Weak Proof-Number Search

Toru Ueda, Tsuyoshi Hashimoto, Junichi Hashimoto, and Hiroyuki Iida

School of Information Science,
Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
{s0610013,t-hashi,j-hashi,iida}@jaist.ac.jp

Abstract. The paper concerns an AND/OR-tree search algorithm to solve hard problems. Proof-number search is a well-known powerful search algorithm for that purpose. Its depth-first variants such as PN*, PDS, and df-pn work very well, in particular in the domain of shogi mating problems. However, there are still possible drawbacks. The most prevailing one is the double-counting problem. To handle this problem the paper proposes a new search idea using proof number and branching factor as search estimators. We call the new method *Weak Proof-Number Search*. The experiments performed in the domain of shogi and Othello show that the proposed search algorithm is potentially more powerful than the original proof-number search or its depth-first variants.

1 Introduction

In 1994, Allis developed the proof-number search (PN-search) algorithm [2] for finding the game-theoretical value in game trees. PN-search is a best-first search, in which the cost function used in deciding which node to expand next is given by the minimum number of nodes that have to be expanded to prove the goal. As such it is a successor of conspiracy-number search [11,19]. PN-search is appropriate in cases where the goal is a well-defined predicate, such as proving a game to be a first-player win.

PN-search can be a powerful game solver in various simple domains such as connect-four and qubic. Its large disadvantage is that, as a genuine best-first algorithm, it uses a large amount of memory, since the complete search tree has to be kept in memory. To handle the memory disadvantage of PN-search, PN* was proposed [22]. It is a search algorithm for AND/OR tree search, which is a depth-first alternative for PN-search. The idea was derived from Korf's RBFS algorithm [10], which was formulated in the framework of single-agent search.

PN* transforms a best-first PN-search algorithm into an iterative-deepening depth-first approach. The PN* algorithm was implemented in a tsume-shogi (Japanese-chess mating-problem) program, and evaluated by testing it on 295 notoriously difficult tsume-shogi problems (one problem has a depth of search of over 1500 plies). The experimental results were compared with those of other programs. The PN* program showed by far the best results, solving all problems but one.

PDS [13], meaning Proof-number and Disproof-number Search, is a straight extension of PN* which uses only proof numbers. PDS is a depth-first algorithm using both proof numbers and disproof numbers. Therefore, PDS is basically more powerful than PN*. Moreover, Nagai [14] developed df-pn, meaning *depth-first proof-number search*. It behaves similarly to PN-search [15], but is more efficient in its use of memory. It solved all hard tsume-shogi problems quite efficiently.

Since then, df-pn has successfully been applied in other domains such as Go problems [9] and checkers [20,21]. However, we found a serious drawback when applying df-pn in other complex domains such as Othello. The drawback is known as the double-counting problem (see Subsection 2.3). Here we remark that Müller [12] calls it the problem of overestimation.

In this paper, we therefore explore a new idea to improve proof-number-based search algorithms and then propose a new search algorithm called *Weak Proof-Number Search*. We evaluate the new algorithm by testing it on some tsume-shogi problems and Othello endgame positions.

The contents of this paper are as follows. Section 2 presents a brief history of the development of proof-number-based AND/OR-tree search algorithms in the domain of mating search in shogi. Section 3 presents our new idea to improve the PN-search. Experimental performance in the domain of shogi and Othello are shown to evaluate the new search algorithm. Finally, concluding remarks are given in Sect. 4.

2 Proof-Number Based AND/OR-Tree Search Algorithms

Best-first algorithms are successfully transformed into depth-first algorithms, such as PN*, PDS, and df-pn. Each of these algorithms aimed at solving hard tsume-shogi problems [18]. The algorithms can be characterized as variants of proof-number search. Note that PN* only uses proof numbers, while PDS and df-pn use both proof numbers and disproof numbers. In this section, we give a short sketch of proof-number-based AND/OR-tree search algorithms.

2.1 PN-Search

The well-known technique of PN-search was designed for finding the game-theoretical value in game trees [2]. It is based on ideas derived from conspiracy-number search [11] and its variants, such as applied cn-search and $\alpha\beta$-cn search. While in cn-search the purpose is to continue searching until it is unlikely that the minimax value of the root will change, PN-search aims at proving the true value of the root. Therefore, PN-search does not consider interim minimax values. PN-search selects the next node to be expanded using two criteria: (1) the potential range of subtree values and (2) the number of nodes which must conspire to prove or disprove that range of potential values. These two criteria enable PN-search to deal efficiently with game trees with a non-uniform branching factor.

2.2 PN*, PDS, and df-pn

PN* [22] is a depth-first search using a transposition table, and a threshold for the proof numbers. PN* searches in a best-first manner, and uses much less working memory than the standard best-first searches.

Nagai proposed the PDS algorithm, that is, Proof-number and Disproof-number Search, which is a straight extension of PN* [13,14]. This search uses a threshold for the disproof number as well as the proof number when it selects and expands the nodes. The nodes with the smaller proof number or the smaller disproof number are searched first. If the proof number or the disproof number exceeds the threshold in a certain node, PDS stops further searching this node. When PDS fails to expand the root node, it increases one of the two threshold values and restarts the search.

PDS performs multiple iterative deepening in both AND nodes and OR nodes, while PN* does so only in OR nodes. Similarly to PN*, PDS's search behavior is in a best-first manner, while the search basically proceeds depth-first. From this point, PDS could be recognized as a variant of the proof-number search [2]. Actually, PDS uses proof and disproof number asymptotically while PN-search regards them fairly.

Nagai modified PDS and developed a new algorithm named df-pn [16]. The algorithm df-pn first sets the thresholds of both proof and disproof numbers in the root node to a certain large value that can be recognized as infinity. The threshold values are distributed among the descendant nodes. In every node, the multiple iterative deepening is performed in the same way as in PDS. Nagai [15] proved that df-pn search behaves in the same way as PN-search in the meaning that always a most-proving node will be expanded.

2.3 Possible Drawbacks of Proof-Number-Based Search Algorithms

PN-search has at least three possible bottlenecks [2]. The first is memory requirement. The second is Graph-History Interaction (GHI). The third is Directed Acyclic Graphs (DAGs). The answer to the first problem was df-pn, by which it became possible to solve efficiently quite difficult problems such as a shogi-mating problem with 1525 steps.

The GHI problem is a notorious problem that causes game-playing programs to return occasionally incorrect solutions [4]. PN-search and its depth-first variants also have to suffer from it. Breuker *et al.* [3] provided a solution for the best-first search. Later, Kishimoto and Müller [9] showed a practical method to cure the GHI problem for the case of the df-pn search.

A well-known problem of PN-search is that it does not handle transpositions very well. If the search builds a DAG instead of a tree, the same node can be counted more than once, leading to incorrect proof numbers and disproof numbers (i.e., *the double-counting problem*). Thus, PN-search overestimates proof numbers in case where DAGs occur. While there are exact methods for computing the proof numbers for DAGs [12], they are too slow to be practical. For some practical applications, Nagai [15] and Kakinoki [8] proposed a domain-dependent improvement, respectively. However, the DAGs problem is still a critical issue when PN-search is applied in very hard domains.

2.4 AND/OR-Tree Search Taking Branching Factors into Account

A new search idea using the number of possibilities (i.e., branching factors) on a path considered as an estimator for AND/OR-tree search, instead of proof/disproof numbers, was proposed by Okabe [17]. It enables a solver to suffer relatively little from the serious problem due to DAGs. Experimental results show that for some very hard tsume-shogi problems with large DAGs, it outperforms df-pn. However, in most cases df-pn outperforms Okabe's search algorithm (named Branch Number Search or BNS in short).

Okabe [17] shows that in an example graph, threshold $n+1$ or more is needed to solve the graph by BNS, whereas 2^n or more is needed to solve the graph by df-pn. This indicates that for the number of n repeated DAGs a proof-number-based search algorithms suffer exponentially, whereas BNS suffers linearly.

Moreover, df-pn (with Nagai's improvement for DAGs) and BNS were compared in the domain of Othello [23]. It shows that as the frequency of DAGs grows, the performance of df-pn drastically decreases. Indeed, the frequency of DAGs increases as the number of search plies becomes larger in the domain of Othello. In the deeper search, therefore BNS outperforms df-pn in the execution time as well as in the number of search nodes.

3 Weak Proof-Number Search

In this section we propose a new search algorithm using information both on proof numbers and branching factors during search. First we present the basic idea of our proposed search algorithm. Then, the performance of the solver, in which the proposed idea is incorporated, is evaluated in the domain of tsume-shogi and Othello problems.

3.1 The Basic Idea of Our Proposed Search Algorithm

Our proposed idea is similar to PN-search. Hence, the implementation is easy. The only difference is, at an AND node, to use additional information (1) on branching factors and (2) on proof numbers. The information is used as a search indicator. In case where DAGs occur, the new search algorithm would better estimate the correct proof number than PN-search that often overestimates it.

The proposed search indicator, calculated as the maximum of the successor's proof number plus branching factors (except the maximum successor and solved/unsolvable successors) at an AND node, is somehow weaker (while underestimating it) than the proof number defined in PN-search. Therefore, we call it *Weak Proof-Number Search* or WPNS in short. The detail of the WPNS algorithm is shown in Appendix A. Note that procedure $\Phi Max(n)$ is its core part.

We expect WPNS to have two advantages: (1) when compared to proof-number-based search algorithms for relatively simple domains in which the DAGs problem is not so critical and (2) when compared to the BNS algorithm for complex domains in which the DAGs problem occur frequently. Note that we usually have little knowledge about the DAGs issue for unknown target problems. Therefore, such synergy of proof number and branching factor for AND/OR-tree search would enable a program to be an all-round powerful solver.

In the case of a tsume-shogi problem [22], an OR node corresponds to a position in which the attacker is to move, where any move that solves the problem denotes a solution. The proof number then is the minimum proof number of its children (i.e., the one potentially easiest to solve). If the attacker has no more moves in the position, the problem is unsolvable from that position and the proof number is set to ∞.

Likewise, an AND node corresponds to a position with the defender to move. To solve the problem for the attacker all the defender's children must be proven to lead to the desired result, thus its proof number is the sum of the children's proof numbers. If the defender has no more legal moves (is mated), the goal is reached and the proof number is set to 0. However, as mentioned in Subsection 2.3, a serious double-counting problem will happen when large DAGs occur. Therefore, we propose a new search algorithm to use the weak proof number instead of Allis's proof number at AND nodes to avoid such a serious problem.

Let $p(n)$ denote the weak proof number of a node n in an AND/OR tree, and $d(n)$ denote weak disproof number. They are calculated as follows:

1. If n is a leaf node, then
   (a) if n is a terminal node and solved (i.e., OR wins), then
   $p(n) = 0$, $d(n) = \infty$;
   (b) else if n is a terminal and is unsolvable (i.e., OR does not win), then
   $p(n) = \infty$, $d(n) = 0$;
   (c) else n is an unsolved leaf node, then
   $p(n) = 1$, $d(n) = 1$;
2. else if n is an OR node whose successor nodes are $n_i (1 \leq i \leq K)$, then[1]
   $p(n) = \min_{1 \leq i \leq K} p(n_i)$,
   $d(n) = \max_{1 \leq i \leq K} d(n_i) + (k-1);$[2]
3. else if n is an AND node whose successor nodes are $n_i (1 \leq i \leq K)$, then
   $p(n) = \max_{1 \leq i \leq K} p(n_i) + (k-1)$,
   $d(n) = \min_{1 \leq i \leq K} d(n_i)$.

For an easy-to-grasp example, see Fig. 1. In this example, the left-hand choice from the root node takes PN=17 and WPN=9 while the right-hand one takes PN=15 and WPN=11. This means that at the root node PN-search expands first the right-hand move whereas WPNS does the left-hand move. Indeed, the left-hand part is more plausible than the right-hand part in the sense of correct proof numbers. It happens because of a DAG in the left-hand part.

The example indicates that as the branching factor increases, the double-counting problem becomes more serious. We then argue that the performance of

[1] K is the number of successor nodes which do not have *terminal value* such as 0 or ∞.

[2] $(K-1)$ means the number of successor nodes which are not selected.

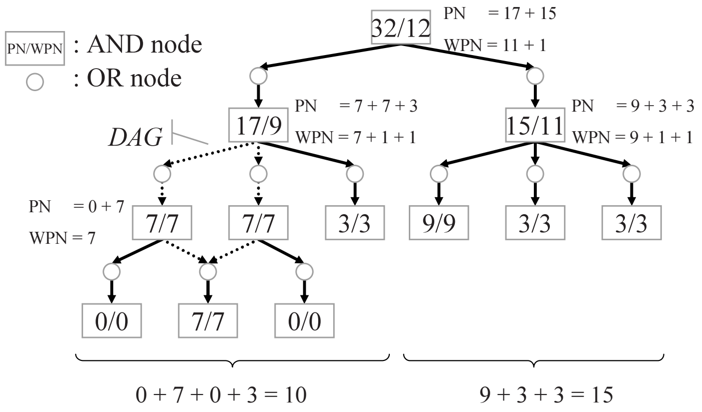

Fig. 1. Proof number (PN) and weak proof number (WPN) compared in an AND/OR tree with a DAG

PN-search decreases when solving a game with higher branching factors. However, the weak proof number is determined by the maximum proof number among all its successors at an AND node, and the number of remaining successors (i.e., branching factor −1) is added. Thus, WPNS relatively suffers little from the influence of DAGs.

3.2 Performance Evaluation

As mentioned in Sect. 2, proof-number-based search algorithms have remarkably been improved in the domain of tsume-shogi. Moreover, the double-counting problem of PN-search or its depth-first variants was found in the domain of Othello. Therefore, it is reasonable to use test sets from the two domains for a performance evaluation of the proposed idea.

WPNS, df-pn, and BNS in the Domain of Tsume-Shogi

In the first experiment, WPNS, df-pn, and BNS were compared in the domain of tsume-shogi. The machine environment was a 3.4 GHz Pentium4 PC running Windows XP and 2,000,000 entries of the transposition table used.

We selected a set of tsume-shogi problems from the book "Zoku-Tsumuya-Tsumazaruya" used as a suite of benchmark problems [7]. It contains 203 numbered problems (200 problems, with 1 problem subdivided into 4 subproblems) from the Edo era to the Showa era, created by 41 composers. The shortest problem is an 11-step problem and the longest one has a solution of 611 steps. The set contains various types of problems. Generally, the book is considered a good benchmark to measure the performance of a tsume-shogi solving program.

WPNS, df-pn, and BNS were implemented in Tacos that is a strong shogi-playing program [6], in which tsume-shogi specific enhancements such as non-pro-motion moves of major pieces are not incorporated. 113 problems were solved

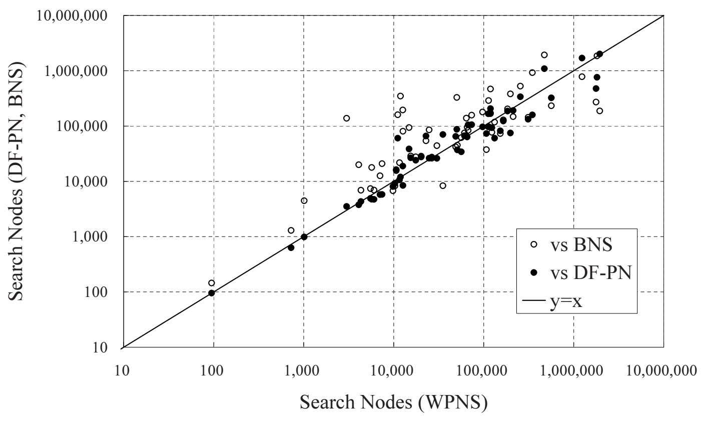

Fig. 2. WPNS, df-pn, and BNS compared on 59 tsume-shogi problems with short solutions

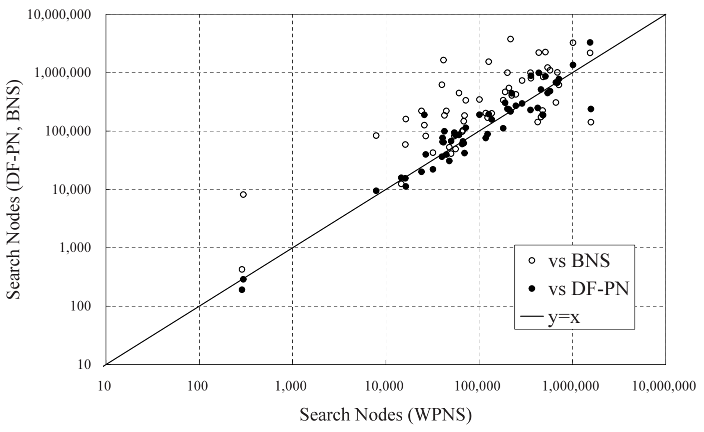

Fig. 3. WPNS, df-pn, and BNS compared on 54 tsume-shogi problems with relatively longer solutions

by each algorithm, whereas df-pn, BNS, and WPNS solved 131, 126, and 123 problems, respectively. We note that the problem with the longest solution was solved only by WPNS. The set of solved problems is categorized into two groups: 59 problems with short solutions (9 to 29 ply) and 54 problems with relatively longer solutions (31 to 79 ply). Let us show, in Fig. 2 and Fig. 3, the experimental results (i.e., the number of search nodes) on the first group and the second group, respectively.

For the first group df-pn outperforms WPNS by a small margin 2%, whereas for the second group WPNS outperforms df-pn by 6%. Moreover, for the first group WPNS outperforms BNS by 38%, whereas for the second group WPNS outperforms BNS by 66%.

WPNS, df-pn, and BNS in the Domain of Othello

In the second experiment, WPNS, df-pn, and BNS were compared in the domain of Othello. The machine environment was a 3.4 GHz Pentium 4 PC running Windows XP and 4,000,000 entries of the transposition table used. We obtained a set of Othello endgame positions through many self-play games using WZEBRA (Gunnar) [5], where each game started with a well-known opening position called *fjt1*. It contains 86 positions. The shortest problem has a 15-ply solution to end and the longest problem has a 20-ply solution.

Let us show, in Fig. 4, the experimental results. The results show that WPNS outperforms BNS by a large margin and is slightly better than df-pn. For 64 positions (75%), WPNS searched fewer nodes than df-pn, whereas for 84 positons (98%) WPNS searched fewer nodes than BNS.

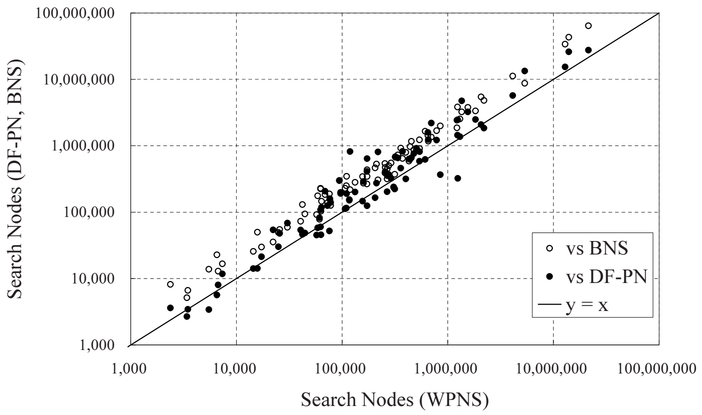

Fig. 4. WPNS, df-pn, and BNS compared on 86 Othello endgame positions

Discussion

From the experiments performed in the domains of tsume-shogi and Othello, we may observe that WPNS basically outperforms df-pn and BNS. In the context of the history of proof-number-based search algorithms, we may now state that df-pn is the most powerful solver in complex domains with some degree such as tsume-shogi problems with long solutions but relatively small branching factors. df-pn drastically decreases its performance in the domain of games with higher branching factors because of DAGs.

It is interesting to know that an average branching factor of tsume-shogi and Othello is 5 [22] and 10 [1], respectively. As shown in Fig. 1, the influence of double-counting problem due to DAGs becomes more serious in solving a game with higher branching factors as well as larger solutions. The experiments performed in the domain of tsume-shogi and Othello support our ideas. Therefore, we claim that WPNS outperforms df-pn when solving complex games with higher branching factors. The reason is that WPNS relatively suffers little from the influence of DAGs and df-pn suffers seriously.

4 Conclusion

Proof-Number Search (PNS) is a powerful AND/OR-tree search algorithm for efficiently solving a hard problem, although it has three possible bottlenecks (memory requirements, the GHI problem, the DAGs). The first two of them have already been improved. The last one is the double-counting problem discussed above. As we noticed in solving Othello endgame positions using df-pn this problem is the most notorious one when using a depth-first variant of PNS.

In this paper, we proposed a new AND/OR-tree search algorithm called Weak Proof-Number Search (WPNS). WPNS is a fruit of the synergy of PN-search and Branch Number Search (BNS). It takes an advantage of proof-number-based search algorithm and avoids the disadvantage of double-counting problem due to DAGs. Experiments performed in the domain of shogi and Othello show that WPNS can be a more powerful solver than df-pn.

References

1. Allis, L.V.: Searching for Solutions in Games and Artificial Intelligence. Ph.D. Thesis, Computer Science Department, Rijksuniversiteit Limburg (1994)
2. Allis, L.V., van der Meulen, M., van den Herik, H.J.: Proof-number Search. Artificial Intelligence 66(1), 91–124 (1994)
3. Breuker, D.M., van den Herik, H.J., Allis, L.V., Uiterwijk, J.W.H.M.: A Solution to the GHI Problem for Best-First Search. In: van den Herik, H.J., Iida, H. (eds.) CG 1998. LNCS, vol. 1558, pp. 25–49. Springer, Heidelberg (1999)
4. Campbell, M.: The graph-history interaction: on ignoring position history. In: 1985 Association for Computing Machinery Annual Conference, pp. 278–280 (1985)
5. Gunnar, A.: ZEBRA, `http://radagast.se/othello/`
6. Hashimoto, J.: Tacos wins Shogi Tournament. ICGA Journal 30(3), 164 (2007)
7. K. Kadowaki. Zoku-Tsumuya-Tsumazaruya, Shogi-Muso, Shogi-Zuko. Heibon-Sha, Toyo-Bunko, (1975) (in Japanese)
8. Kakinoki, Y.: A solution for the double-counting problem in shogi endgame. Technical report (2005) (in Japanese), `http://homepage2.nifty.com/kakinoki_y/free/DoubleCount.pdf`
9. Kishimoto, A., Müller, M.: Df-pn in Go: An Application to the One-Eye Problem. In: Advances in Computer Games 10, pp. 125–141. Kluwer Academic Publishers, Dordrecht (2003)
10. Korf, R.E.: Linear-space best-first search. Artificial Intelligence 62(1), 41–78 (1993)
11. McAllester, D.A.: Conspiracy numbers for min-max search. Artificial Intelligence 35(3), 287–310 (1988)
12. Müller, M.: Proof-Set Search. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 88–107. Springer, Heidelberg (2003)
13. Nagai, A.: A new AND/OR tree search algorithm using proof number and disproof number. In: Proceedings of Complex Games Lab Workshop, pp. 40–45. ETL, Tsukuba (1998)
14. Nagai, A.: A new depth-first-search algorithm for AND/OR trees. M.Sc. Thesis, Department of Information Science, The University of Tokyo, Japan (1999)
15. Nagai, A.: Proof for the equivalence between some best-first algorithms and depth-first algorithms for AND/OR trees. In: Proceedings of Korea-Japan Joint Workshop on Algorithms and Computation, pp. 163–170 (1999)

16. Nagai, A., Imai, H.: Application of df-pn+ to Othello Endgames. In: Game Programming Workshop 1999, Hakone, Japan (1999)
17. Okabe, F.: About the Shogi problem solution figure using the number of course part branches. In: 10th Game Programming Workshop, Hakone, Japan (2005) (in Japanese)
18. Sakuta, M., Iida, H.: AND/OR-tree search algorithms in shogi mating search. ICGA Journal 24(4), 231–235 (2001)
19. Schaeffer, J.: Conspiracy numbers. In: Beal, D.F. (ed.) Advances in Computer Chess, vol. 5, pp. 199–218. Elsevier Science, Amsterdam (1989); Artificial Intelligence, 43(1):67-84 (1990)
20. Schaeffer, J., Björnsson, Y., Burch, N., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers Is Solved. Science 317(5844), 1518–1522 (2007)
21. Schaeffer, J.: Game Over: Black to Play and Draw in Checkers. ICGA Journal 30(4), 187–197 (2007)
22. Seo, M., Iida, H., Uiterwijk, J.W.H.M.: The PN*-search algorithm: Application to tsume-shogi. Artificial Intelligence 129(4), 253–277 (2001)
23. Ueda, T., Hashimoto, T., Hashimoto, J.: Solving an Opening Book of Othello and Consideration of Problem. In: 12th Game Programming Workshop, Hakone, Japan (2007) (in Japanese)

Appendix A

The C++ like pseudo-code of Depth-First Weak Proof-Number Search (DF-WPN) algorithm is given below. For ease of comparison we use similar pseudo-code as given in [14] for the df-pn algorithm. DF-WPN is similar to df-pn. The only difference is, at an AND node, to use additional information on branching factors as well as proof numbers, which appears in line 78.

Below code ϕ and δ are used instead of $WPN(n)$ and $WDN(n)$, just as α and β behave differently in the negamax algorithm compared to classical $\alpha\beta$ algorithm. These are defined as follows:

- $\phi = \begin{cases} WPN(n) & \text{if } n \text{ is OR node} \\ WDN(n) & \text{otherwise,} \end{cases}$
- $\delta = \begin{cases} WDN(n) & \text{if } n \text{ is OR node} \\ WPN(n) & \text{otherwise.} \end{cases}$

```
 1 void df-wpn(root) {
 2     root.th$\phi$ = $\infty$; root.th$\delta$ = $\infty$;
 3     multiID(root);
 4 }
 5
 6 void multiID(n) {
 7     // 1. look up transposition table
 8     retrieve(n, $\phi$, $\delta$);
 9     if ( n.th$\phi$ $\le$ $\phi$ || n.th$\delta$ $\le$ $\delta$ ) {
10         return;
11     }
12     // 2. generate legal moves
```

```
    if ( is_terminal(n) ) {
        if (( is_AND(n) && evaluate(n) = true )||
            ( is_OR(n) && evaluate(n) = false )) {
            store(n, $\infty$, 0); // cannot prove or disprove anymore
        } else {
            store(n, 0, $\infty$);
        }
        return;
    }
    generate_moves();
    // 3. use transposition table to avoide cycle
    store(n, $\phi$, $\delta$);
    // 4. multiple iterative deepening
    while (true) {
        // stop if $\phi$ or $\delta$ is greater or equal to its threshold
        $\phi$ = $\Delta$Min(n);
        $\delta$ = $\Phi$Max(n);
        if ( n.th$\phi \leq \phi$ || n.th$\delta \leq \delta$ ) {
            store(n, $\phi$, $\delta$);
            return;
        }
        child = select(n, $\phi_c$, $\delta_c$, $\delta_2$)
        child.th$\phi$ = n.th$\delta$ + $\phi_c$ - $\delta$;
        child.th$\delta$ = min(n.th$\phi$, $\delta_2$ + 1);
        multiID(child);
    }
}
// select the most proving child node
NODE select(n, &$\phi_c$, &$\delta_c$, &$\delta_2$) {
    $\delta_c$ = $\infty$; $\delta_2$ = $\infty$;
    for ( each child node c ) {
        retrieve(c, $\phi$, $\delta$);
        if ( $\delta < \delta_c$ ) {
            best = c;
            $\delta_2$ = $\delta_c$; $\phi_c$ = $\phi$; $\delta_c$ = $\delta$;
        } else if ( $\delta < \delta_2$ )
            $\delta_2$ = $\delta$;
        if ( $\phi = \infty$ )
            return best;
    }
    return best;
}
// retrieve numbers from transposition table
void retrieve(n, &$\phi$, &$\delta$) {
    if ( n is already recorded ) {
        $\phi$ = Table[n].$\phi$; $\delta$ = Table[n].$\delta$;
    } else {
        $\phi$ = 1; $\delta$ = 1;
    }
}
```

```
// store numbers to transposition table
void store(n, $\phi$, $\delta$) {
    Table[n].$\phi$ = $\phi$; Table[n].$\delta$ = $\delta$;
}
// calculate minimum $\delta$ of the successors (same as df-pn)
unsigned int $\Delta$Min(node) {
    min$\delta$ = $\infty$;
    for ( each child node c ) {
        retrieve(c, $\phi$, $\delta$);
        min$\delta$ = min(min$\delta$, $\delta$);
    }
    return min;
}
// calculate weak proof/disproof number
// df-pn uses $\Phi$Sum(n) instead of this function
unsigned int $\Phi$Max(n) {
    max$\phi$ = 0;
    for ( each child node c ) {
        retrieve(c, $\phi$, $\delta$);
        max$\phi$ = max(max$\phi$, $\phi$);
    }
    return ( max$\phi$ + n.ChildNodeNum - 1 );
}
```