Discounting and Pruning for Nested Playouts in General Game Playing

Michael Schofield, Tristan Cazenave, Abdallah Saffidine, Michael Thielscher
School of Computer Science and Engineering
The University of New South Wales
{mschofield, abdallahs, mit}@cse.unsw.edu.au
LAMSADE - Université Paris-Dauphine
cazenave@lamsade.dauphine.fr

Abstract

In the field of General Game Playing (GGP) there is much emphasis on the use of the Monte Carlo playout as an evaluation function when searching intractable game spaces. This facilitates the use of statistical techniques like MCTS and UCT[1], but it requires significant processing overhead. We seek to improve the quality of information extracted from the Monte Carlo playout in three ways. Firstly by nesting the evaluation function inside another evaluation function, secondly by measuring and utilizing the depth of the playout, and thirdly by incorporating pruning strategies that eliminate unnecessary searches and avoid traps[2]. We present a formalism for a Nested Evaluation Function along with a move selection strategy that incorporates a form of discounting and pruning based on playout depth. We show experimental data on a variety of two-player games from past GGP competitions and compare the performance of a Nested Player against a standard, optimised UCT player.

1 Introduction

General Game Playing (GGP) is concerned with the design of AI systems able to take the rules of any game described in a formal language and to play that game efficiently and effectively [Genesereth *et al.*, 2005]. This area of research is growing with much emphasis on improving the AI systems ability to play games with intractable search spaces by extracting as much "insight" from the game as possible.

We focus our attention on the use of the Monte Carlo technique in GGP. This technique is domain independent, that is, the player does not need to construct a different evaluation function from each game. As such it provides a particularly suitable foundation for GGP systems to play previously unknown games without human intervention. The technique is intuitive and simple to implement. Moreover, it can provide an estimated probability distribution for the game outcomes and hence form the basis of some more advanced statistical techniques. But it also has limitations. The principle limitations are that it has a high cost and the results derived from Monte Carlo playouts[3] are predicated on the assumption that all of the roles in the real game select their move randomly. They do not.

This raises a fundamental question: how do we take a simple random technique and improve the quality of the information it produces without a similar increase in cost, specifically in the context of GGP?

In this paper we offer a way to improve cost-effectiveness by improving the quality of the information extracted from the playouts. We achieve this by implementing three different techniques in concert. Firstly we implement a nested playout such that the higher level playouts are not random but heuristically guided, thereby improving the quality of the terminal value. Secondly we consider the depth of the playouts as a measure of the "value of information" in much the same way as discounting is used in other forms of modeling. Thirdly we prune the search to eliminate wasted effort, with special emphasis on avoiding traps, ie. moves that look promising, but eventually fail.

1.1 Clarification

We use several terms that seem similar, but are not. So we offer these clarifications:

- Nested Player - Any GGP player that uses another set of players to calculate its move evaluation function;
- Nested Playout - Any playout using a nested player(s); and
- Monte Carlo Playout - A playout where all move selections are made randomly.

We use the term "search" to describe a special instance of a "tree search". That is, a tree search limited to a depth of one. In other words, we are choosing a move from the list of legal moves without constructing a game tree. And so, all of our playouts begin at depth = 1, with no expansion, no exploration, and no exploitation[4]. This simple form of search is shown in Figure 1.

[1] Monte Carlo Tree Search and Upper Confidence bound applied to Trees.

[2] Any move that looks promising but is not, especially when it takes a long time to reveal the truth.

[3] even UCT exploration uses decision values derived originally from Monte Carlo playouts.

[4] each phases in various forms of Monte Carlo Tree Searches

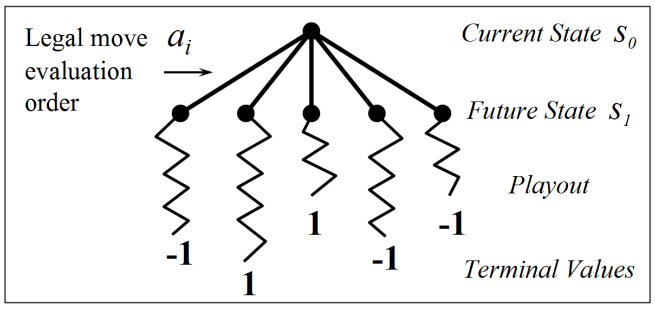

Figure 1: A search using playout terminal values to choose the next legal move a_i.

1.2 Background

The Nested Monte Carlo Search (NMCS) algorithm was proposed as an alternative to single-player Monte Carlo Tree Search (MCTS) for single-player games in [Cazenave, 2009]. NMCS lends itself to many optimizations and improvements, and it has been successful in many single-agent problems [Cazenave, 2010; Akiyama *et al.*, 2010]. In particular, it lead to a new record solution to the Morpion Solitaire mathematical puzzle.

The NMCS technique inspired the Nested Rollout Policy Adaptation algorithm which enabled further record establishing performances in similar domains [Rosin, 2011]. [Méhat and Cazenave, 2010] compare NMCS and UCT for single player games with mixed results, they explore variants of UCT and NMCS and conclude that neither one is a clear winner.

[Pepels *et al.*, 2014] have shown in the context of MCTS that more information than a binary outcome could be extracted from a random playout, even when very little domain knowledge is available. In particular, the outcome of a short playout might be more informative than that of a longer one because fewer random actions have taken place.

The idea of nesting searches of a certain type has been used to distribute the Proof Number Search algorithm over a cluster [Saffidine *et al.*, 2011].

[Chaslot *et al.*, 2009] took advantage of the nesting concept in their Meta-MCTS approach to opening book generation. The book was built through MCTS using self-play games by standard MCTS players in terms of playouts.

Yet another type of nesting was explored for the RecPIMC (Recursive Perfect Information Monte Carlo) search as a way to alleviate the strategy fusion and non-local dependencies problems exhibited by PIMC in imperfect information games [Furtak and Buro, 2013].

Finally we have the [Baier and Winands, 2013] work combining or "nesting" a MiniMax subsearch inside an MCTS with the intent of avoiding traps in tactical situations[5]. They offer three ways to augment the MCTS process with localised MiniMax searches specifically designed to minimise wasted effort.

[5]as distinct from strategic traps, see [Ramanujan *et al.*, 2010]

1.3 Contribution

The NMCS algorithm has been used in a variety of single agent domains and the concepts of discounting and pruning are not new.

We extend the use of NMCS to the general case, implementing it with two-player turn-taking games with win/lose outcomes, and implementing discounting and pruning in a practical way. In this paper, we describe how NMCS can be adapted for these games and propose heuristics that improve its performance.

Our main contributions are as follows:

- A framework for implementing a Nested Playout in two-player win/lose games;
- An improvement to the quality of information produced by the Nested Playout via a discounting heuristic;
- Pruning techniques that improve the cost-effectiveness of the Nested Playout;
- Implementation of a Player using Nested Playouts; and
- Experimental data evaluating a Nested Player for commonly played two-player, win/lose games.

The resulting player will compete favourably with a UCT player that has been optimised for best performance[6]. Experimental results show when the Nested Player is superior to the standard UCT Player and offer insights into why this is so.

2 Nested Playouts

We create a nested playout by wrapping one player around another, simpler, version of itself. At its core we still use a random move selection policy[7]. The level of nesting can be increased by wrapping yet another version around the outside. The only limitation is that the computational costs increase exponentially as the level of nesting increases, the benefit is that the quality of the information from the terminal value of the nested playout also improves.

2.1 Formalism

In our formalism for the Nested Playout we start by adopting a notation for finite games in extensive form by extending the definitions given in [Thielscher, 2011] following the style set out in [Schofield and Thielscher, 2015].

The Game

Let $G = \langle S, R, A, v, \delta \rangle$ be a game determined by a GDL description:

- S is a set of states and R is a set of roles in the game, additionally we use $s_0 \in S$ for the initial state, T for terminal states, and $D = S \backslash T$ for decision states;
- A is a set of moves in the game, and $A(s, r) \subseteq A$ is a set of legal moves, for role $r \in R$ in state $s \in S$;
- $v : T \times R \to \mathbb{R}$ is the payoff function on termination; and
- $\delta : D \times A^{|R|} \to S$ is the joint move successor function.

[6]Optimised for each different game tested so as to play that game as well as it could

[7]based on traditional Monte Carlo playouts

Move Selection Policy
In order for a game to be played out to termination we require a move selection policy π for each role, being an element of the set of all move selection policies Π:

- $\pi \in \Pi : D \times R \rightarrow \phi(A)$ is a move selection policy expressed as a probability distribution across A;
- $\vec{\pi} : \langle \pi_1, \ldots, \pi_{|R|} \rangle$ is a move selection policy tuple; and[8]
- $play : D \times \Pi^{|R|} \rightarrow T$ is the playout of a game to termination according to the given move selection policies.

Move Evaluation Function
Move selection requires an evaluation function $eval()$. We play out the game according to a move selection policy and use the terminal value as a measure of utility:

- $\langle \vec{a}_{-r}, a_r \rangle = \langle a_1 \ldots a_r \ldots a_{|R|} \rangle$ is a move vector containing a specific move a_r for role $r \in R$[9];
- $eval : D \times \Pi^{|R|} \times R \times \mathbb{N} \rightarrow \mathbb{R}$;
- $eval(d, \vec{\pi}, r, n) = \frac{1}{n} \sum_1^n v(play(d, \vec{\pi}), r)$ evaluates the node $d \in D$ using the policies in $\vec{\pi}$ and n playouts;
- $eval(\delta(d, \langle \vec{a}_{-r}, a_{ri} \rangle), \vec{\pi}, r, n)$ is the evaluation of move $a_{ri} \in A(d, r)$ by role r; and
- $a_r = argmax_{a_{ri}}[eval(\delta(d, \langle \vec{a}_{-r}, a_{ri} \rangle), \vec{\pi}, r, n)]$ is the selection process for making a choice.

Nested Evaluation
From the definitions above we take the move selection policy $\pi : D \times R \rightarrow \phi(A)$ of the **parent** player as the maximisation of the evaluation function $eval()$ using move selection policies of the **child** player. This is the basis for a nested playout.

Definition 1. Let the move selection policy for a nested evaluation be defined as:

- $\pi' : a_r = argmax_{a_{ri}}[eval(\delta(d, \langle \vec{a}_{-r}, a_{ri} \rangle), \vec{\pi}, r, n)]$; and
- $\pi'' : a_r = argmax_{a_{ri}}[eval(\delta(d, \langle \vec{a}_{-r}, a_{ri} \rangle), \vec{\pi'}, r, n)]$.

Note that if π were the random move policy, then π' would be a simple Monte Carlo playout; therefore we adopt the following nomenclature for our players:

- NMC(0) is the random player;
- NMC(1) is a simple Monte Carlo player with its move policy based on Monte Carlo playouts; and
- NMC(2) is a nested player using NMC(1) playouts to set its move selection policy, etc..

For example; an NMC(2) player would evaluate each of the 22 opening moves in Breakthrough by playing out a full game using an NMC(1) player for each roles. The two NMC(1) players would, in their turn, use Monte Carlo playouts for each move choice for the length of the game. It is easy to see how the computational cost grows exponential.

[8] we could also say $\vec{\pi}_r$ is the policy tuple used by role r to model all roles behaviour. Our treatment does not use this refinement as all role models are the same, so it is omitted.

[9] this is simplified as we are evaluating turn taking games and the other moves are noop.

3 Heuristic Improvements

The challenge is to improve the cost-effectiveness of the nested playout, and playouts in general. We use the following logic to achieve this;

- A nested playout improves the quality of the evaluation function, but increases the computational cost;
- Using the playout depth for discounting improves the evaluation function quality without increasing cost;
- Using the playout depth facilitates the use of search pruning; and
- Search pruning reduces the cost of the evaluation function without reducing its quality.

3.1 Discounting

When using Monte Carlo playouts, it is common practice to consider only the terminal value $v(t, r)$, from the playout. Here we follow the lead of [Finnsson and Björnsson, 2008] and also consider n, the depth to termination.

Using The Playout Depth
Let $G = \langle S, R, A, v, do \rangle$ be a GDL game described in the previous section, then:

- Without loss of generality, we set the range of the terminal values to $-1 \leq v(t, r) \leq 1$;
- With no prior knowledge, the expected value for a_{ri} is $E(a_{ri}) = e$, ie. all moves have the same expected value;
- Let the estimated branching factor for the sampled branch be $bf \geq 1$, and the estimated depth (number of moves) of the branch be n and $|T| = bf^n$;
- After a single playout revealing a terminal value $v(t, r)$ the expected value of making the same move is $E(a_{ri}) = [(bf^n - 1) \times e + v(t, r)]/bf^n$; and
- The increment is $\delta E(a_{ri}) = (1/bf)^n \times (v(t, r) - e)$;

In many games the branching factor bf may be high, say 30, but as we repeatedly playout out a game there are only a few moves in each round that offer a realistic chance of victory.

In these informed playouts the effective branching factor becomes small, say 2, and the term $(1/bf)^n$ begins to look like a discount factor. Hence we use the term Discounting when considering the depth n of the playout.

Definition 2. Let the move selection policy for our Nested Playout be defined by maximising the increment in the expected value $\delta E(a_{ri}) = (1/bf)^n \times (v(t, r) - e)$.

In operation this is simplified to:

- Maximise $v(t, r)$;
- Settling ties;
  - Case: $max(v(t, r)) < e$, Maximise n;
  - Case: $max(v(t, r)) = e$, Do nothing;
  - Case: $max(v(t, r)) > e$, Minimise n;
- Settle all ties randomly.

Note: In the uninformed case $e = 0$; ie. a drawn game.

The discounting heuristics turns a win/loss game into a game with a wide range of outcomes by having the Max

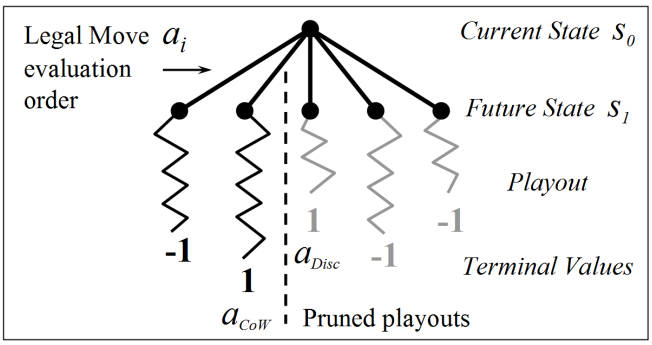

Figure 2: A decision where different heuristics lead different choices, CoW chooses early and prunes a_{Disc}.

player preferring short wins to long wins, and long losses to short losses. The intuition here is to win as quickly as possible so as to minimize our opponent's chance of finding an escape, and to lose as slowly as possible so as to maximize our chance of finding an escape.

A convenient implementation trick is to set a value of $score = v(t,r)/n$ so that a win in depth n moves is $1/n$ and a loss in n moves is $-1/n$. This way, the ordering between game outcomes corresponds exactly to the ordering between the scores considered as real numbers.

Note that our discounting heuristic should not be confused with Steinhauer's discount rate proposed for MCTS [Steinhauer, 2010]. We discount on the length of the playout whereas the latter discounts on how long ago it was performed.

3.2 Pruning

From the classical alpha-beta search [Knuth and Moore, 1975] to the more recent Score-Bounded MCTS adaptation [Cazenave and Saffidine, 2010], two-player games usually lend themselves quite well to pruning strategies. Adapting Nested Playouts to two-player games also gives pruning opportunities that were absent in the single-player case.

Cut on Win

The first pruning strategy, Cut on Win (CoW), hangs on two features of two-player win/loss games. First, we know the precise value of the best outcome of the game. Second, these values are reached regularly in playouts.

The CoW strategy consists of evaluating the moves randomly and selecting the first move that returns the maximum value from its evaluation(s), thus:

- The set of move is evaluated randomly; and
- When $eval()$ returns the maximum value then a choice is made and the remaining evaluations abandoned.

Proposition 1. *Assume that whenever a state is evaluated using a nested playout, move choices are randomly presented in the same order.*

Then NMC(n) with no discounting and no pruning returns the same move as NMC(n) with no discounting and CoW.

Note that when discounting is used, NMC(n) with CoW is not guaranteed to pick the same move as NMC(n). As a counter-example, Figure 2 presents a decision in which NMC(n) with CoW and discounted NMC(n) would select different moves.

Pruning on Depth

The second pruning strategy, Prune on Depth (POD) exploits the use of discounting and the richer outcome structure. Remember the convenient trick of using $score = v(t,r)/n$ to evaluate the game outcomes. With Pruning on Depth we maximise $v(t,r)$ then minimise n thus:

- The set of move is evaluated randomly;
- n is the playout depth;
- $n_{min} = min(n)$ is the minimum depth of any playout to return the maximum score; and
- When $depth(eval()) = n_{min}$ the playout is terminated, returning the goal value of -1.

Put simply, we prune states deeper than the shallowest win.

Proposition 2. *Assume that whenever a state is evaluated using a nested playout, move choices are randomly presented in the same order.*

Then NMC(n) with discounting and no pruning search returns the same move as NMC(n) with discounting and PoD.

Cascade Pruning

The PoD heuristic is like shallow pruning in the alpha-beta algorithm. We now propose a third pruning strategy which is like deep alpha-beta pruning, *Cascade Pruning*.

Whereas in PoD the playout length would only be compared to the length of sibling playouts, in cascade pruning the bound on the length is shared between the parents and the children. That is, the pruning information is cascaded downward from one nesting level to the next. This provides for additional pruning opportunities compared to PoD, but the safety with respect to discounted searches is lost.

The loss of safety can be illustrated in the Breakthrough Challenge in Figure 4. A Monte Carlo playout will often arrive at a favorable outcome via a less than optimal path. For example; Black can counter b6-a7 with b8-a7, but a Monte Carlo playout might make some trivial moves first. If Cascade Pruning is implemented those trivial moves can cause an early termination of the playout returning a loss instead of a safe defense. This impacts the choices made by the higher level player.

3.3 The Trade-off of Unsafe Pruning

Unlike the classical alpha-beta pruning algorithm, the CoW and Cascade Pruning techniques described previously are *unsafe* when using discounting: they may lead to a better move being overlooked.

Unsafe pruning methods are common in the game search community, for instance null move and forward pruning [Smith and Nau, 1994], and the attractiveness of a new method depends on the speed versus accuracy trade-off.

The quality of the choices made by the NMC(3) player is built on both the quality of the choices made by the embedded NMC(2) player and the number of sub-searches that can be performed in a given amount of time. Both aspects can be affected by the pruning policy.

3.4 Embedded MiniMax Search

A close examination of an NMC(n) playout reveals a depth limited MiniMax search being conducted at every step of the playout, very similar to the MCTS-MR proposed by [Baier and Winands, 2013].

Consider an NMC(3) playout being used in a game of Breakthrough for the initial move. The game is at the initial state, $n = 0$. The NMC(3) playouts start at each of the games states of $n = 1$; the embedded NMC(2) playouts start at each of the game states of $n = 2$; and all of the embedded NMC(1) playouts start at each of the game states of $n = 3$.

A similar phenomenon occurs at the bottom of the playout. A depth limited MiniMax at the termination ensures the most correct terminal value is being returned.

4 Experimental Results

4.1 Discounting in TicTacToe

To illustrate discounting we look at the game of TicTacToe in Figure 3. The X player has an opportunity to make the decisive move a_3 = `(does x (mark 3 1))`.

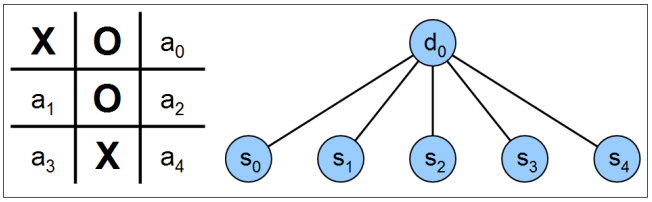

Figure 3: A partially played game of TicTacToe where the X player has the decisive move a_3.

We show the move selection choices made by the NMC(3) player for a 1000 game sample. Terminal values are in the range $-1 \leq v(t,r) \leq 1$ and the values in brackets are (min, max) values.

No Discounting

a_i	$v(t,x)$	$Freq$
a_0	(0, 0)	0
a_1	(0, 1)	176
a_2	(0, 1)	123
a_3	(1, 1)	575
a_4	(0, 1)	126

Discounting

a_i	$v(t,x)$	n	$Freq$
a_0	(0, 0)	(4, 4)	0
a_1	(0, 0)	(4, 4)	0
a_2	(0, 1)	(4, 4)	0
a_3	(1, 1)	(2, 2)	1000
a_4	(0, 1)	(4, 4)	0

Table 1: Results from 1000 games played by a NMC(3) player illustrating the effect that Discounting on depth n has on the frequency of each choice.

Note that, without discounting, moves a_1, a_2 and a_4 are selected occasionally in a tie break. Whereas, with discounting the value of $depth = n$ decides equal values of $v(t,x)$. Also, the embedded NMC(2,Disc) selecting moves for the O player is forcing a draw after move a_1.

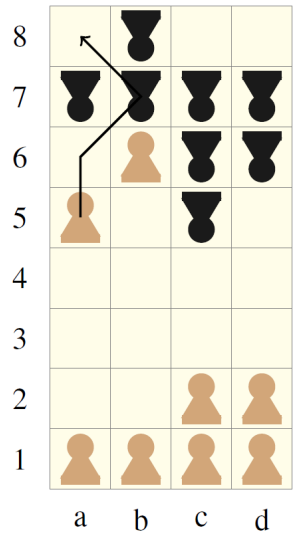

Figure 4: Partially played game of breakthrough. White has a winning move, a5-a6, but also has two incorrect moves b6-a7, b6-c7. The challenge is to avoid the incorrect moves.

4.2 Breakthrough Challenge

As an illustrative example, we look at the game of Breakthrough being played in Figure 4 [Gudmundsson and Björnsson, 2013].[10] This position was used as a test case exhibiting a trap that is difficult to avoid for a plain UCT player. The correct move for White is a5-a6, however b6-a7 and b6-c7 initially look strong.

To better understand the effect of the various heuristics described in the previous sections, we run multiple NMC(3) searches with different parameter settings, starting from the position displayed in Figure 4. For each parameter setting, we record the number of states visited during the search and the likelihood of selecting the correct initial move and present the results in Table 2.

Each entry in the table was an average of 900 games, and the 95% confidence interval on standard error of the mean is reported for each entry. A setting of the type CoW(1) is to be understood as Cut on Win pruning heuristic was activated at nesting level 1 but not at any higher level.

A number of facts can be observed in Table 2:

- If the discounting heuristic is disabled, then the best move is selected with probability around 0.11, but it is identified with probability around 0.63 whenever the discounting heuristic is enabled;
- Both CoW and PoD significantly reduce the number of states explored in a search, and CoW provides more reduction;
- The performance of a non-discounted search is not significantly affected by CoW, as predicted by Proposition 1;

[10] Knowing the rules of Breakthrough is not essential to follow our analysis. However, the reader can find a description of the rules in [Saffidine *et al.*, 2011; Gudmundsson and Björnsson, 2013].

Discounting	Pruning	States Visited(k)	Freq(%)
No	None	$4,459 \pm 27$	11.9 ± 2.2
No	CoW(1)	$1,084 \pm 8$	12.3 ± 2.6
No	CoW(1, 2)	214 ± 2	10.9 ± 2.0
No	CoW(1, 2, 3)	25 ± 1	9.8 ± 2.0
Yes	None	$2,775 \pm 26$	64.1 ± 3.4
Yes	PoD(1)	$1,924 \pm 20$	64.7 ± 3.5
Yes	PoD(1, 2)	$1,463 \pm 16$	58.6 ± 3.5
Yes	PoD(1, 2, 3)	627 ± 19	62.4 ± 3.3
Yes	Cascade	322 ± 16	64.6 ± 2.4

Table 2: Results from 900 games played using a variety of options; showing discounting options, pruning options, the number of states visited and the correct move frequency.

- Similarly the performance of a discounted search is not significantly affected by PoD, as predicted by Proposition 2; and
- Also note that PoD(1,2,3) is 25 times more expensive than CoW(1,2,3), but much more accurate.

If we take a look inside the Nested Playout and examine the quality of the choices being made by the NMC(3), we see that they are built on the quality of the choices being made by the embedded NMC(2) player for both Black and White roles. We can measure this quality by considering Black's response to White's move $does(white\ move\ b6\ a7)$. Remember that this is a trap for White as it eventually fails, but the Black NMC(2) player must spring the trap for the White NMC(3) playout to "get it right".

Black must respond with $does(black\ move\ b8\ a7)$, otherwise White has a certain win. The experimental results, using similar pruning options, were:

- Base Case, $does(black\ move\ b8\ a7) = 15\%$;
- CoW, $does(black\ move\ b8\ a7) = 15\%$;
- Discounting, $does(black\ move\ b8\ a7) = 100\%$; and
- Disc. and PoD, $does(black\ move\ b8\ a7) = 100\%$.

4.3 GGP Games

We use 9 two-player games drawn from games commonly played in GGP competitions, each played on a 5×5 board.

Breakthrough and Knightthrough are racing games where each player is trying to get one their piece across the board, these two games are popular as benchmarks in the GGP community.

Domineering and NoGo are mathematical games in which players gradually fill a board until one of them has no legal moves remaining and is declared loser, these two games are popular in the Combinatorial Game Theory community.

For each of these domain, we construct a *misere* version, which has exactly the same rules but with reverse winning condition. For instance, a player wins misere Breakthrough if they force their opponent to cross the board. To this list, we add AtariGo, a capturing game in which each player tries to surround the opponent's pieces.

AtariGo is popular as a pedagogical tool when teaching the game of Go.

4.4 Performance of the playout engine

It is well known in the games community that increasing the strength of a playout policy may not always result in a strength increase for the wrapping search [Silver and Tesauro, 2009]. Still, it often is the case in practice, and determining whether some of our heuristics improve the strength of the corresponding policy may prove informative. It is also important to know how fast playouts can be performed as it will directly influence how large the MCTS can grow in a given time budget.

Table 3 addresses these two concerns. For each of the nine games of interest, we are provided with the raw speed of the engine in thousands of playouts per second, as well as the scores of a discounted nested player against a nested player without discounting for different levels of nesting. 500 games were run per match, 250 with each color.

	k.Playouts	Nesting Level		
	per second	0	1	2
Breakthrough	1641	79.6	99.6	99.4
MisereBreakthrough	1369	42.4	80.8	90.0
Knightthrough	1632	78.6	100.0	100.0
MisereKnightthrough	1337	46.0	83.2	85.8
Domineering	1406	71.2	77.0	83.8
MisereDomineering	1419	43.4	63.2	68.4
NoGo	397	62.8	76.4	83.4
MisereNoGo	409	53.2	65.6	67.2
AtariGo	123	69.6	97.2	100.0

Table 3: Performance of the playout engine: Implementation speed and win rate of NMC(n) with discounting against NMC(n) without discounting for various nesting level.

4.5 Win Rate Performance

We want to determine whether using nested rather than plain Monte Carlo playouts could improve the performance of a UCT player in two-player games in a more systematic way. We also want to measure the effect of some of the proposed heuristics, *discounting* together with PoD as well as CoW.

We therefore played a parameterised UCT(NMC(n)) player against an optimized standard UCT opponent in a variety of games. Both players are allocated the same thinking time, ranging from 10ms per move to 320ms per move[11]. For each game, each parameter setting, and each time constraint, we run a 500 games match where UCT(NMC(n)) plays as first player 250 times and we record how frequently UCT(NMC(n)) wins. The experiments were run on a 3.0 GHz PC under Linux. The results are presented in Table 4.

The first point to make is that the win/loss ratio must be taken in context. For example 50/50 might mean that each player is playing at optimal performance, or it might mean they are both playing badly.

We can notice in Table 4 that PoD and CoW improve the performance of the search over a basic UCT player. We can

[11] the experimental setup uses hard-coded games, not GDL, so is significantly faster than a GGP.

Game		n	CoW	PoD	10ms	20ms	40ms	80ms	160ms	320ms
Breakthrough		1			3.2	6.0	12.0	11.6	7.8	6.4
		1		✔	27.6	22.6	16.8	21.6	15.4	20.4
		1	✔		22.6	25.2	30.4	34.6	35.2	39.6
		2	✔		4.6	2.0	2.4	1.4	2.4	3.8
	misere	1			85.4	83.4	70.2	60.8	57.0	56.4
	misere	1		✔	91.4	95.6	97.0	97.8	98.8	98.8
	misere	1	✔		95.2	95.2	98.0	99.0	99.8	99.8
	misere	2	✔		1.0	27.6	43.6	87.0	93.2	95.6
Knightthrough		1			42.2	57.2	9.8	49.4	50.2	50.0
		1		✔	68.6	50.2	42.4	42.4	46.4	44.6
		1	✔		27.2	25.4	28.0	43.4	49.2	49.6
		2	✔		20.0	16.4	5.8	1.8	29.2	38.2
	misere	1			43.0	31.6	20.0	15.4	11.2	12.6
	misere	1		✔	54.6	72.2	80.6	88.4	94.2	98.4
	misere	1	✔		77.8	82.2	88.8	94.4	98.2	98.6
	misere	2	✔		20.8	18.6	32.2	42.2	54.0	67.0
Domineering		1			13.4	8.6	8.6	6.0	14.2	28.0
		1		✔	40.8	34.4	37.4	48.4	50.0	50.0
		1	✔		44.4	38.6	40.6	49.4	50.0	50.0
		2	✔		11.2	14.4	20.2	25.2	32.2	45.4
	misere	1			33.4	25.2	20.0	18.8	13.2	12.2
	misere	1		✔	45.4	47.2	56.8	60.2	62.8	54.2
	misere	1	✔		69.4	66.6	71.6	70.4	68.4	58.6
	misere	2	✔		37.0	45.2	45.6	51.0	57.8	53.6
NoGo		1			5.8	3.0	2.6	3.0	0.6	0.8
		1		✔	7.2	16.0	31.8	35.2	35.4	40.6
		1	✔		37.6	39.2	38.4	40.8	47.8	48.0
		2	✔		0.4	2.8	5.4	15.0	20.6	17.0
	misere	1			14.6	6.6	5.2	3.0	2.4	1.8
	misere	1		✔	17.2	25.0	38.8	51.2	48.2	48.8
	misere	1	✔		55.4	56.6	57.0	57.6	54.6	60.8
	misere	2	✔		5.2	10.6	19.4	35.6	37.2	47.8
Atari-Go		1			0.6	2.2	4.6	5.4	6.8	7.6
		1		✔	0.2	19.2	42.0	42.0	55.4	67.2
		1	✔		42.0	59.0	60.2	71.0	71.2	77.2
		2	✔		0.2	0.0	0.6	7.4	8.6	4.8

Table 4: Win percentages for a UCT(NMC(n)) player against an optimized UCT player for various settings and thinking times: "PoD" stands for *Discounting+Pruning on Depth* and "CoW" for *Cut on Win*.

also observe that for this range of computational resources, using a nested search with CoW at both levels does not seem as effective as CoW for level 1 only.

Where the statistic move towards the 50% mark we can infer that games are being played perfectly by both the reference UCT player, and by the Nested Player. Domineering, especially, appears to be an easy task for the algorithms at hand, and Knightthrough might be quite close to solved. On the other hand, the large proportion of games lost by the reference UCT player independent of the side played demonstrate that some games are far from being solved by this algorithm, for instance misere Breakthrough, misere Knightthrough, or Atari-Go are dominated by UCT. This shows that although we have used 9 games all played on boards of the same 5×5 size, the underlying decision problems were of varying difficulty.

The performance improvement on the misere version of some the games seems to be much larger than on the original versions. A tentative explanation for this phenomenon which would be consistent with a similar intuition in single-agent domains is that Nested Player is particularly good at games where the very last moves are crucial to the final score. Since the last moves made in a level n playout are based on a search of an important fraction of the subtree, comparatively fewer mistakes are made at this stage of the game than a plain Monte Carlo playout. Therefore, the estimates of a position's value are particularly more accurate for the Nested Playout than for plain Monte Carlo Playout. This is consistent with the idea[12] that the Nested Playout is performing a Depth Limited MiniMax search at the terminus of the playout.

5 Conclusion

In this paper, we have examined the adaptation of the NMCS algorithm from single-agent problems to two-outcome two-player games.

We have proposed three types heuristic improvements to the algorithm and have shown that these suggestions indeed lead to better performance than that of the naive adaptation. In particular, discounting the reward based on the playout length increases the accuracy of the nested searches, and the various pruning strategies allow to discard very large parts of the search trees.

Together these ideas contribute to creating a new type of domain agnostic search-based artificial player which appears to be much better than a classic UCT player on some games. In particular, in the games misere Breakthrough and misere Knightthrough the new approach wins close to 99% of the games against the best known domain independent algorithm for these games.

Some important improvements to the original single-player NMCS algorithm such as *memorization of the best sequence* [Cazenave, 2009] cannot be adapted to the two-player setting because of the alternation between maximizing and minimizing steps. Still, nothing prevents attempting to generalize some of the other heuristics such as All-Moves-As-First idea [Akiyama *et al.*, 2010] and the Nested Rollout Policy Adaptation [Rosin, 2011] in future work. Future work could also examine how to further generalize NMCS to multi-outcome games.

While we built our Nested Player around a purely random policy as is most common in the GGP community, our technique could also build on the alternative domain-specific pseudo-random policies developed in the Computer Go community [Silver and Tesauro, 2009; Browne *et al.*, 2012]. The interplay between such smart elementary playouts and our nesting construction and heuristics could provide a fruitful avenue for an experimentally oriented study.

[12] in section 3.4

6 Acknowledgments

This research was supported by the Australian Research Council under grant no. DP120102023. The last author is also affiliated with the University of Western Sydney.

This work was granted access to the HPC resources of MesoPSL financed by the Region Ile de France and the project Equip@Meso (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche.

References

[Akiyama *et al.*, 2010] Haruhiko Akiyama, Kanako Komiya, and Yoshiyuki Kotani. Nested monte-carlo search with amaf heuristic. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 172–176. IEEE, 2010.

[Baier and Winands, 2013] Hendrik Baier and Mark HM Winands. Monte-carlo tree search and minimax hybrids. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[Browne *et al.*, 2012] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[Cazenave and Saffidine, 2010] Tristan Cazenave and Abdallah Saffidine. Score bounded Monte-Carlo tree search. volume 6515 of *Lecture Notes in Computer Science*, pages 93–104, Kanazawa, Japan, September 2010. Springer.

[Cazenave, 2009] Tristan Cazenave. Nested Monte-Carlo search. pages 456–461, Pasadena, California, USA, July 2009. AAAI Press.

[Cazenave, 2010] Tristan Cazenave. Nested Monte-Carlo expression discovery. pages 1057–1058, Lisbon, Portugal, August 2010. IOS Press.

[Chaslot *et al.*, 2009] Guillaume M.J.-B. Chaslot, Jean-Baptiste Hoock, Julien Perez, Arpad Rimmel, Olivier Teytaud, and Mark H.M. Winands. Meta Monte-Carlo Tree Search for automatic opening book generation. pages 7–12, Pasadena, California, USA, July 2009.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, volume 8, pages 259–264, 2008.

[Furtak and Buro, 2013] Timothy Furtak and Michael Buro. Recursive Monte Carlo search for imperfect information games. pages 225–232, Niagara Falls, Canada, August 2013. IEEE Press.

[Genesereth *et al.*, 2005] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[Gudmundsson and Björnsson, 2013] Stefan Freyr Gudmundsson and Yngvi Björnsson. Sufficiency-based selection strategy for MCTS. pages 559–565, Beijing, China, August 2013. AAAI Press.

[Knuth and Moore, 1975] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[Méhat and Cazenave, 2010] Jean Méhat and Tristan Cazenave. Combining UCT and nested Monte-Carlo search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277, 2010.

[Pepels *et al.*, 2014] Tom Pepels, Mandy J.W. Tak, Marc Lanctot, and Mark H.M. Winands. Quality-based rewards for Monte-Carlo Tree Search simulations. volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 705–710, Prague, Czech Republic, August 2014. IOS Press.

[Ramanujan *et al.*, 2010] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On adversarial search spaces and sampling-based planning. In *ICAPS*, volume 10, pages 242–245, 2010.

[Rosin, 2011] Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. In Walsh [2011], pages 649–654.

[Saffidine *et al.*, 2011] Abdallah Saffidine, Nicolas Jouandeau, and Tristan Cazenave. Solving Breakthough with race patterns and Job-Level Proof Number Search. volume 7168 of *Lecture Notes in Computer Science*, pages 196–207, Tilburg, The Netherlands, November 2011. Springer.

[Schofield and Thielscher, 2015] Michael Schofield and Michael Thielscher. Lifting model sampling for general game playing to incomplete-information models. In *Proc. AAAI*, Austin Texas, January 2015.

[Silver and Tesauro, 2009] David Silver and Gerald Tesauro. Monte-Carlo simulation balancing. pages 945–952, Montreal, Quebec, Canada, June 2009. ACM.

[Smith and Nau, 1994] Stephen J.J. Smith and Dana S. Nau. An analysis of forward pruning. pages 1386–1391, Seattle, WA, USA, July 1994. AAAI Press.

[Steinhauer, 2010] Janik Steinhauer. Monte-Carlo twixt. Master's thesis, Maastricht University, Maastricht, The Netherlands, June 2010.

[Thielscher, 2011] Michael Thielscher. The general game playing description language is universal. In Walsh [2011], pages 1107–1112.

[Walsh, 2011] Toby Walsh, editor. *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, Barcelona, Catalonia, Spain, July 2011. AAAI Press.