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 $\text{Max}$ and $\text{Min}$. $\mathcal{O} = \{o_1, \ldots, o_m\}$ denotes the possible outcomes of the game. We assume that the outcomes are linearly ordered with the following preference relation for $\text{Max}$: $o_1 <_{\text{Max}} \cdots <_{\text{Max}} o_m$, we further assume that the game is zero-sum and derive a preference relation for $\text{Min}$: $o_m <_{\text{Min}} \cdots <_{\text{Min}} o_1$. 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 \( o_i <_{\text{Max}} o_j \) (resp. \( o_i <_{\text{Min}} o_j \)).

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 \( n \) an estimation of the remaining effort needed to disprove the node. When \( n \) is external, for which the proof and disproof numbers are nodes, in which case the node is already solved. The Fron-

Win in different ways depending on its type. They are summa-

\[ \sum_{c \in \text{chil}(n)} \text{PN}(c) \] and \[ \sum_{c \in \text{chil}(n)} \text{DN}(c) \]

\[
\begin{array}{|c|c|c|}
\hline
\text{Node type} & \text{PN} & \text{DN} \\
\hline
\text{Win} & 0 & \infty \\
\text{Lose} & \infty & 0 \\
\text{Frontier} & 1 & 1 \\
\text{Max} & \min_{c \in \text{chil}(n)} \text{PN}(c) & \sum_{c \in \text{chil}(n)} \text{DN}(c) \\
\text{Min} & \sum_{c \in \text{chil}(n)} \text{PN}(c) & \min_{c \in \text{chil}(n)} \text{DN}(c) \\
\hline
\end{array}
\]

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 de-
veloped nodes where Max (resp. Min) is to play. For such
odes, 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 de-
scended 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 up-
dated 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 out-
comes: \( \emptyset = \{ \text{Win}, \text{Draw}, \text{Lose} \} \). 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 mini-
max value of the starting position can still be found with
PNS using a dichotomic search [Allis et al., 1994]. This di-
ichotomic 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
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 alphabeta 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, \( G(n, o) \), 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, \( S(n, o) \), 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 \( G(n, o) = S(n, o) = 0 \) then \( n \) is solved and its value is \( o \): \( \text{real}(n) = o \).

Figure 2 features an example of effort numbers for a three outcomes game. The effort numbers show that in the position, for all node \( o \) we note it \( o \).

### 4.2 Determination of the effort

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

### 4.3 Properties

\( G(n, o) = 0 \) (resp. \( S(n, o) = 0 \)) 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 \( G(n, o) = \infty \) 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 \( G(n, o_i) \leq G(n, o_j) \) and \( S(n, o_i) \geq S(n, o_j) \). Intuitively, this property states that the better an outcome is, the harder it will be to obtain it or to obtain better.

0 and \( \infty \) are permanent values since when an effort number reached 0 or \( \infty \), 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 \( G(n, o) = 0 \) then for all \( o' < o \), \( S(n, o') = \infty \) and similarly if \( S(n, o) = 0 \) then for all \( o' > o \), \( G(n, o') = \infty \).

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

### 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 \( o^*(n) = \arg\min_{o', G(n, o') > 0} (G(n, o) + S(n, o')) \) and similarly for Min nodes \( o^*(n) = \arg\min_{o, S(n, o) > 0} (G(n, o) + S(n, o)) \). 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 \( G(n, o) \neq \infty \) and \( S(n, o) \neq \infty \). Hence, \( G(n, o^*(n)) + S(n, o^*(n)) \neq \infty \).

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

<table>
<thead>
<tr>
<th>Outcome</th>
<th>( G(n, o) )</th>
<th>( S(n, o) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win</td>
<td>500</td>
<td>3</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Lose</td>
<td>0</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

![Figure 2: Example of effort numbers for a 3 outcome game](image-url)

<table>
<thead>
<tr>
<th>Node type</th>
<th>( G(n, o) )</th>
<th>( S(n, o) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>( \min_{c \in \text{chil}(n)} G(c, o) )</td>
<td>( \sum_{c \in \text{chil}(n)} S(c, o) )</td>
</tr>
<tr>
<td>Max</td>
<td>( \sum_{c \in \text{chil}(n)} G(c, o) )</td>
<td>( \min_{c \in \text{chil}(n)} S(c, o) )</td>
</tr>
</tbody>
</table>

(a) Internal node

<table>
<thead>
<tr>
<th>Outcome</th>
<th>( G(n, o) )</th>
<th>( S(n, o) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>0</td>
</tr>
<tr>
<td>\ldots</td>
<td>( \infty )</td>
<td>0</td>
</tr>
<tr>
<td>\text{real}(n)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\ldots</td>
<td>0</td>
<td>( \infty )</td>
</tr>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

(b) Terminal node

![Figure 3: Determination of effort numbers for GPNS](image-url)
\( o'(n) = o_{\text{max}}(k-1,1) \) and if \( n \) is a \( \text{Min} \) node, then \( o'(n) = o_{\text{min}}(k+1,m) \). As the name indicates, the distracting outcome
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: \( G(n,o) = 1, S(n,o) = |\text{chil}(n)| \). In a Min node, we would have \( G(n,o) = |\text{chil}(n)|, S(n,o) = 1 \).

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 independently 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 \( 7 \times 6 \) 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 game is not won. The third column is the number of descents required by PNS to prove the game is a draw. 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.

<table>
<thead>
<tr>
<th>Size</th>
<th>PNS</th>
<th>GPNS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \leq \text{Draw} )</td>
<td>( \geq \text{Draw} )</td>
</tr>
<tr>
<td>3 ( \times ) 4</td>
<td>1618</td>
<td>673</td>
</tr>
<tr>
<td>3 ( \times ) 5</td>
<td>7799</td>
<td>9403</td>
</tr>
<tr>
<td>4 ( \times ) 3</td>
<td>11427</td>
<td>10888</td>
</tr>
<tr>
<td>3 ( \times ) 6</td>
<td>21746</td>
<td>15759</td>
</tr>
<tr>
<td>4 ( \times ) 4</td>
<td>79601</td>
<td>33393</td>
</tr>
<tr>
<td>3 ( \times ) 7</td>
<td>150172</td>
<td>95159</td>
</tr>
<tr>
<td>5 ( \times ) 3</td>
<td>419952</td>
<td>190813</td>
</tr>
<tr>
<td>4 ( \times ) 5</td>
<td>402603</td>
<td>304862</td>
</tr>
<tr>
<td>3 ( \times ) 8</td>
<td>750745</td>
<td>493702</td>
</tr>
<tr>
<td>5 ( \times ) 4</td>
<td>2220291</td>
<td>1708671</td>
</tr>
<tr>
<td>3 ( \times ) 9</td>
<td>2678172</td>
<td>2992236</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Size</th>
<th>PNS</th>
<th>GPNS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \leq \text{Draw} )</td>
<td>( \geq \text{Draw} )</td>
</tr>
<tr>
<td>3 ( \times ) 4</td>
<td>863</td>
<td>615</td>
</tr>
<tr>
<td>3 ( \times ) 5</td>
<td>3571</td>
<td>2227</td>
</tr>
<tr>
<td>4 ( \times ) 3</td>
<td>3465</td>
<td>3502</td>
</tr>
<tr>
<td>3 ( \times ) 6</td>
<td>10958</td>
<td>12304</td>
</tr>
<tr>
<td>4 ( \times ) 4</td>
<td>43085</td>
<td>19965</td>
</tr>
<tr>
<td>3 ( \times ) 7</td>
<td>59821</td>
<td>45540</td>
</tr>
<tr>
<td>5 ( \times ) 3</td>
<td>220947</td>
<td>132595</td>
</tr>
<tr>
<td>4 ( \times ) 5</td>
<td>297542</td>
<td>212487</td>
</tr>
<tr>
<td>3 ( \times ) 8</td>
<td>302065</td>
<td>228880</td>
</tr>
<tr>
<td>5 ( \times ) 4</td>
<td>1461949</td>
<td>880708</td>
</tr>
<tr>
<td>3 ( \times ) 9</td>
<td>1186057</td>
<td>1291773</td>
</tr>
</tbody>
</table>

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

#### 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 given on an array of predefined length as shown in Figure 4. A 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 Woodpush board. Table 3 gives the corresponding numbers for internal frontier 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.

<table>
<thead>
<tr>
<th>Size</th>
<th>PNS</th>
<th>GPNS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \leq \text{real}(n) )</td>
<td>( \geq \text{real}(n) )</td>
</tr>
<tr>
<td>3 ( \times ) 4</td>
<td>6828</td>
<td>14814</td>
</tr>
<tr>
<td>3 ( \times ) 5</td>
<td>59821</td>
<td>45540</td>
</tr>
<tr>
<td>4 ( \times ) 3</td>
<td>297542</td>
<td>212487</td>
</tr>
<tr>
<td>3 ( \times ) 6</td>
<td>302065</td>
<td>228880</td>
</tr>
<tr>
<td>5 ( \times ) 3</td>
<td>1461949</td>
<td>880708</td>
</tr>
<tr>
<td>3 ( \times ) 9</td>
<td>1186057</td>
<td>1291773</td>
</tr>
</tbody>
</table>

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

---

Figure 4: Woodpush starting position on size 10
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 to GPNS to prove the optimal outcome. We can observe that the number of descents of GPNS is close to the number of descents of Sum.

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 PN² algorithm to GPNS, possibly leading to a GPNS² algorithm that exchanges time for memory.

References


