Deep df-pn and its Efficient Implementations

ZHANG SONG ^{a,1}, HIROYUKI IIDA ^{a,2}, and JAAP VAN DEN HERIK ^{b,3}

[a] *Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan*

[b] *Leiden Centre of Data Science, Leiden, the Netherlands*

Abstract. Depth-first proof-number search (df-pn) is a powerful variant of proof-number search algorithms, widely used for AND/OR tree search or solving games. However, df-pn suffers from the seesaw effect, which strongly hampers the efficiency in some situations. This paper proposes a new proof number algorithm called Deep depth-first proof-number search (Deep df-pn) to reduce the seesaw effect in df-pn. The difference between Deep df-pn and df-pn lies in the proof number or disproof number of unsolved nodes. It is 1 in df-pn, while it is a function of depth with two parameters in Deep df-pn. By adjusting the value of the parameters, Deep df-pn changes its behavior from searching broadly to searching deeply. The paper shows that the adjustment is able to reduce the seesaw effect convincingly. For evaluating the performance of Deep df-pn in the domain of Connect6, we implemented a relevance-zone-oriented Deep df-pn that worked quite efficiently. The experimental results indicate that improving efficiency by the same adjustment technique is also possible in other domains.

Keywords. df-pn, seesaw effect, parameters, Connect6

1. Introduction: from PN-search to Deep df-pn

Proof-Number Search (PN-search) [1] is one of the most powerful algorithms for solving games and complex endgame positions. PN-search focuses on AND/OR tree and tries to establish the game theoretical value in a best-first manner. Each node in PN-search has a proof number (pn) and disproof number (dn). This idea was inspired by the concept of conspiracy numbers, the number of children that need to change their value to make a node change its value [6]. A proof (disproof) number shows the scale of difficulty in proving (disproving) a node. PN-search expands the most-proving node, which is the most efficient one for proving (disproving) the root.

Although PN-search is an effective AND/OR-tree search algorithm, it still has some problems. We mention two of them. The first one is that PN-search uses a large amount of memory space because it is a best-first algorithm. The second one is that the algorithm is not efficient as hoped for because of the frequently updating of the proof and disproof

[1] E-mail: zhangsong@jaist.ac.jp.

[2] E-mail: iida@jaist.ac.jp.

[3] E-mail: jaapvandenherik@gmail.com.

numbers. So, Nagai [5] proposed a depth-first algorithm using both proof number and disproof number based on PN-search, which is called depth-first proof-number search (df-pn). The procedure of df-pn can be characterized as (1) selecting the most-proving node, (2) updating thresholds of proof number or disproof number in a transposition table, and (3) multiple iterative deepening until the ending condition is satisfied. Nagai proved the equivalence between PN-search and df-pn [5]. He noticed that df-pn always selects the most-proving node as PN-search does in the searching path. Moreover, its depth-first manner and the use of a transposition table give df-pn two clear advantages: (1) df-pn saves more storage, and (2) it is more efficient than PN-search.

Yet, both PN-search and df-pn suffer from the seesaw effect which can be characterized as frequently going back to the ancestor nodes for selecting the most-proving node, as described in [8,10,11]. They showed that the seesaw effect works strongly against the efficiency in some situations. In Ishitobi et al. [14], the seesaw effect was discussed in relation to PN-search. The authors arrived at a DeepPN search. However, DeepPN has in turn still at least two drawbacks: (1) it suffers from a big cost of storage as PN-search, and (2) DeepPN spends much time on updating the proof and disproof number, which makes DeepPN actually not an efficient algorithm. This paper proposes a Deep depth-first proof-number search algorithm (Deep df-pn) to reduce the seesaw effect in df-pn. The difference between Deep df-pn and df-pn lies in the proof number or disproof number of unsolved nodes. In df-pn the proof number or disproof number of unsolved nodes is 1, while in Deep df-pn it is a function of depth with two parameters. By adjusting the value of parameters, Deep df-pn changes its behavior from searching broadly to searching deeply. It will be proved in this paper that doing so will be able to reduce the seesaw effect convincingly.

To evaluate the performance of Deep df-pn, we implement a relevance-zone-oriented Deep df-pn to make it work efficiently in the domain of Connect6 [2]. The concept of relevance zone in Connect6 is introduced by Wu and Lin [13]. It is a zone of the board in which the defender has to place at least one of the two stones, otherwise the attacker will simply win by playing a VCDT (victory by continuous double threat) strategy. Such a zone indicates which moves are necessary for the defender to play. It helps to cut down the branch size of the proof tree. With a relevance zone, Deep df-pn can solve positions of Connect6 efficiently. Experimental results show its good performance in improving the search efficiency.

The remainder of the paper is as follows. We briefly summarize the details of PN-search and df-pn in Section 2, and introduce the seesaw effect in Section 3. Definitions of Deep df-pn and its characteristics are presented in Section 4. In Section 5, we introduce the relevance-zone-oriented Deep df-pn for Connect6. Then, we conduct experiments to show its better performance in reducing the seesaw effect in Section 6. Finally, concluding remarks are given in Section 7.

2. PN-Search and its Depth-First Variant

In this section, we summarize the original proof-number search (PN-search) and depth-first proof-number search (df-pn), a depth-first variant with advantages on space saving and efficiency.

2.1. PN-Search

Proof-Number Search (PN-search) [1] is a native best-first algorithm, using proof numbers and disproof numbers, always expanding one of the most-proving nodes. All nodes have proof and disproof numbers, they are stored to indicate which frontier node should be expanded, and updated after expanding. The node to be expanded is called the *most-proving node*. It is considered the most efficient one for proving (disproving) the root.

By exploiting the search procedure, two characteristics of the search tree are established [7]: (1) the shape (determined by the branching factor of every internal node), and (2) the values of the leaves (in the end they deliver the game theoretic value). Basically, unenhanced PN-search is an uninformed search method that does not require any game-specific knowledge beyond its rules [3]. The formalism is given in [1].

2.2. Df-pn

Although PN-search is an ideal AND/OR-tree search algorithm, it still has at least two problems (see section 1). To solve the problems, Nagai [5] proposed a depth-first like algorithm using both proof number and disproof number. He called it df-pn (depth-first proof-number search). The procedure of df-pn can be characterized as (1) selecting the most-proving node, (2) updating the thresholds of proof number or disproof number in a transposition table, and (3) applying multiple iterative deepening until the ending condition is satisfied. Although df-pn is a depth-first like search, it has a same behavior as PN-search. The equivalence between PN-search and df-pn is proved in [5].

In df-pn, proof number and disproof number are renamed as follows.

$$n.\phi = \begin{cases} n.pn & (\text{n is an OR node}) \\ n.dn & (\text{n is an AND node}) \end{cases}, \quad n.\delta = \begin{cases} n.dn & (\text{n is an OR node}) \\ n.pn & (\text{n is an AND node}) \end{cases}$$

Moreover, each node n has two thresholds: one for the proof number th_{pn} and the other for the disproof number th_{dn}. Similarly, th_{pn} and th_{dn} are renamed as follows.

$$n.th_{\phi} = \begin{cases} n.th_{pn} & (\text{n is an OR node}) \\ n.th_{dn} & (\text{n is an AND node}) \end{cases}, \quad n.th_{\delta} = \begin{cases} n.th_{dn} & (\text{n is an OR node}) \\ n.th_{pn} & (\text{n is an AND node}) \end{cases}$$

Df-pn expands the same frontier node as PN-search in a depth-first manner guided by a pair of thresholds (th_{pn}, th_{dn}), which indicates whether the most-proving node exists in the current subtree [4]. The procedure is described below [5].

Procedure Df-pn

For the root node r, assign values for $r.th_{\phi}$ and $r.th_{\delta}$ as follows.

$$r.th_{\phi} = \infty, \quad r.th_{\delta} = \infty$$

Step 1. At each node n, the search process continues to search below n until $n.\phi \geq n.th_{\phi}$ or $n.\delta \geq n.th_{\delta}$ is satisfied (we call it ending condition).

Step 2. At each node n, select the child n_c with minimum δ and the child n_2 with second minimum δ. (If there is another child with minimum δ, that is n_2.) Search below n_c with assigning

$$n_c.th_\phi = n.th_\delta + n_c.\phi - \sum n_{child}.\phi\ , \quad n_c.th_\delta = \min\left(n.th_\phi, n_2.\delta + 1\right)$$

Repeat this process until the ending condition holds.

Step 3. If the ending condition is satisfied, the search process returns to the parent node of n. If n is the root node, then the search is totally over.

3. Seesaw Effect and DeepPN

In this section, we introduce the seesaw effect and DeepPN, a variant of PN-search focusing on reducing the seesaw effect.

3.1. Seesaw Effect

PN-search and df-pn are highly efficient in solving games. However, both are facing the drawback named as *seesaw effect* [9]. It can be best characterized as frequently going back to the ancestor nodes for selecting the most-proving node.

To explain it precisely, we show, in Fig. 1, an example where the root node has two subtrees. The size of both subtrees is almost the same. Assume that the proof number of subtree L is larger than the proof number of subtree R. In this case, PN-search or df-pn will continue search in subtree R, which means that the most-proving node is in subtree R. After PN-search or df-pn has expanded the most-proving node, the shape of the game tree will change as shown in Fig. 1(b). By expanding the most-proving node, the proof number of subtree R becomes larger than the proof number of subtree L. So PN-search or df-pn changes its searching direction from subtree R to subtree L. In turn, when the search expands the most-proving node in subtree L, then the proof number of subtree L becomes larger than the one in subtree R. Thus, the search changes its focus from subtree L to subtree R. This change keeps going back and forth, which looks like a seesaw. Therefore, it is named as seesaw effect.

The seesaw effect happens when the two trees are almost equal in size. If the seesaw effect occurs frequently, the performance of PN-search and df-pn deteriorates significantly and cannot reach the required search depth. In games which need to reach a large fixed search depth, the seesaw effect works strongly against efficiency.

The seesaw effect is mostly caused by two issues: the shape of game tree and the way of searching. Concerning the shape of game tree, there are two characteristics: (1) a tendency of the newly generated children to keep the size equal and (2) the fact that many nodes with equal values exist deep down in a game tree. If the structure of each node remains almost the same (cf. characteristic 1), then the seesaw effect may occur easily. For characteristic 2, it is common in games such as Othello and Hex to search a large fixed number of moves before settling. This is also the case in connect-type games such as Gomoku and Connect6 which have a sudden death in the game tree. Therefore, it is necessary to design a new search algorithm to reduce the seesaw effect in these games.

3.2. DeepPN

To tackle the seesaw effect problem, Ishitobi et al. [14] proposed Deep Proof-Number Search (DeepPN), a variant of PN-search while focusing on reducing the seesaw effect. It employs two important values associated with each node, the usual proof number and

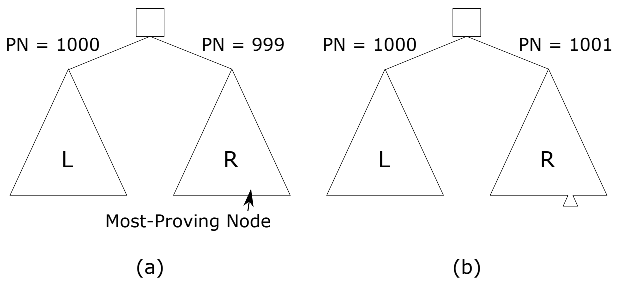

Figure 1. An example of seesaw effect [14]: (a) An example game tree (b) Expanding the most-proving node

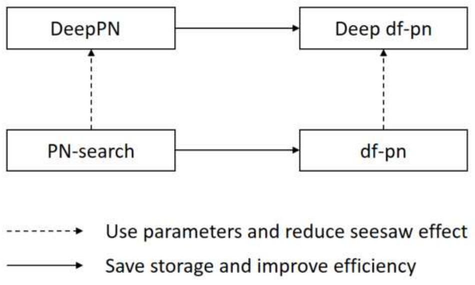
DeepPN
Deep df-pn
PN-search
df-pn
Use parameters and reduce seesaw effect
Save storage and improve efficiency

Figure 2. Relationship between PN-search, df-pn, DeepPN and Deep df-pn

a deep value called *R*. The deep value is defined as the depth of a node which shows the progress of the search in the depth direction. After mixing the proof numbers and the deep value, the DeepPN can change its behavior from the best-first manner (equal to the original proof-number search) to the depth-first manner by adjusting the parameter *R*. Compared to the original PN-search, DeepPN shows better results when *R* comes to a proper value which defines the nature of the search to be between best-first search and depth-first search. The formal definitions of DeepPN are described in [14].

4. Deep df-pn

In this section, we propose a new proof-number algorithm based on df-pn to cover the shortage of DeepPN (see section 1), named as Deep Depth-First Proof-Number Search or Deep df-pn in short. It not only extends the improvements of df-pn on (1) saving storage and (2) efficiency, but also (3) reduces the seesaw effect. Fig. 2 shows the relationship between PN-search, df-pn, DeepPN, and Deep df-pn.

Similar to DeepPN, the proof number and disproof number of unsolved nodes are adjusted in Deep df-pn by a function of depth with two parameters. By adjusting the values of the two parameters, Deep df-pn can change its behavior from searching broadly to searching deeply (and vice versa). Definitions of Deep df-pn are given below.

In Deep df-pn, the proof number and disproof number of node *n* are calculated as given in section 2.2 (here repeated for readability).

$$n.\phi = \begin{cases} n.pn & (\text{n is an OR node}) \\ n.dn & (\text{n is an AND node}) \end{cases}, \quad n.\delta = \begin{cases} n.dn & (\text{n is an OR node}) \\ n.pn & (\text{n is an AND node}) \end{cases}$$

When n is a leaf node, there are three cases.
(a) When n is proved (disproved) and n is an OR (AND) node, i.e., OR wins

$$n.\phi = 0\,, \quad n.\delta = \infty$$

(b) When n is proved (disproved) and n is an AND (OR) node, i.e., OR does not win

$$n.\phi = \infty\,, \quad n.\delta = 0$$

(c) When the value of n is unknown

$$n.\phi = \underline{Ddfpn}(n.depth)\,, \quad n.\delta = \underline{Ddfpn}(n.depth)$$

When n is an internal node, the proof and disproof number are defined as follows

$$n.\phi = \underset{n_c \in \text{children of n}}{Min} n_c.\delta\,, \quad n.\delta = \sum_{n_c \in \text{children of n}} n_c.\phi$$

Definition 1 *$\underline{Ddfpn}(x)$ is a function from $\mathbb{N}$ to $\mathbb{N}$, which*

$$\underline{Ddfpn}(x) = \begin{cases} E^{D-x} & (D > x \land E > 0) \\ 1 & (D \leq x \land E > 0) \\ 0 & (E = 0) \end{cases}$$

where E and D are parameters on $\mathbb{N}$, E denotes a threshold of branch size and D denotes a threshold of depth.

The complete algorithm of Deep df-pn is accessible on the website[4]. Table 1 shows the behavior of Deep df-pn with different values of E and D. When $E = 0$, Deep df-pn is a depth-first search. When $E > 1$ and $D > 1$, Deep df-pn is an intermediate search procedure between depth-first search and df-pn. For other cases, Deep df-pn is the same as df-pn. Deep df-pn focuses on changing the search behavior of df-pn. The procedure of selecting the most proving node in df-pn is controlled by the thresholds of proof number and disproof number. So changing the search behavior of df-pn can be implemented by two methods: (1) changing the thresholds of proof number and disproof number (such as $1+\varepsilon$ trick [8]); (2) changing the proof number and disproof number of unsolved nodes. Deep df-pn implements the method (2). If E or D becomes smaller, Deep df-pn tends to search more broadly usually with more seesaw effect. If E or D becomes larger, Deep df-pn tends to search more deeply usually with less seesaw effect. Below we will prove that Deep df-pn helps reduce the seesaw effect in df-pn.

Theorem 1 *Deep df-pn outperforms df-pn in reducing the seesaw effect.*

[4]`http://www.jaist.ac.jp/is/labs/iida-lab/Deep_df_pn_Algorithm.pdf`

Proof Assume that node n is a most-proving node in a seesaw effect (see Fig. 1(b)). Without loss of generality, n is an AND node in subtree L. According to the feature of the seesaw effect, after expanding n, its proof number becomes larger, which makes the proof number of subtree L larger. Then df-pn changes its focus on subtree R and the seesaw effect happens.

From the definitions of Deep df-pn, the proof number of n is given by: $\underline{Ddfpn}(n.depth)$. After expanding n, its proof number is given by

$$\sum_{\text{children of n}} \underline{Ddfpn}(n.depth+1) = E^{'} \cdot \underline{Ddfpn}(n.depth+1)\,.$$

where $E^{'}$ denotes the number of children of n. If $E^{'} \leq E$ and $n.depth+1 < D$, then we have $E^{'} \cdot \underline{Ddfpn}(n.depth+1) = E^{'} \cdot E^{D-(n.depth+1)}$ and $E^{'} \cdot E^{D-(n.depth+1)} \leq E^{D-depth}$. So we obtain the following inequation

$$\sum_{\text{children of n}} \underline{Ddfpn}(n.depth+1) \leq \underline{Ddfpn}(n.depth)\,.$$

Therefore, Deep df-pn continues focusing on subtree L and the seesaw effect does not occur. For a certain proof tree, the degree of reducing the seesaw effect increases as the value of E increases. As a result, Deep df-pn outperforms df-pn in reducing the seesaw effect. □

Table 1. Different behaviors by changing parameters

	$E=0$	$E=1$	$E>1$
$D=0$	Depth-first	Df-pn	Df-pn
$D=1$	Depth-first	Df-pn	Df-pn
$D>1$	Depth-first	Df-pn	Intermediate

5. Deep df-pn in Connect6

In this section, we implement a relevance-zone-oriented Deep df-pn and make Deep df-pn work efficiently in Connect6. We first introduce the game of Connect6, then present the structure of relevance-zone-oriented Deep df-pn.

5.1. Connect6

Connect6 is a two-player strategy game similar to Gomoku. It is first introduced by Wu and Huang [2] as a member of the connect games family. The game of Connect6 is played as follows. Black (first player) places one stone on the board for its first move. Then both players alternatively place two stones on the board at their turn. The player who first obtains six or more stones in a row (horizontally, vertically or diagonally) wins the game. Connect6 is usually played on a (19×19) Go board. Both the state-space and game-tree complexities are much higher than those in Gomoku and Renju. The reason is that two stones per move results in an increase of branching factor by a factor of half of the board size. Based on the standard used in [12], the state-space complexity of Connect6 is 10^{172}, the same as that in Go, and the game-tree complexity is 10^{140}, much higher than that for Gomoku. If a larger board is used, the complexity is much higher. So finding a way to cut down the branch size of the proof tree is important for solving positions of Connect6. In [2], Wu and Huang showed a type of winning strategy by making continuously double-threat moves and ending with a triple-or-more-threat move or connecting up to six in all variations. This is called victory by continuous double-threat-or-more moves (VCDT). Using a VCDT solver is a key method to reduce the complexity of solving a position of Connect6.

5.2. Relevance-zone-oriented Deep df-pn

The implementations of a relevance-zone-oriented Deep df-pn (i.e., a Deep df-pn procedure and a VCDT solver) is used to find winning strategies and to derive a relevance zone for Deep df-pn to cut down the branch size. According to the description in [13], the relevance zone is a zone of the board in which the defender has to place at least one of the two stones, otherwise the attacker will simply win by playing a VCDT strategy. Such a zone indicates which moves are necessary for the defender to play. It helps to cut down the branch size of the proof tree. The relation between Deep df-pn and the relevance zone is as follows. When Deep df-pn generates new moves for the defender, it first generates a null move which means that the defender places no stone for this move. Then the VCDT solver is started for the attacker. If a winning strategy is found, the VCDT solver derives a relevance zone Z (it is a zone where defense is necessary). Subsequently, the defender places one stone on each square s in Z to generate seminull moves. For each seminull move, the VCDT solver starts to derive a relevance zone Z' corresponding to this seminull move. As a result, all the necessary moves of the defender are generated by setting one stone on square s in Z and another on one square in Z' corresponding to the seminull move at s. The size of generated defender moves is far smaller than the one without relevance zone. For the next step, the VCDT solver starts to analyze the best move for each new position derived from these defender moves. If VCDT solver finds a winning strategy, then it returns a win to Deep df-pn. If not, Deep df-pn is continued recursively.

6. Experiments

In this section we chose Connect6 as a benchmark to evaluate the performance of Deep df-pn. We first present the experimental design, then we show and discuss the experimental results. Next, we compare the performance of Deep df-pn and $1+\varepsilon$ trick [8]. Finally, we propose a method to find the relatively optimized parameters of Deep df-pn.

6.1. Experimental Design

To solve the positions of Connect6, we use relevance-zone-oriented Deep df-pn. Each time Deep df-pn generates the defender's moves, the VCDT solver generates relevance zones to indicate the necessary moves which the defender has to set on the board. Here,

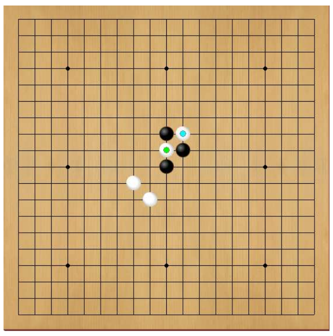

Figure 3. Example position 1 of Connect6 (Black is to move and Black wins)

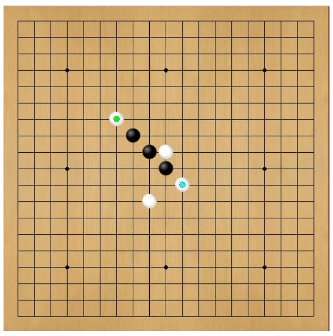

Figure 4. Example position 2 of Connect6 (Black is to move and Black wins)

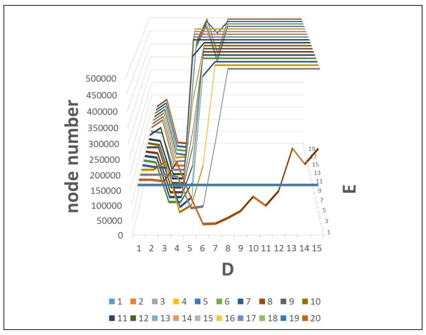

Figure 5. Deep df-pn and df-pn compared in node number (including repeatedly traversed nodes) with various values of parameter E and D for position 1 (Df-pn when $D = 1$)

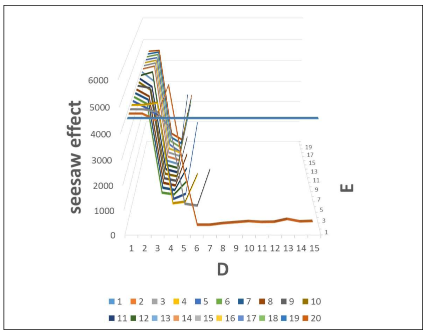

Figure 6. Deep df-pn and df-pn compared in seesaw effect number with various values of parameter E and D for position 1 (Df-pn when $D = 1$)

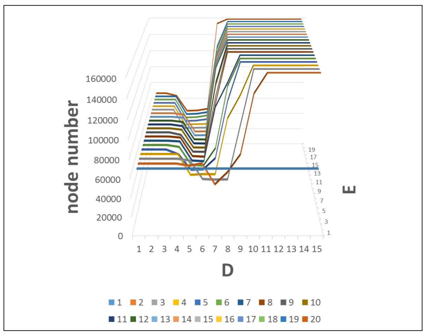

Figure 7. Deep df-pn and df-pn compared in node number (including repeatedly traversed nodes) with various values of parameter E and D for position 2 (Df-pn when $D = 1$)

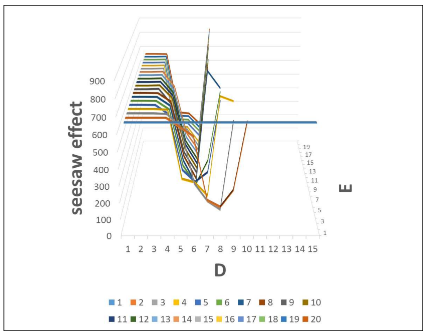

Figure 8. Deep df-pn and df-pn compared in seesaw effect number with various values of parameter E and D for position 2 (Df-pn when $D = 1$)

we remark that each time Deep df-pn generates the attacker's moves, it only generates the top 5 evaluated moves (according to some heuristic values) to reduce the complexity. Moreover, we did not recycle the child nodes after Deep df-pn has returned to its parent to reserve the winning path. Actually, these nodes *can* be recycled when Deep df-pn returns to its parent and are generated again for next time, if the cost of storage is considered. The VCDT solver is implemented with the techniques of iterative deepening and transposition table to control the time. It can search up to a depth of 25 where the longest winning path is 13 moves.

In this paper, we first investigate 2 positions: position 1 (see Fig. 3) and position 2 (see Fig. 4). We use Deep df-pn to solve these positions with various values of parameter E and D (D is from 1 to 15 with a step length 1, and E is from 1 to 20 with a step length 1. Totally we can get 300 results). Then we get a series of changing curves of the node number (see Fig. 5 and Fig. 7) and the seesaw effect number (see Fig. 7 and Fig. 8) for parameter E and D. In this paper, the node number equals VCDT node number + Deep df-pn node number. It includes repeatedly traversed nodes. And the seesaw effect number is initialized as 0 and increased by 1 when a node in Deep df-pn is traversed again. To obtain these curves efficiently, we set a threshold to the node number (500000 for position 1 and 160000 for position 2). When the node number of solving a position is already larger than the threshold, the solver will shut down to reduce the time cost, then we use the value of the threshold to replace the exact node numbers and use blank points to replace the exact seesaw effect numbers in the curves. The pattern of the search time is almost the same as the node number, so we do not show it in this paper.

Moreover, we select other 6 positions (see Appendix) which can be solved by df-pn and apply Deep df-pn to them. Among all the positions (together 8 positions), 4 positions (see Fig. 3, Fig. 4, Fig. 13, and Fig. 16) are four-moves opening (Black has 2 moves and White has 2 moves), and Fig. 14 is a special opening, in which White sets two useless stones for its first move and Black is proved to win. We apply Deep df-pn with the best selected E and D (E is selected out from 1 to 20, and D is selected out from 1 to 15) for each position, and present the experimental results of the 8 positions in Table 2 (column "Deep df-pn").

All the experiments are implemented on the computer with Windows 10 x64 system and Core i7-4790 CPU.

6.2. Results and Discussion

The first position of analysis is Fig. 3 (Black is to move and Black wins). If $E = 0$, Deep df-pn is a depth-first search which takes far more time than the original df-pn [5]. So we do not present it in this paper. If $E > 0$ and $D > 0$, a series of changing curves for each value of parameter E and D can be obtained as shown in Fig. 5, and Fig. 6 with respect to the node number, and the seesaw effect, respectively. According to the curves, if $D = 1$ or $E = 1$, Deep df-pn is the same as df-pn. As E and D increase within a boundary, the node number and the seesaw effect number decrease, because Deep df-pn is forced to search deeper and obtains the solution faster. If E or D becomes too large, Deep df-pn is forced to search too deep. As a result, it takes more cost and causes more seesaw effect in the search process. When E and D are well chosen, Deep df-pn can get an optimal

[5] In this section, Deep df-pn is actually a relevance-zone-oriented Deep df-pn and the original df-pn is a relevance-zone-oriented df-pn for the application in Connect6.

performance for a certain position. The second position of investigation is Fig. 4 (Black is to move and Black wins). It has a similar result as above. The changing curves obtained from Fig. 4 are presented in Fig. 7 and Fig. 8.

We also conduct experiments on other 6 positions (see Appendix) and present the experimental results of all the positions (together 8 positions) in Table 2 (column "Deep df-pn"). The experimental data is generated by Deep df-pn solver with the best selected parameter E and D (E is selected out from 1 to 20, and D is selected out from 1 to 15) for each position. According to the table, we can conclude that Deep df-pn with the best selected parameters is more efficient than original df-pn, because it reduces the node number and the seesaw effect number significantly.

6.3. Comparision

There is other techniques also trying to solve the seesaw effect, such as $1+\varepsilon$ trick [8]. The algorithm of $1+\varepsilon$ trick is almost the same as original df-pn. The only difference is the way of calculating the threshold $n_c.th_\delta$, which is presented below.

$$n_c.th_\phi = n.th_\delta + n_c.\delta - \sum n_{\text{child}}.\phi \ , \qquad n_c.th_\delta = \min(n.th_\phi, \lceil n_2.\delta(1+\varepsilon)\rceil)$$

ε is a real number bigger than zero. If ε increases, $1+\varepsilon$ trick searches deeper and usually has less seesaw effect. If ε equals a very small number, $1+\varepsilon$ trick works the same as the df-pn.

To compare the performance, we implement a $1+\varepsilon$ trick solver and conduct experiments on position 1 (see Fig. 3) and position 2 (see Fig. 4). The experimental results of position 1 are presented in Fig. 9 and Fig. 10. And the experimental results of position 2 are presented in Fig. 11 and Fig. 12. These figures show the changing curves of the node number and the seesaw effect number for various values of parameter ε. Here, ε is from 0.05 to 15 with a step length 0.05 (totally 300 items). To obtain these curves efficiently, we set a threshold to the node number (500000 for Fig. 3 and 160000 for Fig. 4). When the node number of solving a position is already larger than the threshold, the solver will shut down to reduce the time cost, and then we use the threshold value to replace the exact node numbers and use blank points to replace the exact seesaw effect numbers in the curves. According to the figures, the curves of $1+\varepsilon$ trick are not so consistent as Deep df-pn's, so it is more likely affected by the *noise effect*. The noise effect can be concluded as hugely jumping up or down of solving time caused by slightly changing parameters of the modification which forces df-pn to stay longer in a subtree to avoid frequently switching to another branch. Considering that ε is a real number with an infinitesimal scale, it is more difficult for $1+\varepsilon$ trick to find an optimal parameter ε in practice, while it is easy for Deep df-pn to find the optimal parameters by a hill-climbing method (see subsection 6.4).

To compare Deep df-pn with $1+\varepsilon$ trick more precisely, we collect experimental data on all the 8 positions. For each position, we select the best case (case with the least node number) of both two methods by adjusting the parameters and present them in Table 2. The results show that Deep df-pn has a better performance (less node number and less seesaw effect number) than $1+\varepsilon$ trick on average.

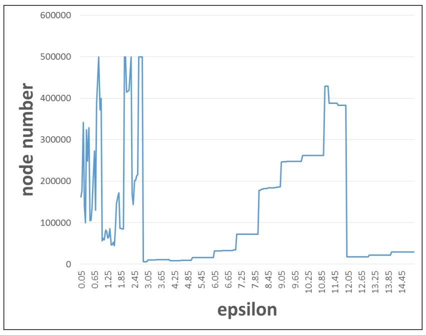

Figure 9. Node number (including repeatedly traversed nodes) of $1+\varepsilon$ trick with various values of parameter ε for position 1

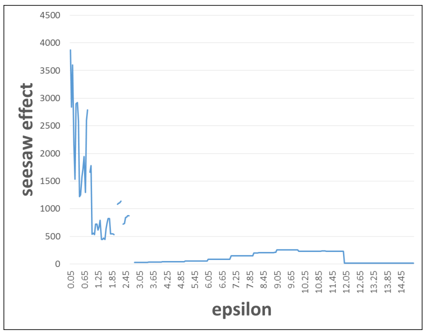

Figure 10. Seesaw effect number of $1+\varepsilon$ trick with various values of parameter ε for position 1

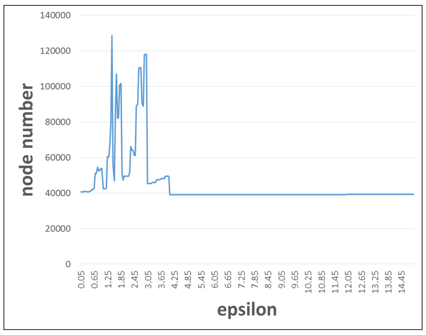

Figure 11. Node number (including repeatedly traversed nodes) of $1+\varepsilon$ trick with various values of parameter ε for position 2

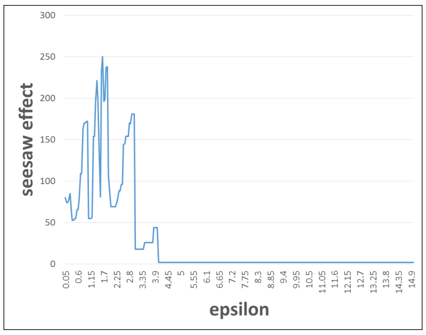

Figure 12. Seesaw effect number of $1+\varepsilon$ trick with various values of parameter ε for position 2

6.4. Finding optimized parameters

For finding the optimized parameters E and D of Deep df-pn, the hill-climbing method, a kind of local search for finding optimal solutions, is used. Although hill-climbing does not necessarily guarantee to find the best possible solution, it is efficient and allows Deep df-pn to obtain a relatively better performance than the original df-pn. To avoid the noise effect which makes hill-climbing stop too early, this method is implemented to ignore some local optimums. Here, we set the $node(E,D)$ as the target function for minimizing the value of the function. The parameters E and D are input and the node number is output. The node number is computed in real time by the relevance-zone-oriented Deep df-pn solver. To control the time, we set a node number threshold N to the solver. When the node number is already larger than the threshold N, the solver will shut down and the target function will return ∞ representing that current values of parameters are not optimal and will not be considered. Relevant details of the method are presented in Algorithm 1. The procedure $isNotFlat()$ returns false, if the value of $node(E,D)$ does

Table 2. Deep df-pn and $1+\varepsilon$ trick compared in the best case (The number in the bracket represents the reduction percentage compared with df-pn)

Position	Deep df-pn			$1+\varepsilon$ trick		
	Node number	Seesaw effect	E, D	Node number	Seesaw effect	ε
1	5568 (96.5%)	122 (97.3%)	17, 4	5633 (96.4%)	28 (99.4%)	2.85
2	45300 (33.7%)	101 (84.6%)	7, 6	38948 (43.0%)	2 (99.7%)	4.05
3	21157 (0.7%)	1 (95.2%)	5, 4	21309 (0%)	21 (0%)	0.05
4	99073 (17.1%)	128 (79.3%)	8, 6	95472 (20.2%)	372 (39.9%)	0.25
5	163 (99.8%)	0 (100%)	18, 2	82777 (5.7%)	936 (6.1%)	0.05
6	47213 (8.6%)	185 (35.8%)	14, 4	46255 (10.4%)	252 (12.5%)	0.15
7	74061 (45.9%)	582 (50.2%)	7, 4	143609 (-4.9%)	1158 (0.9%)	0.05
8	203188 (13.1%)	670 (38.1%)	5, 4	187198 (20.0%)	786 (27.4%)	0.25
average	61965.4 (43.5%)	223.6 (80.8%)		77650.1 (29.2%)	444.4 (61.9%)	

Table 3. Experimental data of Deep df-pn using hill-climbing method (The number in the bracket represents the difference between Deep df-pn using hill-climbing method and Deep df-pn in the best case)

Position	Node number	Seesaw effect	E, D	iteration time (s)
1	10529 (4961)	392 (270)	7, 4	159.7
2	45300 (0)	101 (0)	7, 6	208.6
3	21157 (0)	1 (0)	5, 4	66.4
4	107194 (8121)	912 (784)	6, 4	349.2
5	163 (0)	0 (0)	18, 2	346.5
6	50325 (3112)	268 (83)	3, 4	86.6
7	74061 (0)	582 (0)	7, 4	286.0
8	203188 (0)	670 (0)	5, 4	372.3
average	63989.6 (2024.3)	365.8 (142.1)		234.4

not change after several times iteration. And we call these continuous points (E,D) with a same value of $node(E,D)$ as a "flat".

The experimental results of the 8 positions are presented in Table 3. According to the table, by using hill-climbing method, Deep df-pn can get the same performance (the difference is 0) as its best case for most of the positions. On average, the difference from the best case is small (about 3.3%: $2024.3/(63989.6-2024.3)$) and the iteration time is also acceptable.

Algorithm 1 Hill-climbing method

```
1:  $E = 2; D = 2;$
2:  while $isNotFlat()$ do
3:      if $node(E+1,D) \leq node(E,D+1)$ then
4:          $E' = E+1; D' = D;$
5:      else
6:          $E' = E; D' = D+1;$
7:      end if
8:      if $node(E',D') > node(E,D)$ && $node(E'+1,D') > node(E',D')$
          && $node(E',D'+1) > node(E',D')$ then
9:          return $E$, $D$;
10:     end if
11:     $N = node(E',D'); E = E'; D = D';$
12: end while
13: return the minimum $E$ and $D$ on the flat;
```

7. Concluding Remarks

In this paper, we proposed a new proof-number algorithm called Deep Depth-First Proof-Number Search (Deep df-pn) to improve df-pn by reducing the seesaw effect. Deep df-pn is a natural extension of Deep Proof-Number Search (DeepPN) and df-pn. The relation between PN-search, df-pn, DeepPN and Deep df-pn was discussed. The main difference between Deep df-pn and df-pn is the proof number or disproof number of unsolved nodes. It is 1 in df-pn, while it is a function of depth with two parameters in Deep df-pn. By adjusting the values of the parameters, Deep df-pn changes its behavior from searching broadly to searching deeply which has been proved to be able to reduce the seesaw effect. For evaluating the performance of Deep df-pn, we implemented a relevance-zone-oriented Deep df-pn to make it work efficiently in the domain of Connect6. The experimental results show a convincing effectiveness (see Table 2) in search efficiency, provided that the parameters E and D are well chosen.

In this paper, Connect6 was chosen as a benchmark to evaluate the performance of Deep df-pn. Connect6 is a game with an unbalanced game tree (with a lot of sudden deaths). Our first recommendation is that further investigations will be made using other types of games with a balanced game tree (fix-depth tree or nearly fix-depth tree), such as Othello and Hex. Our second recommendation is that the procedure to find the optimal values of the parameters E and D is further analyzed and improved.

References

[1] L V. Allis, M. van der Meulen, H J. van den Herik. “Proof-number search,” *Artificial Intelligence*, vol. 66, no. 1, pp.91–124, 1994.

[2] I C. Wu, D Y. Huang. “A new family of k-in-a-row games,” *Advances in Computer Games*, Lecture Notes in Computer Science 4250, Springer Berlin Heidelberg, pp.180–194, 2005.

[3] A. Kishimoto, M H M. Winands, M. Müller, J T. Saito. “Game-tree search using proof numbers: The first twenty years,” *ICGA Journal*, vol. 35, no. 3, pp.131–156, 2012.

[4] T. Kaneko. “Parallel depth first proof number search,” *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pp.95–100, 2010.

[5] A. Nagai. “Df-pn algorithm for searching AND/OR trees and its applications,” *PhD Thesis*, Department of Information Science, University of Tokyo, 2002.

[6] D A. McAllester. “Conspiracy numbers for min-max search,” *Artificial Intelligence*, vol. 35, no. 3, pp.287–310, 1988.

[7] H J. van den Herik, M H M. Winands. “Proof-number search and its variants,” *Oppositional Concepts in Computational Intelligence*, Springer Berlin Heidelberg, pp.91–118, 2008.

[8] J. Pawlewicz, L. Lew. “Improving depth-first pn-search: $1+\varepsilon$ trick,” *International Conference on Computers and Games*, Lecture Notes in Computer Science 4630, Springer Berlin Heidelberg, pp.160–171, 2006.

[9] J. Hashimoto. “A study on game-independent heuristics in game-tree search,” *PhD Thesis*, School of Information Science, Japan Advanced Institute of Science and Technology, 2011.

[10] A. Kishimoto, M. Müller. “Search versus knowledge for solving life and death problems in Go,” *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, pp.1374–1379, 2005.

[11] A. Kishimoto. “Correct and efficient search algorithms in the presence of repetitions,” *PhD Thesis*, University of Alberta, 2005.

[12] H J. van den Herik, J W H M. Uiterwijk, J. van Rijswijck. “Games solved: Now and in the future,” *Artificial Intelligence*, vol. 134, no. 1, pp.277–311, 2002.

[13] I C. Wu, P H. Lin. “Relevance-zone-oriented proof search for connect6”. *IEEE Transactions on computational intelligence and AI in games*, vol. 2, no. 3, pp.191-207, 2010.

[14] T. Ishitobi, A. Plaat, H. Iida, J. van den Herik. “Reducing the seesaw effect with deep proof-number search,” *Advances in Computer Games*, Lecture Notes in Computer Science 9525, Springer International Publishing, pp.185–197, 2015.

Acknowledgement

The authors thank the referees for their constructive comments and suggestions for improvements.

Appendix

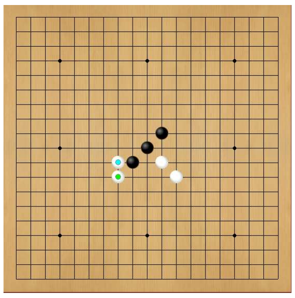

Figure 13. Example position 3 of Connect6 (Black is to move and Black wins)

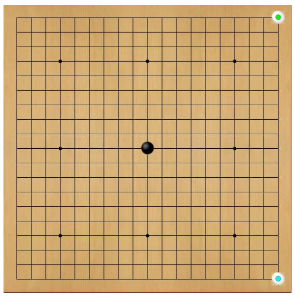

Figure 14. Example position 4 of Connect6 (Black is to move and Black wins)

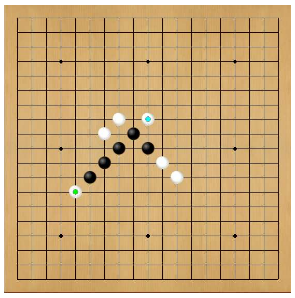

Figure 15. Example position 5 of Connect6 (Black is to move and Black wins)

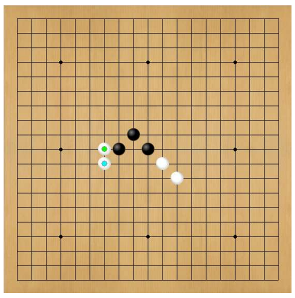

Figure 16. Example position 6 of Connect6 (Black is to move and Black wins)

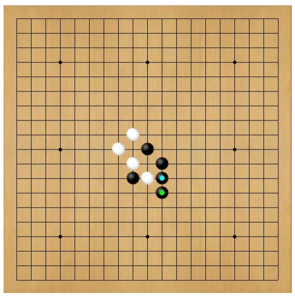

Figure 17. Example position 7 of Connect6 (White is to move and White wins)

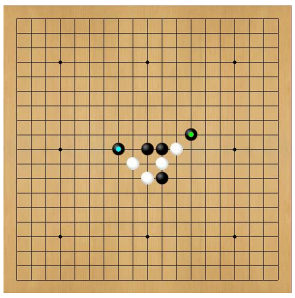

Figure 18. Example position 8 of Connect6 (White is to move and White wins)