Generalized Proof Number Search

Abdallah Saffidine **Tristan Cazenave**
LAMSADE
Université Paris-Dauphine
Paris, France

Abstract

We present Generalized Proof Number Search (GPNS), a Proof Number based algorithm able to prove positions in games with multiple outcomes. GPNS 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, GPNS 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, GPNS is called only once to produce the same information. We present experimental results on solving various sizes of the games Connect Four and Woodpush showing that the total number of tree descents of GPNS is much lower than the cumulative number of tree descents of PNS.

1 Introduction

Proof Number Search (PNS) [Allis *et al.*, 1994] is a best first search algorithm that enables to dynamically focus the search on the parts of the search tree that are easier to solve. PNS solves games with two outcomes, either a win or a loss. It can solve games with more than two outcomes using a dichotomic search and thresholds on the outcomes. It has been successfully used for solving difficult games such as Fanorona that has been proven a draw [Schadd *et al.*, 2008].

1.1 Motivation

In this paper we propose a new PNS 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 multiple times. We avoid this re-expansion sharing the common nodes. Moreover we can safely prune some nodes using considerations on bounds as in [Cazenave and Saffidine, 2010].

1.2 Previous work

There has been a lot of developments of the original PNS algorithm [Allis *et al.*, 1994]. 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 [Allis, 1994]. 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. The other improvements of PNS are depth first algorithms that behave similarly to PNS. PN* [Seo, 1995] is the first depth first iterative deepening version of PNS that uses thresholds at AND nodes and a transposition table. It was extended to PDS [Nagai, 1999] to deal with disproof numbers and later to df-pn [Nagai, 2002].

PNS algorithms have been successfully used in many games and especially as solvers for games such as Checkers [Schaeffer, 2007; Schaeffer *et al.*, 2007], Shogi [Seo *et al.*, 2001] and Go [Kishimoto and Müller, 2005].

Conspiracy numbers search [McAllester, 1988; Schaeffer, 1990] 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 GPNS has no evaluation function and only scores solved positions. Moreover the development of the tree is not the same for GPNS and for Conspiracy numbers search since GPNS 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 [Moldenhauer, 2009] also deals with multiple outcomes but uses the usual proof and disproof numbers as well as a value for each node and a cache.

1.3 Outline of the paper

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 GPNS. 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 $\text{chil}(n)$) 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 $\text{real}(n) \in \mathbb{O}$.

Solving a position consists in obtaining its minimax value. It is possible to compute directly the minimax values 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 node. For instance, if we know that *Max* is to play in a position n and one child has the best value possible, then $\text{real}(n)$ 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. An internal node is called *developed* if its children have been added to the current game tree. Nodes that are not developed are called *frontier* nodes.

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 $\mathbb{O} = \{\text{Lose}, \text{Win}\}$. 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 $\text{PN}(n) = 0$ and $\text{DN}(n) = \infty$, when n has been disproved, $\text{PN}(n) = \infty$ and $\text{DN}(n) = 0$.

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 Frontier row designates an internal frontier node that has not been expanded yet, for which the proof and disproof numbers are

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

Figure 1: Determination of effort numbers for PNS

initially set to 1, although more elaborate initializations exist (see section 4.6). The *Max* (resp. *Min*) row designates developed nodes where *Max* (resp. *Min*) is to play. For such nodes, the numbers are deduced from the effort numbers of the children nodes.

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 an internal frontier node 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 a frontier node to a developed node and $\text{PN}(n)$ and $\text{DN}(n)$ 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 frontier node 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: $\mathbb{O} = \{\text{Win}, \text{Draw}, \text{Lose}\}$. We describe the game of Woodpush in the fifth section. A game of Woodpush of size S has $S \times (S + 1)$ 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 using a dichotomic search [Allis *et al.*, 1994]. 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 Generalized Proof Number Search

GPNS aims at applying the ideas from PNS to multi-outcome games. However, contrary to dichotomic PNS and iterative PNS, GPNS dynamically adapts the search depending on the

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

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

outcomes and searches the same tree for all the possible outcomes.

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

In PNS, two effort numbers are associated with every node, whereas in GPNS, 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 GPNS and can be compared to alpha-beta pruning.

4.1 Effort Numbers

GPNS 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 *Max* can force a win.[1]

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 frontier node is initialized at 1, while the effort numbers of a developed 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 $o \geq \mathrm{real}(n)$, $\mathrm{S}(n, o) = 0$.

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.

[1] $\mathrm{S}(n, \mathrm{Win}) \neq 0$ 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.

Node type	$\mathrm{G}(n, o)$	$\mathrm{S}(n, o)$
Frontier	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 GPNS

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 internal frontier nodes 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 best opponent try 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 frontier node 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 frontier node 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, GPNS 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 GPNS: $\mathrm{G}(n, \mathrm{Win}) = \mathrm{PN}(n)$, $\mathrm{S}(n, \mathrm{Lose}) = \mathrm{DN}(n)$. If the game is finite with two outcomes, then the attracting outcome of the root is Win and the distracting outcome is Lose. Hence, GPNS and PNS behave in the same manner. □

4.5 Pruning

We define the pessimistic and optimistic bounds for a node n as $\mathrm{pess}(n) = \arg\max_o(\mathrm{G}(n, o) = 0)$ and $\mathrm{opti}(n) = \arg\min_o(\mathrm{S}(n, o) = 0)$. The following inequality gives their name to the bounds $\mathrm{pess}(n) \leq \mathrm{real}(n) \leq \mathrm{opti}(n)$, $\mathrm{pess}(n)$ (resp. $\mathrm{opti}(n)$) 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 $\mathrm{pess}(n) = \mathrm{opti}(n)$. Although the definition is different, these bounds coincide with those described in Score Bounded Monte-Carlo Tree Search [Cazenave and Saffidine, 2010].

We also define *relevancy bounds* that are similar to alpha and beta bounds in the classic Alpha-Beta algorithm [Russell and Norvig, 2002]. 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 $\alpha(n) = \mathrm{pess}(n)$ and $\beta(n) = \mathrm{opti}(n)$. Otherwise, we use the relevancy bounds of the father node of n: if $n \in \mathrm{chil}(f)$, we set $\alpha(n) = \max_{Max}(\alpha(f), \mathrm{pess}(n))$ and $\beta(n) = \min_{Max}(\beta(f), \mathrm{opti}(n))$.

The relevancy bounds of a node n take their name from the fact that if $\mathrm{real}(n) \leq \alpha(n)$ or if $\mathrm{real}(n) \geq \beta(n)$, then having more information about $\mathrm{real}(n)$ 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 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 GPNS.

Compatibility of pruning and the root descent policy

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, $o^* > \mathrm{pess}(r) = \alpha(r)$. Using Proposition 1, we know that all outcomes better than the optimistic bound cannot be achieved: $\forall o > \mathrm{opti}(r) = \beta(r)$, $\mathrm{G}(o, r) = \infty$. Since $\mathrm{G}(r, o^*) + \mathrm{S}(r, o^*) \neq \infty$, 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 $c = \arg\min_{c \in \mathrm{chil}(n)} \mathrm{G}(c, o^*)$ also satisfies the inequality. Recall that the pessimistic bounds of n and c satisfy the following order: $\mathrm{pess}(c) \leq \mathrm{pess}(n)$ and obtain the first part of the inequality $\alpha(c) = \alpha(n) < o^*$. From the induction hypothesis, $o^* \leq \beta(n) \leq \mathrm{opti}(n)$, so from Proposition 1 $\mathrm{G}(n, o^*) \neq \infty$, moreover, the selection process ensures that $\mathrm{G}(c, o^*) = \mathrm{G}(n, o^*) \neq \infty$, therefore $\mathrm{G}(c, o^*) \neq \infty$ which using Proposition 2 leads to $o^* \leq \mathrm{opti}(c)$. Thus, $o^* \leq \beta(c)$. The induction step when n is a *Min* node is similar and is omitted for lack of space. □

4.6 Applicability of improvements of PNS to GPNS

Many improvements of PNS are directly applicable to GPNS. For instance, the *current-node enhancement* presented in [Allis, 1994] 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 GPNS.

	PNS			GPNS
Size	$\leq$ Draw	$\geq$ Draw	Sum	
3×4	1618	673	2291	813
3×5	4799	4903	9702	2498
4×3	11427	10888	22315	2919
3×6	21746	15759	37505	8714
4×4	79601	33393	112994	22691
3×7	150172	95159	245331	28571
5×3	419952	190813	610765	65694
4×5	402603	304862	707465	171222
3×8	750745	493702	1244447	84314
5×4	2220291	1708671	3928962	2234554
3×9	2678172	2992236	5670408	270004

Table 1: Number of descents required for solving various sizes of Connect Four

It is possible to initialize internal frontier nodes in a more elaborate way than presented in Figure 3a. Most initializations available to PNS can be used with GPNS, for instance the *mobility initialization* [van den Herik and Winands, 2008] in a *Max* node n consists in setting the initial smaller number to the number of legal moves: $\mathrm{G}(n, o) = 1$, $\mathrm{S}(n, o) = |\operatorname{chil}(n)|$. In a *Min* node, we would have $\mathrm{G}(n, o) = |\operatorname{chil}(n)|$, $\mathrm{S}(n, o) = 1$.

A generalization of PN^2 is also straightforward. If n is a new internal frontier node and d descents have been performed in the main tree, then we run a nested GPNS 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.

5 Experimental results

In this section we detail the application of GPNS to the games of Connect Four and Woodpush.

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 [Allis, 1988].

Table 1 gives the number of descents for proving the outcomes for various sizes of Connect Four. The first column is the size of the board. The second column is the number of descents of PNS to prove that the games is not won. The third column is the number of descents required by PNS to prove the game is not lost. The fourth column is the sum of these two numbers which is the number of descents required by PNS to prove the game is a draw. The last column gives the number of descents for GPNS. We can see that the number of descents for GPNS are 1.75 times to 21 times smaller than the number of descents required by PNS to prove the draw.

	PNS			GPNS
Size	$\leq$ Draw	$\geq$ Draw	Sum	
3×4	863	615	1478	625
3×5	3571	2227	5798	2092
4×3	3465	3502	6967	2094
3×6	10958	12304	23262	6871
4×4	43085	19965	63050	17260
3×7	59821	45540	105361	22842
5×3	220947	132595	353542	49187
3×8	297542	212487	510029	69954
4×5	302065	228880	530945	130581
5×4	1461949	880708	2342657	1561760
3×9	1186057	1291773	2477830	230508

Table 2: Number of descents required for solving various sizes of Connect Four using mobility

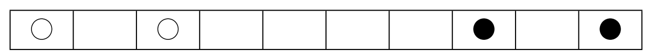

Figure 4: Woodpush starting position on size 10

Table 2 gives the corresponding numbers for internal frontiers nodes initialized with the mobility. The number of descents for GPNS is 1.5 times to 10.75 times smaller than the number of descents required for PNS.

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. A starting position (see figure 4) 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 4. A Left move consists in sliding one of the left piece 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 a previous position. The game is won when the opponent has no more pieces on the board. The score of a won game is the number of moves that the winner can play before the board is completely empty.

The first column of table 3 gives the size S of the Wood-

		PNS				GPNS
S	$\mathrm{real}(n)$	$\leq \mathrm{real}(n)$	$\geq \mathrm{real}(n)$	Sum	Dicho.	
6	-1	177	121	298	682	226
7	-1	281	924	1205	2744	1257
8	-1	1643	2570	4213	10867	4331
9	-1	6828	14814	21642	54417	16771
10	-2	37549	27009	64558	147383	71147
11	-1	80557	238088	318645	999552	388698

Table 3: Number of descents required for solving various sizes of Woodpush

S	real(n)	PNS $\leq$ real(n)	$\geq$ real(n)	Sum	Dicho.	GPNS
6	-1	141	97	238	457	236
7	-1	192	681	873	2223	1127
8	-1	1139	1718	2857	7039	3274
9	-1	4035	9609	13644	35166	12689
10	-2	30255	17124	47379	106499	53098
11	-1	43605	142374	185979	603411	261128

Table 4: Number of descents required for solving various sizes of Woodpush using mobility

push line board. The second column gives the optimal outcome of the board. The third column gives the number of descents required by PNS to prove the outcome is smaller than or equal to the optimal outcome. The fourth column gives the number of descents required by PNS to prove the outcome is greater than or equal to the optimal outcome. The fifth column is the sum of the two previous numbers, it is the number of descents required by PNS to prove the outcome if we already know the outcome of the game. The sixth column is the number of descents required if a dichotomic search on the outcomes is used to decide which PNS are tried to find the optimal outcome, the outcomes ranging from -16 to 16. The last column gives the number of descents of GPNS to prove the optimal outcome. We can observe that the number of descents of GPNS are close to the number of descents of Sum. They are 1.21 to 0.81 times smaller. When compared to the dichotomic search GPNS has 2 to 3 times fewer descents.

Table 4 gives the corresponding results using mobility. Again the number of descents of GPNS is close to the number of descents of Sum.

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. In GPNS these nodes are developed only once. For small Connect Four boards, GPNS solves the games with up to 21 times less descents than PNS. For Woodpush, GPNS solves the games with a number of descents close to the number of descents used by PNS if it already knows the optimal outcome of the game. In future work we plan to adapt the PN^2 algorithm to GPNS, possibly leading to a $GPNS^2$ algorithm that exchanges time for memory.

References

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

[Allis, 1988] 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.

[Allis, 1994] Louis Victor Allis. *Searching for Solutions in Games an Artificial Intelligence*. Phd thesis, Vrije Universitat Amsterdam, Maastricht, 1994.

[Cazenave and Saffidine, 2010] Tristan Cazenave and Abdallah Saffidine. Score bounded Monte-Carlo tree search. In *Computer and Games*, 2010.

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

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

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

[Nagai, 1999] A. Nagai. A new depth-first search algorithm for AND/OR trees. Master's thesis, University of Tokyo, 1999.

[Nagai, 2002] A. Nagai. Df-pn algorithm for searching AND/OR trees and its applications. Phd thesis, Department of Information Science, University of Tokyo, 2002.

[Russell and Norvig, 2002] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2 edition, December 2002.

[Schadd *et al.*, 2008] 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.

[Schaeffer *et al.*, 2007] 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.

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

[Schaeffer, 2007] Jonathan Schaeffer. Game over: Black to play and draw in Checkers. *ICGA Journal*, 30(4):187–197, 2007.

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

[Seo, 1995] M. Seo. The C* algorithm for AND/OR tree search and its application to a tsume-shogi program. Master's thesis, Departement of Information Science, University of Tokyo, 1995.

[van den Herik and Winands, 2008] H. Jaap van den Herik and Mark Winands. Proof-Number Search and Its Variants. *Oppositional Concepts in Computational Intelligence*, pages 91–118, 2008.