ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-708

Multiple-Outcome Proof Number Search

Abdallah Saffidine[1] and **Tristan Cazenave**[1]

Abstract. We present Multiple-Outcome Proof Number Search (MOPNS), a Proof Number based algorithm able to prove positions in games with multiple outcomes. MOPNS is a direct generalization of Proof Number Search (PNS) in the sense that both behave exactly the same way in games with two outcomes. However, MOPNS targets a wider class of games. When a game features more than two outcomes, PNS can be used multiple times with different objectives to finally deduce the value of a position. On the contrary, MOPNS is called only once to produce the same information. We present experimental results on solving various positions of the games CONNECT FOUR and WOODPUSH showing that in most problems, the total number of node creations of MOPNS is lower than the cumulative number of node creations of PNS, even in the best case where PNS does not need to perform a binary search.

1 INTRODUCTION

Proof Number Search (PNS) [4] is a best first search algorithm that enables to dynamically focus the search on the parts of the search tree that seem to be easier to solve. While PNS is primarily applicable to games with two outcomes, win and loss, it can also solve games with more than two outcomes using a binary search and thresholds on the outcomes. PNS based algorithms have been successfully used in many games and especially as a solver for difficult games such as CHECKERS [17], SHOGI [19], and GO [7].

In this paper, we propose a new *effort number* based algorithm that enables to solve games with multiple outcomes. The principle guiding our algorithm is to use the same tree for all possible outcomes. When using a dichotomic PNS, the search trees are independent of each other and the same subtrees are expanded again. We avoid this re-expansion sharing the common nodes. Moreover we can safely prune some nodes using considerations on bounds as in Score Bounded Monte Carlo Tree Search (MCTS) [6].

There has been a lot of developments of the original PNS algorithm [4]. An important problem related to PNS is memory consumption as the tree has to be kept in memory. In order to alleviate this problem, V. Allis proposed PN^2 [3]. It consists in using a secondary PNS at the leaves of the principal PNS. It allows to have much more information than the original PNS for equivalent memory, but costs more computation time. PN^2 has recently been used to solve FANORONA [15].

The main alternative to PN^2 is the Depth-First Proof Number Search (DFPN) algorithm [11]. DFPN is a depth-first variant of PNS based on the iterative deepening idea. DFPN will explore the game tree in the same order as PNS with a lower memory footprint but at the cost of re-expanding some nodes.

[1] LAMSADE, Université Paris-Dauphine, France. abdallah.saffidine@dauphine.fr, cazenave@lamsade.dauphine.fr

Conspiracy numbers search [8, 16] also deals with a range of possible evaluations at the leaves of the search tree. However, the algorithm works with a heuristic evaluation function whereas Multiple-Outcome Proof Number Search (MOPNS) has no evaluation function and only scores solved positions. Moreover the development of the tree is not the same for MOPNS and for Conspiracy numbers search since MOPNS tries to prove the outcome that costs the less effort whereas Conspiracy numbers search tries to eliminate unlikely values of the evaluation function.

The Iterative PNS algorithm [9] also deals with multiple outcomes but uses the usual proof and disproof numbers as well as a value for each node and a cache. The main difference between Iterative PNS and the proposed MOPNS, is that Iterative PNS tries to find the value of the game by eliminating outcomes step by step. On the other hand, MOPNS can dynamically focus on newly promising values even if previously promising values have not been completly outruled yet.

The next section gives some definitions that will be used in the remainder of the paper. The third section details PNS. The fourth section explains MOPNS. The fifth section gives experimental results for the games CONNECT FOUR and WOODPUSH.

2 DEFINITIONS

We consider a two player game. The players are named *Max* and *Min*. $\mathbb{O} = \{o_1, \ldots, o_m\}$ denotes the possible outcomes of the game. We assume that the outcomes are linearly ordered with the following preference relation for *Max*: $o_1 <_{Max} \cdots <_{Max} o_m$, we further assume that the game is zero-sum and derive a preference relation for *Min*: $o_m <_{Min} \cdots <_{Min} o_1$. In the following, we will always stand in the point of view from *Max* and use $o_i < o_j$ (resp. $o_i \leq o_j$) as a shorthand for $o_i <_{Max} o_j$ (resp. $\neg o_i <_{Min} o_j$).

We assume the game is finite, acyclic, sequential and deterministic. Each position n is either *terminal* or *internal* and some player is to move. When a position n is internal and player p is to play, we call children of n (noted $\mathrm{chil}(n)$) the positions that can be reached by a move of p. Using backward induction, we can therefore associate to each position n a so-called minimax value, noted $\mathrm{real}(n) \in \mathbb{O}$.

Solving a position consists in obtaining its minimax value. It is possible to compute directly the minimax value of a given position by building the whole subsequent game tree and using straightforwardly the definition of minimax values. This naive procedure, however, is resource intensive and more practical methods can be sought. Indeed, not every part of the subsequent game tree is needed to compute the minimax value of a position. For instance, if we know that *Max* is to play in a position n and one child has the best value possible, then $\mathrm{real}(n)$ does not depend on the value of the other children and they need not be calculated.

As the current game tree is not necessarily completely expanded, the following classification of nodes arises. *Internal nodes* corre-

spond to internal positions and have their children in the tree. *Terminal nodes* correspond to terminal positions and have no children. Non-terminal leaf nodes (*leaves* for short) correspond to internal positions and do not have their children in the tree.

We call *effort numbers* heuristic numbers which try to quantify the amount of information needed to prove some fact about the minimax value of a position. The higher the number, the larger the missing piece of information needed to prove the result. When an effort number reaches 0, then the corresponding fact has been proved to be true, while if it reaches ∞ then the corresponding fact has been proved to be false.

3 PROOF NUMBER SEARCH

PNS is an algorithm that can solve positions without exploring the whole game tree. It is essentially designed for games with two outcomes $\mathbb{O} = \{\mathrm{Lose}, \mathrm{Win}\}$. In the context of PNS, proving that the minimax value of a node is Win is called *proving* the node, while proving that it is Lose is called *disproving* the node.

3.1 Determination of the effort

PNS is a best first search algorithm which tries to minimize the effort needed to solve the root position. Two effort numbers are associated to each node in the tree, the proof number (PN) represents an estimation of the remaining effort needed to prove the node, while the disproof number (DN) represents an estimation of the remaining effort needed to disprove the node. When a node n has been proved, we have $\mathrm{PN}(n) = 0$ and $\mathrm{DN}(n) = \infty$, when n has been disproved, $\mathrm{PN}(n) = \infty$ and $\mathrm{DN}(n) = 0$.

The effort needed to solve a node in the tree is determined in different ways depending on its type. They are summarized in Figure 1. The Win and Lose rows designate terminal nodes, in which case the node is already solved. The Leaf row designates a leaf node. Such a node has not been expanded yet, and the proof and disproof numbers are initially set to 1, although more elaborate initializations exist (see Section 4.6). The *Max* (resp. *Min*) row designates internal nodes where *Max* (resp. *Min*) is to play. For such nodes, the numbers are deduced from the effort numbers of the children nodes.

Node type	PN	DN
Win	0	∞
Lose	∞	0
Leaf	1	1
Max	$\min_{c\in\mathrm{chil}(n)} \mathrm{PN}(c)$	$\sum_{c\in\mathrm{chil}(n)} \mathrm{DN}(c)$
Min	$\sum_{c\in\mathrm{chil}(n)} \mathrm{PN}(c)$	$\min_{c\in\mathrm{chil}(n)} \mathrm{DN}(c)$

Figure 1: Determination of effort numbers for PNS

3.2 Descent and expansion of the tree

If the root node is not solved, then more information needs to be added to the tree. Therefore a (non-terminal) leaf needs to be expanded. To select it, the tree is recursively descended selecting at each *Max* node the child minimizing the proof number and at each *Min* node the child minimizing the disproof number.

Once the node to be expanded, n, is reached, each of its children are added to the tree. Thus the status of n changes from leaf to internal node and $\mathrm{PN}(n)$ and $\mathrm{DN}(n)$ have to be updated. This update may in turn lead to an update of the proof and disproof numbers of its ancestors.

After the proof and disproof numbers in the tree are updated to be consistent with formulae from Figure 1, another most proving leaf can be expanded. The process continues iteratively with a descent of the tree, its expansion and the consecutive update until the root node is solved.

3.3 Multi-outcome games

Many interesting games have more than two outcomes, for instance CHESS, DRAUGHTS and CONNECT FOUR have three outcomes: $\mathbb{O} = \{\mathrm{Win}, \mathrm{Draw}, \mathrm{Lose}\}$. We describe the game of WOODPUSH in the fifth section. A game of WOODPUSH of size S has $2 \times S \times (S+1)$ possible outcomes. For many games, it is not only interesting to know who is the winner but also what is the exact score of the game.

If there are more than two possible outcomes, the minimax value of the starting position can still be found with PNS by using a dichotomic search [4]. This dichotomic search is actually using PNS on transformed games. The transformed games have exactly the same rules and game tree as the original one but have binary outcomes. If there are m different outcomes, then the dichotomic search will make about $\lg(m)$ calls to PNS.

If the minimax value is already known, e.g., from expert knowledge, but needs to be proved, then two calls to PNS are necessary and sufficient.

4 MULTIPLE-OUTCOME PROOF NUMBER SEARCH

MOPNS aims at applying the ideas from PNS to multi-outcome games. However, contrary to dichotomic PNS and iterative PNS, MOPNS dynamically adapts the search depending on the outcomes and searches the same tree for all the possible outcomes.

MOPNS shares many similarities with PNS. A game tree is kept in memory and it is extended through cycles of descent, expansion and updates. MOPNS also makes use of effort numbers.

In PNS, two effort numbers are associated with every node, whereas in MOPNS, if there are m outcomes, then $2m$ effort numbers are associated with every node. In PNS, only completely solved subtrees can be pruned, while pruning plays a more important role in MOPNS and can be compared to alpha-beta pruning.

4.1 Effort Numbers

MOPNS also uses the concept of effort numbers but different numbers are used here in order to account for the multiple outcomes. Let n be a node in the game tree, and $o \in \mathbb{O}$ an outcome. The *greater number*, $\mathrm{G}(n, o)$, is an estimation of the number of node expansions required to prove that the value of n is greater than or equal to o (from the point of view of *Max*), while conversely the *smaller number*, $\mathrm{S}(n, o)$, is an estimation of the number of node expansions required to prove that the value of n is smaller than or equal to o. If $\mathrm{G}(n, o) = \mathrm{S}(n, o) = 0$ then n is solved and its value is o: $\mathrm{real}(n) = o$.

Figure 2 features an example of effort numbers for a three outcomes game. The effort numbers show that in the position under consideration *Max* can force a draw and it seems unlikely that at that point the *Max* can force a win.[2]

[2] $\mathrm{S}(n, \mathrm{Win}) \neq 0$ means that it was not assumed that the game is finite and it has not been proved yet that *Min* can force the game to end.

Outcome	G	S
Win	500	3
Draw	0	10
Lose	0	∞

Figure 2: Example of effort numbers for a 3 outcome game

4.2 Determination of the effort

The effort numbers of internal nodes are obtained in a very similar fashion to PNS, G is analogous to PN and S is analogous to DN. Every effort number of a leaf is initialized at 1, while the effort numbers of an internal node are calculated with the sum and min formulae as shown in Figure 3a.

If n is a terminal node and its value is $\mathrm{real}(n)$, then the effort numbers are associated as shown in Figure 3b. We have for all $o \leq \mathrm{real}(n)$, $\mathrm{G}(n,o) = 0$ and for all $o \geq \mathrm{real}(n)$, $\mathrm{S}(n,o) = 0$.

Node type	$\mathrm{G}(n,o)$	$\mathrm{S}(n,o)$
Leaf	1	1
Max	$\min_{c\in\mathrm{chil}(n)} \mathrm{G}(c,o)$	$\sum_{c\in\mathrm{chil}(n)} \mathrm{S}(c,o)$
Min	$\sum_{c\in\mathrm{chil}(n)} \mathrm{G}(c,o)$	$\min_{c\in\mathrm{chil}(n)} \mathrm{S}(c,o)$

(a) Internal node

Outcome	G	S
o_m	∞	0
...	∞	0
$\mathrm{real}(n)$	0	0
...	0	∞
o_1	0	∞

(b) Terminal node

Figure 3: Determination of effort numbers for MOPNS

4.3 Properties

$\mathrm{G}(n,o) = 0$ (resp. $\mathrm{S}(n,o) = 0$) means that the value of n has been proved to be greater than (resp. smaller) or equal to o, i.e., *Max* (resp. *Min*) can force the outcome to be at least o (resp. at most o). Conversely $\mathrm{G}(n,o) = \infty$ means that it is impossible to prove that the value of n is greater than or equal to o, i.e., *Max* cannot force the outcome to be greater than or equal to o.

As can be observed in Figure 2, the effort numbers are monotonic in the outcomes. If $o_i \leq o_j$ then $\mathrm{G}(n,o_i) \leq \mathrm{G}(n,o_j)$ and $\mathrm{S}(n,o_i) \geq \mathrm{S}(n,o_j)$. Intuitively, this property states that the better an outcome is, the harder it will be to obtain it or to obtain better.

0 and ∞ are permanent values since when an effort number reached 0 or ∞, its value will not change as the tree grows and more information is available. Several properties link the permanent values of a given node. The proofs are straightforward recursions from the leaves and are omitted for lack of space. Care must only be taken that the initialization of leaves satisfies the property which is the case for all the initializations discussed here.

Proposition 1. *If* $\mathrm{G}(n,o) = 0$ *then for all* $o' < o$, $\mathrm{S}(n,o') = \infty$ *and similarly if* $\mathrm{S}(n,o) = 0$ *then for all* $o' > o$, $\mathrm{G}(n,o') = \infty$.

Proposition 2. *If* $\mathrm{G}(n,o) = \infty$ *then* $\mathrm{S}(n,o) = 0$ *and similarly if* $\mathrm{S}(n,o) = \infty$ *then* $\mathrm{G}(n,o) = 0$.

4.4 Descent policy

We call *attracting outcome* of a node n, the outcome $o^*(n)$ that has not been proved to be achievable by the player on turn and minimizing the sum of the corresponding effort numbers. We have for *Max* nodes $o^*(n) = \arg\min_{o,\mathrm{G}(n,o)>0}(\mathrm{G}(n,o)+\mathrm{S}(n,o))$. Similarly, we have for *Min* nodes $o^*(n) = \arg\min_{o,\mathrm{S}(n,o)>0}(\mathrm{G}(n,o)+\mathrm{S}(n,o))$. As a consequence of the existence of a minimax value for each position, for all node n, there always exists at least one outcome o, such that $\mathrm{G}(n,o) \neq \infty$ and $\mathrm{S}(n,o) \neq \infty$. Hence, $\mathrm{G}(n,o^*(n)) + \mathrm{S}(n,o^*(n)) \neq \infty$.

We call *distracting outcome* of a *Max* (resp. *Min*) node n the outcome just below (resp. above) its attracting outcome, we note it $o'(n)$. When the attracting outcome of a *Max* (resp. *Min*) node is the worst (resp. best) outcome in the game, we set the distracting outcome to be equal to the most likely outcome. That is, if n is a *Max* node with $o^*(n) = o_k$, then $o'(n) = o_{\max(k-1,1)}$ and if n is a *Min* node, then $o'(n) = o_{\min(k+1,m)}$. As the name indicates, the distracting outcome of a node is the one towards which it would be simplest for the opponent to deviate if he or she wanted to disprove the attracting outcome.

Consider Figure 2, if these effort numbers were associated to a *Max* node, then the attracting outcome would be Win and the distracting outcome would be Draw, while if they were associated to a *Min* node then the attracting outcome would be Draw and the distracting outcome would be Win.

From now on, unless we specify otherwise, we will only consider the attracting and distracting outcomes of the root node r of the tree and note $o^* = o^*(r), o' = o'(r)$. We assume *Max* is at turn in the root node. We can now define the *root descent policy* that specify how the leaf to be expanded is selected (see Algorithm 1). We first estimate which outcome is attracting at the root node, then we try to prove this value at *Max* nodes and to disprove it at *Min* nodes.

Algorithm 1 Root descent policy

```
argument root Max node r
compute o* and o'
n ← root.
while n is not a leaf do
  if n is a Max node then
    n ← arg min_{c∈chil(n)} G(c, o*)
  else
    n ← arg min_{c∈chil(n)} S(c, o')
  end if
end while
return n
```

Proposition 3. *For finite two outcome games,* MOPNS *and* PNS *develop the same tree.*

Proof. If we know the game is finite, the *Max* is sure to obtain at least the worst outcome so we can initialize the greater number for the worst outcome to 0, we can also initialize the smaller number for the best outcome to 0. If there are two outcomes only, $\mathbb{O} = \{\mathrm{Lose}, \mathrm{Win}\}$, then we have the following relation between effort numbers in PNS and MOPNS: $\mathrm{G}(n,\mathrm{Win}) = \mathrm{PN}(n)$, $\mathrm{S}(n,\mathrm{Lose}) = \mathrm{DN}(n)$. If the game is finite with two outcomes, then the attracting outcome of the root is Win and the distracting outcome is Lose. Hence, MOPNS and PNS behave in the same manner. □

4.5 Pruning

We define the pessimistic and optimistic bounds for a node n as $\text{pess}(n) = \arg\max_o(\text{G}(n,o) = 0)$ and $\text{opti}(n) = \arg\min_o(\text{S}(n,o) = 0)$. The following inequality gives their name to the bounds $\text{pess}(n) \leq \text{real}(n) \leq \text{opti}(n)$, $\text{pess}(n)$ (resp. $\text{opti}(n)$) is the worst value possible (resp. the best value possible) for n consistent with the current information in the tree. For any node n, n is solved as soon as $\text{pess}(n) = \text{opti}(n)$. Although the definition is different, these bounds coincide with those described in Score Bounded Monte Carlo Tree Search [6].

We also define *relevancy bounds* that are similar to alpha and beta bounds in the classic Alpha-Beta algorithm [13]. For a node n, the *lower* relevancy bound is noted $\alpha(n)$ and the *upper* relevancy bound is noted $\beta(n)$. These bounds are calculated using the optimistic and pessimistic bounds as follows. If n is the root of the tree, then $\alpha(n) = \text{pess}(n)$ and $\beta(n) = \text{opti}(n)$. Otherwise, we use the relevancy bounds of the father node of n: if $n \in \text{chil}(f)$, we set $\alpha(n) = \max_{Max}(\alpha(f), \text{pess}(n))$ and $\beta(n) = \min_{Max}(\beta(f), \text{opti}(n))$.

The relevancy bounds of a node n take their name from the fact that if $\text{real}(n) \leq \alpha(n)$ or if $\text{real}(n) \geq \beta(n)$, then having more information about $\text{real}(n)$ will not contribute to solving the root of the tree. Therefore they enable safe pruning.

Proposition 4. *For each node n, if we have $\beta(n) \leq \alpha(n)$ then the the subtree of n need not be explored any further.*

Subtrees starting at a pruned node can be completely removed from the main memory as they will not be used anymore in the proof. This improvement is crucial as lack of memory is one of the main bottleneck of PNS and MOPNS.

We now show that pruning does not interfere with the root descent policy in the sense that it will not affect the number of descents performed before the root is solved. For this purpose, we prove that the root descent policy does not lead to a node which can be pruned.

Proposition 5. *If r is not solved, then for all nodes n traversed by the root descent policy, $\alpha(n) < o^* \leq \beta(n)$.*

Proof. We first prove the inequality for the root node. If the root position r is not solved, then by definition of the attractive outcome, $o^* > \text{pess}(r) = \alpha(r)$. Using Proposition 1, we know that all outcomes better than the optimistic bound cannot be achieved: $\forall o > \text{opti}(r) = \beta(r)$, $\text{G}(o,r) = \infty$. Since $\text{G}(r,o^*) + \text{S}(r,o^*) \neq \infty$, then $\alpha(r) < o^* \leq \beta(r)$.

For the induction step, suppose n is a *Max* node that satisfies the inequality. We need to show that $c = \arg\min_{c \in \text{chil}(n)} \text{G}(c,o^*)$ also satisfies the inequality. Recall that the pessimistic bounds of n and c satisfy the following order: $\text{pess}(c) \leq \text{pess}(n)$ and obtain the first part of the inequality $\alpha(c) = \alpha(n) < o^*$. From the induction hypothesis, $o^* \leq \beta(n) \leq \text{opti}(n)$, so from Proposition 1 $\text{G}(n,o^*) \neq \infty$, moreover, the selection process ensures that $\text{G}(c,o^*) = \text{G}(n,o^*) \neq \infty$, therefore $\text{G}(c,o^*) \neq \infty$ which using Proposition 2 leads to $o^* \leq \text{opti}(c)$. Thus, $o^* \leq \beta(c)$. The induction step when n is a *Min* node is similar and is omitted. □

4.6 Applicability of classical improvements

Many improvements of PNS are directly applicable to MOPNS. For instance, the *current-node enhancement* presented in [3] takes advantage of the fact that many consecutive descents occur in the same subtree. This optimization allow to obtain a notable speed-up and can be straightforwardly applied to MOPNS.

It is possible to initialize leaves in a more elaborate way than presented in Figure 3a. Most initializations available to PNS can be used with MOPNS, for instance the *mobility initialization* [20] in a *Max* node n consists in setting the initial smaller number to the number of legal moves: $\text{G}(n,o) = 1$, $\text{S}(n,o) = |\text{chil}(n)|$. In a *Min* node, we would have $\text{G}(n,o) = |\text{chil}(n)|$, $\text{S}(n,o) = 1$.

A generalization of PN^2 is also straightforward. If n is a new leaf and d descents have been performed in the main tree, then we run a nested MOPNS independent from the main search starting with n as root. After at most d descents are performed, the nested search is stopped and the effort numbers of the root are used as initialization numbers for n in the main search. We can safely propagate the interest bounds to the nested search to obtain even more pruning.

Similarly, a transformation of MOPNS into a depth-first search is possible as well, adapting the idea of Nagai [11]. Just as in DFPN, only two threshold numbers would be needed during the descent, one threshold would correspond to the greater number for the current attractive outome at the root and one threshold would correspond to the smaller number for the distractive outcome.

Finally, given that MOPNS is very close in spirit to PNS, a careful implementer should not face many problems adapting the various improvements that make DFPN such a successful technique in practice. Let us mention in particular Nagai's garbage collection technique [11], Kishimoto and Müller's solution to the Graph History Interaction problem [7], and Pawlewicz and Lew's $1 + \varepsilon$ trick [12].

5 EXPERIMENTAL RESULTS

To assess the validity of our approach, we implemented a prototype of MOPNS and tested it on two games with multiple outcomes, namely CONNECT FOUR and WOODPUSH. Our prototype does not detect transposition and is implemented via the best first search approach described earlier. As such, we compare it to the original best-first variation of PNS, also without transposition detection. Note that the domain of CONNECT FOUR and WOODPUSH are acyclic, so we do not need to use the advanced techniques presented by Kishimoto and Müller to address the Graph History Interaction problem [7]. Additionally, the positions that constitute our testbed were easy enough that they could be solved by search trees of at most a few million nodes. Thus, the individual search trees for PNS as well as MOPNS could fit in memory without ever pruning potentially useful nodes.

In our implementation, the two algorithms share a generic code for the best first search module and only differ on the initialization, the update, and the selection procedures. The experimental results were obtained running OCaml 3.11.2 under Ubuntu on a laptop with Intel T3400 CPU at 2.2 GHz and 1.8 GiB ofmemory.

For each test position and each possible outcome, we performed one run of the PNS algorithm and recorded the time the number of node creation it needed. We then discarded all but the two runs needed to prove the final result. For instance, if a position in WOODPUSH admitted non-zero integer scores between -5 and $+5$ and its perfect play score was 2, we would run PNS ten times, and finally output the measurements for the run proving that the score is greater or equal to 2 and the measurements for the run disproving that the score is greater or equal to 3. This policy is beneficial to PNS compared to doing a binary search for the outcome.

To compare MOPNS to PNS on a wide range of positions, we created the list of all positions reached after a given number of moves from the starting position of a given size. These positions range from being vastly favourable to *Min* to vastly favourable to *Max*, and from trivial (solved in a few milliseconds) to more involved (each run being

around two to three minutes).

5.1 CONNECT FOUR

CONNECT FOUR is a commercial two-player game where players drop a red or a yellow piece on a 7×6 grid. The first player to align four pieces either horizontally, vertically or diagonally wins the game. The game ends in a draw if the board is filled and neither player has an alignment. The game was solved by James D. Allen and Victor Allis in 1988 [2].

Table 1 presents aggregate data over our experiments on size 4×5 and 5×5. In both cases, we used the positions occuring after 4 moves. In the first case, 16 positions among the 256 positions tested were a first player win, 222 were a draw while 18 were a first player loss. In the second list of positions, there were 334 wins, 267 draws, and 24 losses.

Table 1: Cumulated time and number of node creation for the MOPNS and PNS algorithms in the game of CONNECT FOUR. For both algorithm, *Lowest time* indicates the number of positions that were soved faster by this algorithm, while *Lowest node creations* indicates the number of positions which needed fewer node creations.

		MOPNS	PNS
Size 4×5, 256 positions after 4 moves	Total time (seconds)	99	85
	Total node creations	16,947,536	20,175,238
	Lowest time	21	235
	Lowest node creations	227	13
Size 5×5, 625 positions after 4 moves	Total time (seconds)	11,230	9055
	Total node creations	1,557,490,694	1,757,370,222
	Lowest time	55	570
	Lowest node creations	406	140

Figure 4 plots the number of node creations needed to solve each of the 256 4×5 positions. We can see that for a majority of positions, MOPNS needed fewer node creations than PNS. There are 16 positions that needed the same number of node creations by both algorithm and these positions are exactly the positions that are first player wins.

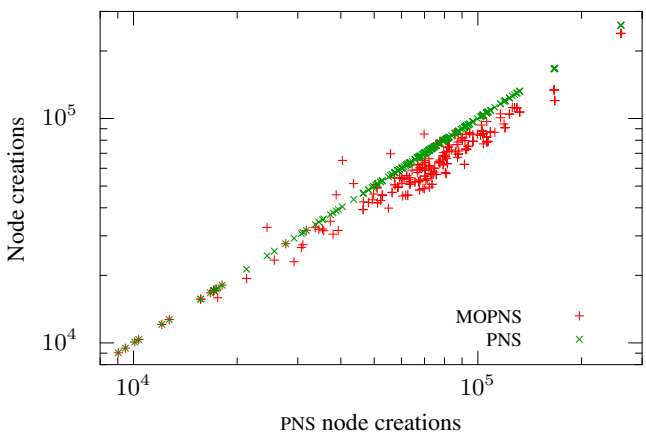

Figure 4: Comparison of the number of node creations for MOPNS and PNS for solving 256 CONNECT FOUR positions on size 4×5.

5.2 WOODPUSH

The game of WOODPUSH is a recent game invented by combinatorial game theorists to analyze a game that involves forbidden repetition of the same position [1, 5]. A starting position consists of some pieces for the left player and some for the right player put on an array of predefined length as shown in Figure 5. A Left move consists in sliding one of the left pieces to the right. If some pieces are on the way of the sliding piece, they are jumped over. When a piece has an opponent piece behind it, it can move backward and push all the pieces behind, provided it does not repeat the previous position. The game is won when the opponent has no more pieces on the board. The score of a game is the number of moves that the winner can play before the board is completely empty.

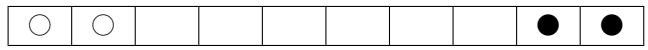

Figure 5: WOODPUSH starting position on size $(10, 2)$

The experimental protocol for WOODPUSH was similar to that of CONNECT FOUR. The first list of problems corresponds to positions occuring after 4 moves on a board of length 8 with 3 pieces for each player. The second list of problems corresponds to positions occuring after 8 moves on a board of length 13 with 2 pieces for each player. Table 2 presents aggregates data for the solving time and the number of node creations, while Figure 6 presents the number of node creations for each problem in the second list.

Table 2: Cumulated time and number of node creation for the MOPNS and PNS algorithms in the game of WOODPUSH.

		MOPNS	PNS
Size $(8, 3)$, 99 positions after 4 moves	Total time (seconds)	718	702
	Total node creations	31,328,178	34,869,213
	Lowest time	25	74
	Lowest node creations	76	23
Size $(13, 2)$, 256 positions after 8 moves	Total time (seconds)	4796	4573
	Total node creations	155,756,022	174,285,199
	Lowest time	98	158
	Lowest node creations	205	51

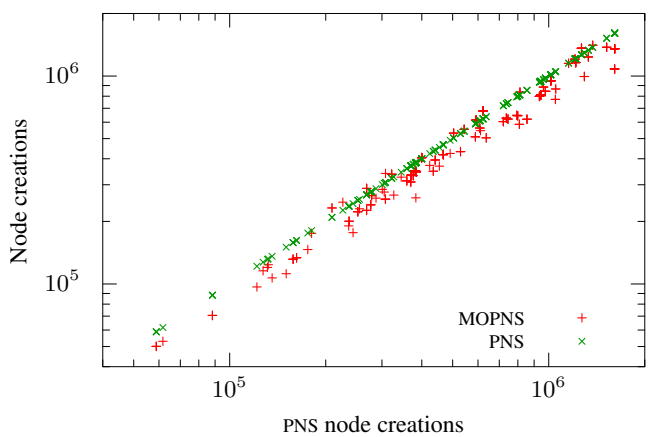

Figure 6: Comparison of the number of node creations for MOPNS and PNS for solving 256 WOODPUSH positions on size $(13, 2)$.

In WOODPUSH $(8, 3)$, it is possible to create final positions with scores ranging from -18 to 18 but these positions might not be accessible from the start position. Indeed, in our experiments, no final position with a score below -5 or over 5 was ever reached. However, while the scores remained between -5 and 5, the exact range varied depending on the problem. While doing a binary search for the

Table 3: Detailed results for the 86th WOODPUSH problem on size $(8, 3)$.

	PNS									Dichotomic PNS		MOPNS	
Setting	≥ -4	≥ -3	≥ -2	≥ -1	≥ 1	≥ 2	≥ 3	≥ 4	≥ 5	[-5, 5]	[1, 3]	[-5, 5]	[1, 3]
Time	0.508	0.500	0.884	1.188	1.200	1.204	3.084	1.360	1.356	6.676	4.288	4.556	3.684
Nodes	39340	39340	68035	84184	84568	84545	178841	98069	98069	351366	263386	210183	191127
Result	true	true	true	true	true	true	false	false	false	2	2	2	2

outcome is the natural generic process for solving a multi-outcome game with PNS, we decided to compare MOPNS to the ideal case for PNS which only involves two runs per position. On the other hand, we only assumed for MOPNS that the outcome was in $[-5, 5]$. Therefore, the results presented in Table 2 and Figure 6 significantly favour PNS.

Table 3 details the results for the position presented in Figure 7. The PNS tree did not access any position with a score lower or equal to -4 nor any position with a score greater or equal to 5.

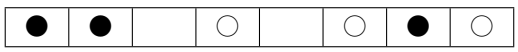

Figure 7: 86th WOODPUSH problem on size $(8, 3)$.

6 CONCLUSION AND DISCUSSION

We have presented a generalized Proof Number algorithm that solves games with multiple outcomes in one run. Running PNS multiple times to prove an outcome develops the same nodes multiple times while in MOPNS these nodes are developed only once. MOPNS has been formally proved equivalent to PNS in two-outcome games and we have shown how safe pruning could be performed in multiple outcome games. For small CONNECT FOUR and WOODPUSH boards, in most cases MOPNS solves the games with fewer node creations than PNS even if it already knows the optimal outcome of the game and no binary search is needed.

We have assumed in this article that the game structure was a tree. In most practical cases it actually is a Directed Acyclic Graph (DAG) and in some cases the graph contains cycles.[3] The theoretical results presented in this article still hold in the DAG case, provided the definition of the relevancy bounds is adapted to reflect the fact that a node may have multiple parents and some of them might not yet be in the tree. The double count problem of PNS will also affect MOPNS in DAGs, but it is possible to take advantage of previous work on the handling of transpositions in PNS [18, 10]. Similarly, the problems encountered by MOPNS in cyclic graphs are similar to that of PNS and DFPN in cyclic graphs. Fortunately, it should be straightforward to adapt Kishimoto and Müller's ideas [7] from DFPN to a depth-first version of MOPNS.

In future work, we plan on trying to adapt the PN^2 paralellization scheme suggested by Saffidine et al. [14] to games with multiple outcomes via MOPNS. We would also like to study a depth-first version of MOPNS that can be obtained via Nagai's transformation [11].

Finally, studying how MOPNS can be extended to deal with problems where the outcome space is not known beforehand or is continuous in order to develop an effort number algorithm for non-deterministic two-player games is definitely an attractive research agenda.

[3] For instance, the original rules for CHESS result in a DAG because of the 50-moves rule, but this rule is usually abstracted away, resulting in a cyclic structure.

REFERENCES

[1] Michael H. Albert, Richard J. Nowakowski, and David Wolfe, *Lessons in play: an introduction to combinatorial game theory*, AK Peters Ltd, 2007.

[2] Louis Victor Allis, *A Knowledge-based Approach of Connect-Four The Game is Solved: White Wins*, Masters thesis, Vrije Universitat Amsterdam, Amsterdam, The Netherlands, October 1988.

[3] Louis Victor Allis, *Searching for Solutions in Games an Artificial Intelligence*, Phd thesis, Vrije Universitat Amsterdam, Department of Computer Science, Rijksuniversiteit Limburg, 1994.

[4] Louis Victor Allis, M. van der Meulen, and H. Jaap van den Herik, 'Proof-Number Search', *Artificial Intelligence*, **66**(1), 91–124, (1994).

[5] Tristan Cazenave and Richard J. Nowakowski, 'Retrograde analysis of woodpush', in *Games of no chance 4*, to appear.

[6] Tristan Cazenave and Abdallah Saffidine, 'Score bounded Monte-Carlo tree search', in *Computers and Games*, eds., H. van den Herik, Hiroyuki Iida, and Aske Plaat, volume 6515 of *Lecture Notes in Computer Science*, 93–104, Springer-Verlag, Berlin / Heidelberg, (2011).

[7] Akihiro Kishimoto and Martin Müller, 'A solution to the GHI problem for depth-first proof-number search', *Information Sciences*, **175**(4), 296–314, (2005).

[8] David A. McAllester, 'Conspiracy numbers for min-max search', *Artificial Intelligence*, **35**(3), 287–310, (1988).

[9] Carsten Moldenhauer, *Game tree search algorithms for the game of cops and robber*, Master's thesis, University of Alberta, September 2009.

[10] Martin Müller, 'Proof-set search', in *Computers and Games 2002*, Lecture Notes in Computer Science, 88–107, Springer, (2003).

[11] Ayumu Nagai, *Df-pn algorithm for searching AND/OR trees and its applications*, Ph.D. dissertation, University of Tokyo, December 2001.

[12] Jakub Pawlewicz and Łukacz Lew, 'Improving depth-first PN-search: $1+\varepsilon$ trick', in *Proceedings of the 5th international conference on Computers and games*, pp. 160–171. Springer-Verlag, (2006).

[13] Stuart J. Russell and Peter Norvig, *Artificial Intelligence — A Modern Approach*, Pearson Education, third edn., 2010.

[14] Abdallah Saffidine, Nicolas Jouandeau, and Tristan Cazenave, 'Solving Breakthough with race patterns and Job-Level Proof Number Search', in *Advances in Computer Games*, Springer-Verlag, Berlin / Heidelberg, (2011).

[15] Maarten P.D. Schadd, Mark H.M. Winands, Jos W.H.M. Uiterwijk, H. Jaap van den Herik, and M.H.J. Bergsma, 'Best Play in Fanorona leads to Draw', *New Mathematics and Natural Computation*, **4**(3), 369–387, (2008).

[16] Jonathan Schaeffer, 'Conspiracy numbers', *Artificial Intelligence*, **43**(1), 67–84, (1990).

[17] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen, 'Checkers is solved', *Science*, **317**(5844), 1518, (2007).

[18] Martin Schijf, L. Victor Allis, and Jos W.H.M. Uiterwijk, 'Proof-number search and transpositions', *ICCA Journal*, **17**(2), 63–74, (1994).

[19] Masahiro Seo, Hiroyuki Iida, and Jos W.H.M. Uiterwijk, 'The PN*-search algorithm: Application to tsume-shogi', *Artificial Intelligence*, **129**(1-2), 253–277, (2001).

[20] H. Jaap van den Herik and Mark Winands, 'Proof-Number Search and its variants', *Oppositional Concepts in Computational Intelligence*, 91–118, (2008).