pm$ 3.09%
Video Game 5x5 specific codes	10x10	5000	84.50 $\pm$ 2.51%

4.6 Abstract Codes

In order to evaluate the usefulness of abstract codes for moves, we made MCPS with abstract codes for moves play against regular GRAVE with exact codes for moves.

Results are given in Table 6. We can observe that MCPS with abstract codes is much better than GRAVE with exact codes.

Comparing the results for GRAVE with exact codes to the results for GRAVE with abstract codes in Table 2 and Table 4, we can see that GRAVE also benefits from abstract codes.

For the 5x5 Video Game we tested the same abstract codes as the one used in Table 4: encoding the attacking and the target units. We also tried a different, more specific code encoding the arrival cell of the attacking unit, the move, the health of the target unit, and the player. This more specific code has better results against GRAVE with exact codes.

4.7 Statistics on Codes

Abstract codes should not be overly abstract. Otherwise, they are present in all playouts, and they are not discriminative enough. If they are too specific, they are present in only a small fraction of the playouts, and the weight of the permutation statistics in MCPS is too small to make a difference with GRAVE.

Table 7 gives the statistics on the average length of playouts and the frequency of exact and abstract codes in playouts from the initial state.

For board games, MCPS has the best performance in Hex (74.12%) and Nogo (63.25%). These are also the games that have the highest probability of code presence in the playouts and where all permutations of moves lead to the

Table 7. Statistics on the presence of codes in playouts. For board games the abstract does are the same as the exact codes. For the wargame and the video game abstract codes appear much more often in playouts than exact codes. Statistics are made with 10 000 playouts starting from the initial state of the games.

Game	Size	#moves	Length	Exact	Abstract
Atarigo	6x6	36	24.165	0.343	0.343
Breakthrough	8x8	22	64.100	0.423	0.423
Gomoku	9x9	81	53.426	0.333	0.333
Hex	7x7	48	41.300	0.435	0.435
Knightthrough	8x8	40	33.624	0.210	0.210
Nogo	5x5	25	22.023	0.450	0.450
Wargame	10x10	10	54.448	0.116	0.297
Wargame	20x20	20	110.525	0.059	0.157
Video Game	5x5	17	55.353	0.060	0.256
Video Game specific codes	5x5	17	55.353	0.060	0.131
Investment Pair Game	11x11	11	46.212	0.372	0.901
Investment Pair Game	21x21	21	87.559	0.353	0.890
Three Player Nogo	5x5	25	20.844	0.292	0.292
Three Player Wargame	10x10x10	20	84.520	0.058	0.159
Three Player Investment Pair Game	10x10x10	10	52.254	0.295	0.847

same state. MCPS has the worst performance in Knightthrough (56.38%) and it is also the game with the smallest probability of presence of code in the playouts.

For the Wargame, the Video Game, and the Investment Pair Game, the presence of the abstract code is two to four times higher than the presence of the exact code. As we have seen, MCPS with abstract codes is much better than GRAVE with exact codes.

For the Three-player Wargame and the Three-player Investment Pair Game, abstract codes are approximately three times more frequent than exact codes. However, due to the game, the better use of statistics does not translate into an increased winning rate.

5 Conclusion

MCPS makes use of statistics on the playouts that contain a sequence of moves in any order. We derived analytically the formulas for the parameters of the exploration term of MCPS. The modified GRAVE algorithm incorporating this exploration term gives the MCPS algorithm. Experiments on six different board games reveal that MCPS is better than GRAVE and that it improves more than GRAVE when the budget of playouts is increased. Experiments with a simple sequential wargame show that MCPS is better than GRAVE for this game. Other experiments with the Investment Pair Game and the Video Game confirm that MCPS is better than GRAVE and that it takes more advantage than GRAVE of

an increased budget of playouts. For multi-player games, the results are balanced for the reason that the results of these games are inherently balanced.

Using abstract codes instead of exact codes greatly improves the results for the Wargame, the Investment Pair Game, and the Video Game. Using abstract codes for GRAVE also improves GRAVE. The analysis of the benefits that arise from abstract codes, using statistical analysis of the presence of abstract codes in playouts, reveals that abstract codes are useful for both GRAVE and MCPS because there are many more moves with the abstract codes than with the exact code. It is also important to note that the design of abstract code retains the permutation property of moves. For example, in the Wargame, playing any sequence of moves that matches the abstract codes leads to the same state as the sequence of moves that matches the exact codes.

MCPS does not require hyperparameter tuning, in contrast to GRAVE. The bias hyperparameter in RAVE and GRAVE enables behavior close to UCT when a node has a large number of playouts: the β parameter approaches zero, and the usual statistics dominate. For MCPS, the limits of the weights for an infinite number of playouts do not guaranty that the usual statistics dominate in the end. However, the GRAVE and MCPS statistics can converge in practice to the usual statistics when a move dominates, since the GRAVE and MCPS statistics include the usual statistics. In our experiments, we have also observed that the ref hyperparameter is not sensitive and that a default value of 50 works well for all games.

The idea of combining three sources of statistics in MCTS is not specific to MCPS and can be reused for other kinds of statistics.

We hope that MCPS will be used for problems beyond those described in this paper, as it is a general MCTS algorithm.

References

1. Best, G., Cliff, O.M., Patten, T., Mettu, R.R., Fitch, R.: Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research* **38**(2-3), 316–337 (2019)
2. Blackshaw, T.M., Davies, J.C., Spoerer, K.T., Hirst, J.D.: Enhancing monte carlo tree search for retrosynthesis. *Journal of Chemical Information and Modeling* (2025)
3. Bouzy, B., Cazenave, T.: Computer Go: An AI oriented survey. *Artificial Intelligence* **132**(1), 39–103 (2001)
4. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* **4**(1), 1–43 (Mar 2012)
5. Brüggmann, B.: Monte Carlo Go. Tech. rep., Max-Planck-Inst. Phys., Munich (1993)
6. Cazenave, T.: Virtual global search: Application to 9×9 Go. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers. Lecture Notes in Computer Science*, vol. 4630, pp. 62–71. Springer (2006)

7. Cazenave, T.: Generalized rapid action value estimation. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 754–760 (2015)
8. Clary, P., Morais, P., Fern, A., Hurst, J.: Monte-carlo planning for agile legged locomotion. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 28, pp. 446–450 (2018)
9. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers. Lecture Notes in Computer Science, vol. 4630, pp. 72–83. Springer (2006)
10. Coulom, R.: Computing elo ratings of move patterns in the game of Go. ICGA Journal **30**(4), 198–208 (2007)
11. Dubey, R., Louis, S., Gajurel, A., Liu, S.: Comparing three approaches to micro in rts games. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 777–784. IEEE (2019)
12. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatin, M., Novikov, A., R. Ruiz, F.J., Schrittwieser, J., Swirszcz, G., et al.: Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**(7930), 47–53 (2022)
13. Gaymann, A., Montomoli, F.: Deep neural network and monte carlo tree search applied to fluid-structure topology optimization. Scientific reports **9**(1), 15916 (2019)
14. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence **175**(11), 1856–1875 (2011)
15. Genesereth, M.R., Love, N., Pell, B.: General game playing: Overview of the AAAI competition. AI Magazine **26**(2), 62–72 (2005)
16. Genheden, S., Thakkar, A., Chadimová, V., Reymond, J.L., Engkvist, O., Bjerrum, E.: Aizynthfinder: a fast, robust and flexible open-source software for retrosynthetic planning. Journal of cheminformatics **12**(1), 70 (2020)
17. Hennes, D., Izzo, D.: Interplanetary trajectory planning with Monte Carlo Tree Search. In: IJCAI. pp. 769–775 (2015)
18. Hu, Z., Tu, J., Li, B.: Spear: Optimized dependency-aware task scheduling with deep reinforcement learning. In: 2019 IEEE 39th international conference on distributed computing systems (ICDCS). pp. 2037–2046. IEEE (2019)
19. Karwowski, J., Mańdziuk, J.: Double-oracle sampling method for stackelberg equilibrium approximation in general-sum extensive-form games. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 2054–2061 (2020)
20. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: 17th European Conference on Machine Learning (ECML’06). LNCS, vol. 4212, pp. 282–293. Springer (2006)
21. Liu, S., Louis, S.J., Ballinger, C.: Evolving effective micro behaviors in rts game. In: 2014 IEEE Conference on Computational Intelligence and Games. pp. 1–8. IEEE (2014)
22. Liu, S., Louis, S.J., Ballinger, C.A.: Evolving effective microbehaviors in real-time strategy games. IEEE Transactions on Computational Intelligence and AI in Games **8**(4), 351–362 (2016)
23. Mańdziuk, J.: Mcts/uct in solving real-life problems. In: Advances in Data Analysis with Computational Intelligence Methods: Dedicated to Professor Jacek Żurada, pp. 277–292. Springer (2017)

24. Michelucci, R., Pallez, D., Cazenave, T., Comet, J.: Improving continuous monte carlo tree search for identifying parameters in hybrid gene regulatory networks. In: Affenzeller, M., Winkler, S.M., Kononova, A.V., Trautmann, H., Tusar, T., Machado, P., Bäck, T. (eds.) *Parallel Problem Solving from Nature - PPSN XVIII - 18th International Conference, PPSN 2024, Hagenberg, Austria, September 14-18, 2024, Proceedings, Part IV. Lecture Notes in Computer Science*, vol. 15151, pp. 319–334. Springer (2024)
25. Pitrat, J.: Realization of a general game-playing program. In: *IFIP Congress* (2). pp. 1570–1574 (1968)
26. Roucairol, M., Cazenave, T.: Comparing search algorithms on the retrosynthesis problem. *Molecular Informatics* **43**(7), e202300259 (2024)
27. Ruiz, F.J., Laakkonen, T., Bausch, J., Balog, M., Barekatain, M., Heras, F.J., Novikov, A., Fitzpatrick, N., Romera-Paredes, B., van de Wetering, J., et al.: Quantum circuit optimization with alphasolver. *Nature Machine Intelligence* pp. 1–12 (2025)
28. Sabar, N.R., Kendall, G.: Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences* **314**, 225–239 (2015)
29. Saffidine, A., Cazenave, T., Méhat, J.: UCD: Upper Confidence bound for rooted Directed acyclic graphs. *Knowledge-Based Systems* **34**, 26–33 (Dec 2011). <https://doi.org/10.1016/j.knosys.2011.11.014>
30. Segler, M.H., Preuss, M., Waller, M.P.: Planning chemical syntheses with deep neural networks and symbolic ai. *Nature* **555**(7698), 604–610 (2018)
31. Sentuc, J., Ellouze, F., Lucas, J.Y., Cazenave, T.: Learning the bias weights for generalized nested rollout policy adaptation. In: *International Conference on Learning and Intelligent Optimization*. pp. 194–207. Springer (2023)
32. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016)
33. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**(6419), 1140–1144 (2018)
34. Soemers, D.J., Bams, G., Persoon, M., Rietjens, M., Sladić, D., Stefanov, S., Driessens, K., Winands, M.H.: Towards a characterisation of monte-carlo tree search performance in different games. In: *2024 IEEE Conference on Games (CoG)*. pp. 1–4. IEEE (2024)
35. Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J.: Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review* **56**(3), 2497–2562 (2023)
36. Wu, F., Ramchurn, S., Jiang, W., Fischer, J., Rodden, T., Jennings, N.R.: Agile planning for real-world disaster response (2015)
37. Young, J., Hawes, N.: Learning micro-management skills in rts games by imitating experts. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. vol. 10, pp. 195–201 (2014)