THE PERFORMANCE OF PN*, PDS, AND PN SEARCH ON 6x6 OTHELLO AND TSUME-SHOGI

M. Sakuta and H. Iida[1]

Shizuoka University
Hamamatsu, Japan

ABSTRACT

Best-first search algorithms such as proof-number search work well when solving endgame positions in chess-like games. However, it is hard, even for proof-number (PN) search, to solve an endgame problem with very long sequences because of its inherent memory limits.

After PN search was developed and had achieved many successes for solving some hard problems, innovations to depth-first search algorithms were based on the idea of proof numbers. This article introduces two such innovative algorithms: PN* and PDS. The PN* algorithm was developed for solving Tsume-Shogi problems, among which one with more than 1500 steps. Then, PDS was proposed to strengthen the power of the PN* algorithm.

The current contribution presents an empirical comparison of PN*, PDS, and PN search, as well as some variants of depth-first search on the domains of 6x6 Othello and Tsume-Shogi. The comparison shows that PDS outperforms PN* on the two testbeds. In 6x6 Othello, the PN-search variant does not show its characteristic remarkable advantage over the usual depth-first search algorithms. In Tsume-Shogi, forcing move sequences and sudden terminations play an important part when solving such problems; therefore PN-search variants, in particular PDS, show the expected remarkable advantage over other depth-first searches.

1. INTRODUCTION

Problem solving is one of the main themes in artificial intelligence. Generally, a problem is formulated into an AND/OR graph. So one of the subjects in AI is to solve an AND/OR graph effectively. Moreover, this is an essential requirement of solving large puzzles. It is closely connected to searching a game tree. A game tree for endgame positions of most two-player zero-sum games can be recognized as an AND/OR tree with considering the occurrences of transpositions additionally. Best-first search algorithms, such as AO* (Nilsson, 1980), have been used in the AND/OR-graph (or -tree) search. The main characteristic of these methods is that they expand the tree in memory while searching.

In 1995, Seo developed a new depth-first search algorithm based on proof numbers, which behaves like best-first search. It is based on Korf's IDA* (1985) or RBFS (1993) algorithm. Seo (1995) originally named it C*, but it was later dubbed PN* (Seo, Iida, and Uiterwijk, 2001). PN*'s most interesting feature is that it does not expand the whole search tree in memory. So it requires much less memory in view of the usual best-first algorithms. Seo's solver showed a marvelous power by solving Tsume-Shogi problems with very long sequences and in 1997 it solved the longest-sequence problem (1525 steps).

Recently, Nagai (1998; 1999) developed PDS, a novel depth-first search algorithm as an extension of the PN* algorithm. However, the new algorithm as well as PN* have not yet been tested worldwide. In this paper, we make a start with such extended testing and show the results of the two algorithms on two different types of games: 6x6 Othello and Tsume-Shogi.

[1]Department of Computer Science, Shizuoka University, 3-5-1 Johoku, Hamamatsu, 432-8011 Japan. E-mail: {sakuta,iida}@cs.inf.shizuoka.ac.jp

The course of the article is as follows. In Section 2 we briefly describe three search algorithms, viz. PN search, PN* and PDS. Then, in Section 3 we compare the results of the three algorithms when applied in the 6x6 Othello domain. In Section 4 we repeat the comparison for the domain of Tsume-Shogi. Section 5 provides conclusions and suggests some future work.

2. THREE ALGORITHMS

In this section we introduce PN search, PN* and PDS. Since PN search is well-known we will rely on an adequate summary published by Breuker (1998).

PN* and PDS both stem from conspiracy-number search (CN search) (McAllester, 1988; Schaeffer, 1990) and proof-number search (PN search) (Allis, van der Meulen, and van den Herik, 1994b; Allis, 1994a). In CN search, the concept of a conspiracy number was introduced in the game-tree search. This search was primarily meant for the minimax-tree search.

2.1 PN Search

Below we briefly describe the idea of PN search. For this we quote a suitable resume by Breuker (1998). "Proof-number search is a best-first AND/OR tree-search algorithm, and is inspired by the conspiracy-number algorithm (McAllester, 1988; Schaeffer, 1990). Before starting the search, a search goal is defined (e.g., try to reach at least a draw). The evaluation of a node returns one of three values: *true*, *false*, or *unknown*. The evaluation is seen from the point of view of the player to move in the root position. The value *true* indicates that the player to move in the root position can achieve the goal, while *false* indicates that the goal is unreachable. A node is *proved* if its value has been established to be *true*, whereas the node is *disproved* if its value has been determined to be *false*. A node is *solved* as soon as it has been proved or disproved. A tree is solved (proved or disproved) if its root is solved. The goal of pn search is to solve a tree."

This search has a high ability for solving AND/OR trees; however, it expands the whole search tree in working memory and requires much memory. This means that there is a severe memory restriction for hard problems. Some variations have been proposed for modification with success (cf. Breuker, 1998; Breuker, Uiterwijk, and van den Herik, 2001). In our experiments, we have not implemented their recommendations.

2.2 PN* Algorithm

PN* is a recursive iterative-deepening depth-first search algorithm using proof numbers as a criterion to develop the frontier nodes (Seo *et al.*, 2001). PN* starts searching by setting the proof-number threshold to 1 and expands the root node. If the proof number of the node under consideration exceeds the threshold, PN* stops node expansion at this node. When all frontier nodes have proof numbers exceeding the proof-number threshold, the search tree is discarded, the threshold of the root is incremented and a new search starts. This iterative searching is enhanced by storing the expanded nodes and their properties in the transposition table. As the threshold is getting larger, node expansion proceeds in such a way that nodes having relatively small proof numbers are expanded first, like in best-first search, without the disadvantage of the memory requirement.

PN* also uses the method of multiple iterative deepening. At an AND node the threshold values assigned to the child OR nodes are iteratively increased from 1 to their current maximum.

2.3 PDS Algorithm

The PDS (Proof-number and Disproof-number Search) algorithm is a straight extension of PN*. Basically, searching proceeds like PN* except that PDS uses both proof numbers and disproof numbers. It uses two thresholds in searching, the proof-number threshold and the disproof-number threshold. If the proof number or the disproof number exceeds the threshold at a certain node, PDS stops node expansion at this node. When PDS fails the root node expansion it increases the either threshold value and restarts the searching. Multiple
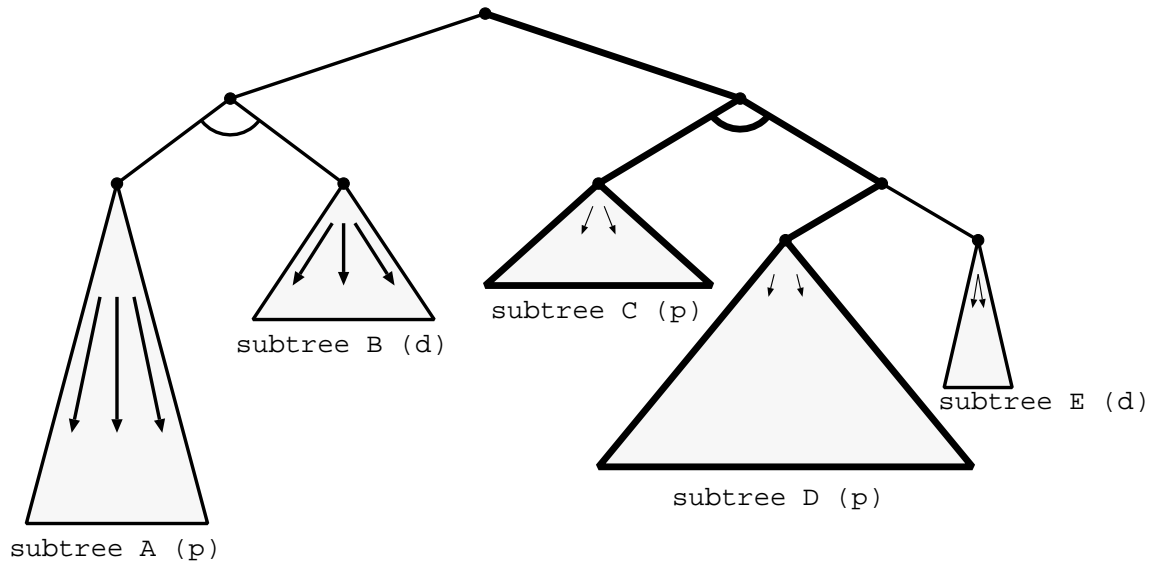
Figure 1: Example search tree.

iterative deepening is also applied in PDS, but differs from PN* in that it performs multiple iterative deepening at both AND and OR nodes.

PN* and PDS are depth-first searches but behave like best-first searches.

Figure 1 schematically shows a typical example where PN* works inefficiently and PDS does not. In the figure, **(p)** means that the subtree is to be proved and **(d)** that it is to be disproved. In the figure, the parts denoted by a thick line belong to the solution tree. In searching, PN* gives precedence to the first plausible subtree A. Since subtree A is rather narrow and hence has small proof numbers, it is searched to a considerable depth before the win in the right-hand branch is discovered. PDS searches this example tree much more efficiently. Using also disproof numbers it quickly discovers that the root of subtree B is insolvable and hence the parent of subtrees A and B is insolvable. From then on it will search solely in the right-hand branch of the tree and will thus discover the solution much faster than PN*.

3. COMPARISON ON 6×6 OTHELLO

Experiments have been performed on 6x6 Othello to compare the search ability of three algorithms, PN*, PDS and PN search. For reference purposes, the experiments are also executed with the help of PVS (principal-variation search), and other alpha-beta variants.

3.1 6x6 Othello

6x6 Othello is a smaller family member of the 8x8 Othello game. Feinstein (1993) reported that by optimal play from both sides the game ends in a win for the second player (White) by 4 discs (16-20) after 33 plies on the principal variation (see Appendix A).

3.2 Experimental Design

The 6x6 Othello positions are coded in 64-bit integers using the available perfect information. We employed the double hashing method; the codes are effected as the first indexes via a hashing function. The number of entries of the transposition table is 16 million and the size of the transposition table is about 200MB as used in PN* and PDS. We have neither implemented the replacement of unimportant elements by more important elements, nor the garbage collection for the transposition table. In PN search, the code of a position is stored

in the corresponding node and it is decoded to the position when expanding the node. Thus the memory requirement per node is small. The number of nodes in heap memory is 15 million and the total size of the heap is about 370MB for PN search.

At first, we hoped to solve the whole tree, but unfortunately we could not succeed in solving the start position or the position after the first move. So we selected intermediate positions on the principal variation (PV) and tried to solve them, i.e., finding a winning line for the second player. The moves on the PV investigated and the tested positions are shown in Appendix A. Since the terminal node values in the AND/OR tree under investigation are only either true or false, we have to assign the terminal node values as true or false in a certain criterion. For proving, we tested two different criteria, i.e., White winning by 1 disc or White winning by 4 discs. The first criterion means if White wins by one or more discs the result is recognized as true, otherwise it is recognized as false. The second criterion means if White wins by four or more discs the result is recognized as true, otherwise it is recognized as false. The first case is a little easier to solve since it suffices to find an **arbitrary** win, whereas the second criterion succeeds when an **optimal** win is found (since in the optimal line White wins by four discs). At every position, we have compared three algorithms, PN*, PDS and PN search. For reference purposes, the experiments are also performed for PVS (principal-variation search), an alpha-beta variant. In addition, we have examined the disproving cases. For disproving, we set the criterion of winning by 5 discs. As mentioned above, there is no solution and the search is to be disproved.

The solving time was measured by seconds using a Pentium II 450 MHz computer with 384 MB RAM under Windows 98.

The classifications of the terminal nodes in the whole game tree from the selected positions are obtained by the program using the full-width search with no cutoffs.

ply	vs	terminal nodes	Black's win		Draw		White's win		White wins by 4 discs or more	
25	7	305	200	(65.6)	14	(4.6)	91	(29.8)	70	(23.0)
23	9	3381	2336	(69.1)	242	(7.2)	803	(23.8)	617	(18.2)
21	11	119923	75716	(63.1)	7938	(6.6)	36269	(30.2)	27834	(23.2)
19	13	3619363	1959233	(54.1)	206417	(5.7)	1453713	(40.2)	1225048	(33.8)
17	15	251184784	109753338	(43.7)	14934037	(5.9)	126497409	(50.4)	110580515	(44.0)
15	17	20200398479	11147001000	(55.2)	1257191715	(6.2)	7796205764	(38.6)	6552920862	(32.4)

Table 1: Classification of terminal nodes of the game tree from n-ply positions on PV in 6x6 Othello.
ply: n-th ply position on PV, vs: number of vacant squares. The numbers in parentheses indicate the percentages.

3.3 Results and Discussions

Table 1 lists the statistics of the complete tree originating from the selected positions. The opportunities of winning by the first player and by the second player are roughly the same as the vacant squares increase. Experimental results for proving the value of the positions are listed in the Tables 2 and 3. The node count is the number of nodes registered into the transposition table (for PN* and PDS) and it is the number of nodes expanded into working-memory (for PN search). For reference, Table 4 gives the experimental results of PVS. The node count is the number of visited nodes in PVS.

There is no remarkable difference between the results of the criterion of White's winning by 1 and those of the criterion of White's winning by 4. So, we focus on the result of the criterion of White's winning by 4. The relation between vacant squares and the common logarithm of the node count is plotted in Figure 2, and the relation between vacant squares and the common logarithm of solving time is plotted in Figure 3.

Only the alpha-beta PVS can get the data at the position with 23 vacant squares. However, PVS only finds the principal variation at the given position, not giving the solution tree.

In Figure 2 we can see that PDS expands the least nodes when solving the position under investigation, PN* the second least, and PN search and alpha-beta expand the most nodes.

In Figure 3, PN search and PDS are roughly as fast as PVS and are faster than PN* in general.

Table 5 lists the experimental results for disproving. PN search is the fastest for the positions that are solvable in relatively few steps, but expands the most nodes for solving among the algorithms tested. However, there

	PN*				PDS				PN search			
			Solved Tree				Solved Tree				Solved Tree	
vs	tt count	time	size	md	tt count	time	size	md	nd count	time	size	md
3	7	8.7E-4	5	4	7	6.5E-4	5	4	12	2.3E-4	5	4
5	11	9.9E-4	7	6	9	8.3E-4	7	6	16	2.8E-4	7	6
7	32	1.87E-2	16	8	23	2.2E-3	16	8	50	6.8E-4	16	8
9	128	2.96E-2	84	11	100	1.20E-2	84	11	378	5.1E-3	85	11
11	695	8.07E-2	236	14	547	7.16E-2	236	14	2472	2.92E-2	233	13
13	3180	0.370	561	17	2674	0.360	559	17	9779	0.110	520	17
15	27948	2.444	2439	20	6530	0.878	2320	20	34394	0.366	2178	19
17	123532	8.429	5492	23	23874	2.555	5494	23	223738	2.325	4588	21
19	536875	32.50	44201	27	322605	25.26	55247	27	3050725	30.33	35267	25
21	3305531	195.42	138284	31	850351	67.02	100791	31	13114201	127.10	83487	29

Table 2: Solving 6x6 Othello PV positions (win criterion = 1).

vs: number of vacant squares, tt count: number of nodes in the transposition table, nd count: number of nodes expanded, md: maximum depth of the solution tree. These are also used in Table 3 and 5.

	PN*				PDS				PN search			
			Solved Tree				Solved Tree				Solved Tree	
vs	tt count	time	size	md	tt count	time	size	md	nd count	time	size	md
3	7	6.7E-4	5	4	7	6.9E-4	5	4	12	2.3E-4	5	4
5	11	1.0E-3	7	6	9	8.1E-4	7	6	16	2.8E-4	7	6
7	32	1.09E-2	16	8	23	2.2E-3	16	8	50	6.8E-4	16	8
9	128	1.43E-2	84	11	100	1.23E-2	84	11	378	5.0E-3	85	11
11	685	7.86E-2	236	13	579	7.32E-2	236	13	2495	2.97E-2	235	13
13	3184	0.370	557	17	2664	0.360	559	17	10067	0.113	522	17
15	27325	2.407	2469	20	7457	0.963	2347	20	35449	0.380	2204	19
17	108120	7.686	5687	23	27798	2.867	5574	23	226205	2.362	4728	21
19	569874	34.71	44241	27	322709	24.65	45002	27	3384774	33.88	37904	25
21	4937376	300.22	120276	31	885778	67.03	116046	31	14134470	136.98	88349	29

Table 3: Solving 6x6 Othello PV positions (win criterion = 4).

vs	node count	time	depth
3	15	1.4E-4	4
5	29	3.7E-4	6
7	92	1.2E-3	8
9	300	4.1E-3	10
11	1037	1.46E-2	12
13	16375	0.219	14
15	109329	1.395	16
17	232713	3.062	18
19	3.24E+6	43.50	20
21	1.25E+7	166.92	22
23	2.54E+8	3371.11	24

Table 4: Searching PV sequence by depth-first alpha-beta PVS at 6x6 Othello PV positions.
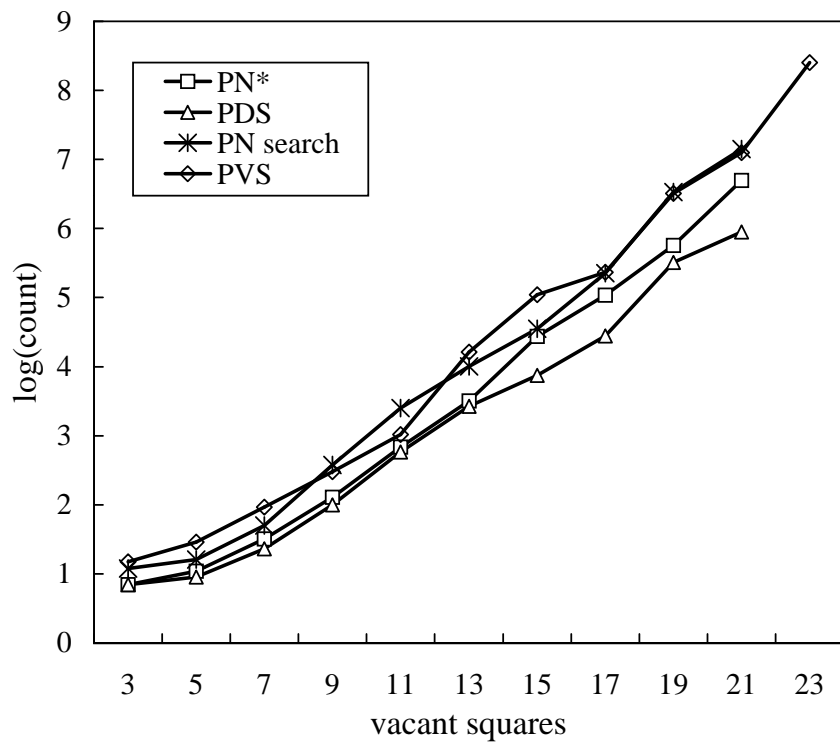vs: number of vacant squares, depth: length of the PV sequence

Figure 2: Number of nodes in the transposition table in solving 6x6 Othello as a function of the number of the remaining vacant squares (win criterion = 4).
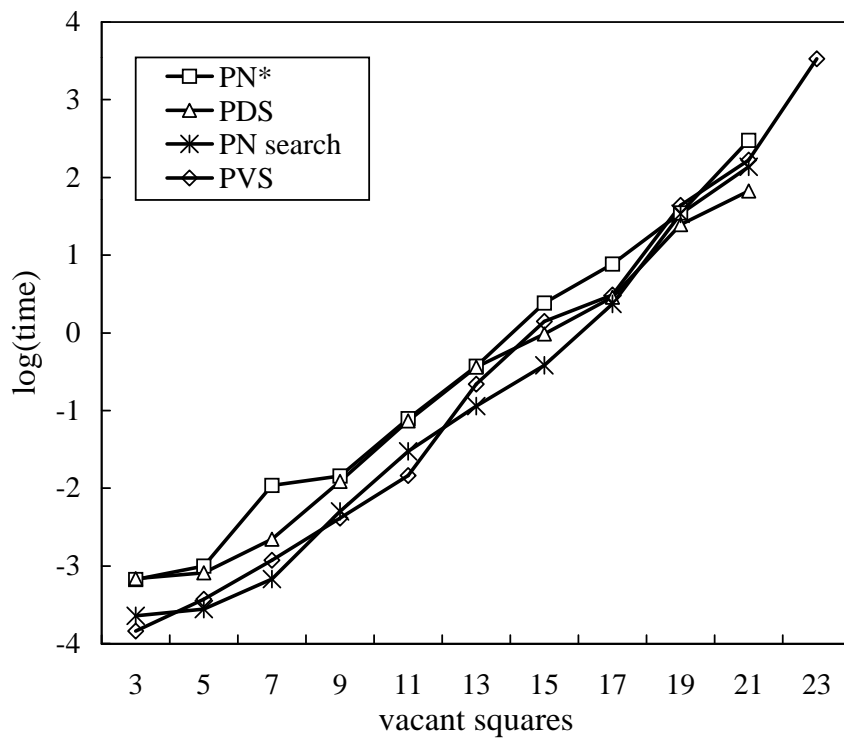

Figure 3: Solving time of 6x6 Othello positions on PV as a function of the number of the remaining vacant squares (win criterion = 4).

is no significant difference between PN* and PDS in the solving time. There is also no significant difference between PN* and PDS in the node count except for the longest-step position, which only PDS can solve.

	PN*				PDS				PN search			
			Solved Tree				Solved Tree				Solved Tree	
vs	tt count	time	size	md	tt count	time	size	md	nd count	time	size	md
3	13	9.5E-4	13	4	13	8.6E-4	13	4	13	2.5E-4	13	4
5	27	3.1E-3	27	7	27	1.9E-3	27	7	37	6.4E-4	27	7
7	75	9.7E-3	81	9	99	9.3E-3	81	9	165	2.4E-3	82	10
9	140	1.91E-2	119	11	202	2.04E-2	119	11	411	5.5E-3	114	12
11	472	6.82E-2	457	14	775	9.75E-2	347	13	2839	3.44E-2	343	14
13	4268	0.513	2067	18	5890	0.675	1849	18	20158	0.243	1577	17
15	30708	2.720	4787	20	26973	2.715	4697	20	156949	1.653	3275	19
17	100132	7.307	25878	22	98478	7.847	14700	22	652769	6.497	10642	22
19	723942	44.31	77520	26	343001	25.73	24283	24	2887320	28.58	20552	26
21	4777343	292.31	526228	28	2928420	223.60	109755	28	28338861	307.41	90549	28
23					6176601	552.27	314670	30				

Table 5: Disproving 6x6 Othello PV positions (win criterion = 5).

Here is a summary of the results on 6x6 Othello. As for the number of elements of the transposition table there seems to be the tendency that PDS surpasses PN* by a factor from 2 to 5 as the searching space is increasing. This seems to be caused by the fact that in 6x6 Othello the first player and the second player are nearly even in the opening, i.e., the number of winning positions for each player is roughly comparable, therefore PDS can perform the disproving ability as well as the proving one, while PN* can not.

The severe memory restriction in the current implementation with respect to solving trees results in PDS having a better solving ability than PN*. PN search is one of the fastest algorithms but has a severe restriction of working memory for positions that have a long-sequence solution. The alpha-beta variant PVS shows the high solving ability and speed. All searches using proof numbers, such as PN search, PN* and PDS, work well when the sequences of forcing moves and sudden terminations play an important part in searching. However, in the game of Othello, there probably are more non-forcing good moves than forcing moves and the ratio of the sudden termination is quite small. In this case, in view of finding a winning move in a certain position, standard depth-first search, i.e., alpha-beta variants have the predominance over PN-search variants.

4. COMPARISON ON TSUME-SHOGI

To compare the search ability of three algorithms, PN*, PDS and PN search, experiments have also been performed on Tsume-Shogi. For reference purposes, the experiments are also performed by iterative deepening variants of depth-first search. The standard depth-first search is impractical for Tsume-Shogi because most problems need a fairly long sequence and cannot be solved in practical time.

4.1 Tsume-Shogi

Tsume-Shogi is a puzzle that can be seen as a variety of Shogi (Japanese Chess). Its rules are roughly as follows:

1. The attacking side (usually Black) is to move first.
2. The attacking side must play check move sequences.
3. The attacking side must select the check moves in such a way that they lead to checkmate as soon as possible.
4. The defending side must defend against the check moves so as to prolong the checkmate as much as possible.
5. All other rules are the same as in Shogi.

In this contribution we do not obey rule 3 in every respect, i.e. a longer sequence than necessary is now and then allowed. There are a whole lot of easy Tsume-Shogi puzzles for beginners or intermediate players. But there are many hard problems for the real Tsume-Shogi enthusiasts. Among them there are some famous and very difficult problems. The longest and hardest problem at present, named "Microcosmos", takes 1525 steps. It was solved in 1997 by the Seo's Tsume-Shogi solver based upon PN* algorithm (Seo *et al.*, 2001).

4.2   Experimental Design

The board positions are coded into 64-bit integers by the Zobrist (1970) method. In addition, there are reusable pieces in hand in Shogi. The pieces in hand of the first player are coded with perfect information into 32-bit integers. Positions are recognized as identical when both the code of the board and the code of the pieces are same. We used the double hashing method; the low-bits of the board codes are used as the first indexes of the transposition table and the high-bits as the second. The number of entries of the transposition table is 8 million and the size of the transposition table is about 200 MB for PN*, PDS and iterative deepening. We have not implemented the replacement of unimportant elements by more important elements, nor the garbage collecting for the transposition table.

In PN search, the essential information of a position is stored in the corresponding node and it is converted to the position with additional information when expanding the node. In our implementation, the essential information of a position requires 138 bytes, so the memory requirement per node is not so small. The number of nodes in heap memory is 1.5 million and the total size of the heap is about 240 MB for PN search.

In Shogi the defending side can make a meaningless move dropping a piece to the intermediate square between the King and the attacking piece. In addition, there is a dominance relation between two positions when the boards are the same and the set of pieces in hand of a certain side in one position is a superset/subset of that in the other position. Although the efficiency and ability of searching could be much higher by omitting the meaningless moves and considering the dominance relation, we did not implement them. In addition, our solver does not necessarily find the shortest solution, in some cases it finds a longer sequence as solution containing some redundant roundabout moves.

While the common Tsume-Shogi problems are composed apart from real Shogi games, we selected some comparatively easy 30 positions from the endgames of Shogi among grandmaster players (Hiura, 1996) and tried to solve them. There are some easy problems of 5 to 7 steps and some hard problems of over 30 steps. All problems should end in a checkmate according to the source, but one of them turned out not to result in a checkmate by any means, that is, the problem was disproved as a problem. This has been found by PDS search. In Appendix B, we show the tested problems.

4.3   Results and Discussion

The experimental results are listed in Table 6 and 7. The node count is the number of nodes registered into the transposition table in PN*, PDS, and iterative deepening, and is the number of nodes expanded into working-memory in PN search. Problem #2 has no checkmate by any means; this was found by PDS.

PDS could solve all the problems (proved 29 problems and disproved 1 problem), whereas PN* could solve 25 among 30. PDS shows the highest solving ability as well as disproving ability, among four algorithms. PDS can prove one problem with 57 steps and can disprove another problem with 72 steps. For problems both solved by PN* and PDS the solving speed of PDS is higher than that of PN* in many cases. Moreover, the number of nodes in the transposition table of PN* is sometimes over 10 times larger than that of PDS, and PN* could not solve the problems with relatively large steps. Therefore, PDS surpasses PN* in many problems used with regard to solving. However, Seo's (1995) Tsume-Shogi solver has a much higher solving ability when omitting the meaningless moves and considering the dominance relation and other domain-specific features. In addition, the problems tested were chosen from endgame positions of Shogi games, therefore the results may differ from other publications since there may be an essential difference between the characteristics of the game tree in Tsume-Shogi problems and that in endgame positions of Shogi games.

Problem #30 is the most difficult problem that only PDS could prove. The maximum depth of the solution tree found is 57; this shows the power of PDS. But the solution includes many roundabout moves. Problem #2 is

		PN*					PDS			
				Solved Tree					Solved Tree	
No.	S/U	tt count	time	size	md	S/U	tt count	time	size	md
1		4212239	671.	528	23		62866	10.4	414	23
2	X					D	4634026	720.	6402307	72
3		393674	53.0	120	17		18057	3.12	130	17
4		46209	7.38	66	11		29943	5.04	98	13
5		300	0.074	20	5		495	0.122	20	5
6		4488	0.981	54	13		2896	0.635	58	13
7		1135	0.225	24	15		325	0.056	24	15
8		429391	64.4	204	19		83512	12.3	242	21
9		19720	4.87	228	17		32336	6.11	254	23
10		7129960	1000.	1664	37		5289355	757.	2504	49
11		561	0.107	26	17		158	0.031	26	17
12	X						3627412	568.	4302	35
13		20567	4.00	38	13		6862	1.48	38	13
14		6797	1.42	34	7		5053	1.09	38	9
15	X						1821443	236.	2424	41
16		178381	25.0	340	19		28744	4.45	262	23
17		13151	2.68	150	17		6355	1.07	154	17
18	X						5025454	832.	2606	39
19		27044	6.51	406	19		13046	2.55	794	31
20		3942699	648.	1108	25		3842257	574.	1546	31
21		1006945	175.	1850	29		176759	29.3	1160	29
22		455033	60.6	566	19		159226	21.8	552	23
23		1055634	208.	510	21		314574	52.8	712	27
24		4150	0.832	40	13		2623	0.464	40	13
25		367876	57.8	350	23		63574	9.42	340	23
26		4256	0.994	26	13		1228	0.284	26	13
27		55737	9.02	176	19		31026	4.99	218	19
28		4281814	722.	1990	35		397399	54.4	816	31
29		192251	35.1	380	27		38023	6.94	444	35
30	X						4708956	719.	11694	57
		Proved:25					Proved:29, Disproved:1			

Table 6: Results of solving Tsume-Shogi problems (PN* and PDS).
'X' in the column 'S/U' means the problem could not be solved and 'D' means the problem was disproved.
tt count: number of nodes in the transposition table, md: maximum depth of the solution tree

		PN search					depth-first iterative deepening				
				Solved Tree					Solved Tree		
No.	S/U	nd count	time	size	md	S/U	tt count	time	size	md	sold
1		470901	15.6	402	23	X					
2	X					X					
3		248212	7.76	120	17		3271554	603.	120	17	
4		546201	18.5	72	13		29831	4.98	64	11	
5	X						88	0.016	20	5	
6		7782	0.253	54	13		71894	12.1	54	13	
7		1195	0.044	24	15		7773	1.29	24	13	
8		603134	18.8	270	27	X					
9		227163	7.93	290	35		76680	22.6	362	17	
10	X					X					
11		1130	0.038	34	17		1203	0.191	26	17	
12	X					X					
13		24050	0.786	38	13		92896	19.6	38	13	
14		18758	0.676	32	7		2010	0.432	32	7	
15	X					X					
16		267353	9.73	248	25		145384	23.5	274	19	(15)
17		49692	1.68	148	19		36057	6.12	236	17	
18	X					X					
19		100580	3.92	504	27		43518	11.2	374	19	(17)
20	X						1022076	159.	776	17	(15)
21	X					X					
22		1269687	40.5	568	23		3935429	734.	1352	23	(19)
23		1571678	56.2	596	35		7985853	1700.	12576	25	(21)
24		7654	0.271	40	13		10522	1.82	40	13	
25		474852	15.5	348	25	X					
26		4177	0.148	26	13		119357	33.9	26	13	
27	X						480520	99.8	208	19	
28	X						1402273	340.	2940	25	(21)
29		308424	10.5	474	31	X					
30	X					X					
		Proved:19					Proved:19				

Table 7: Results of solving Tsume-Shogi problems (PN search and iterative deepening).

nd count: number of nodes expanded, tt count: number of nodes in the transposition table, md: maximum depth of the solution tree, sold: solved search depth.

In iterative deepening, if the iteration depth is different from the depth of the solution tree, it is shown in parentheses in the column 'sold'.

the most difficult problem that only PDS could disprove.

PN search solved 19 problems. This algorithm is fast but cannot solve some easy problems. Problem #5 is a problem that PN search could not solve, whereas the depth of the solution tree is only 5 and this problem seems to be fairly easy for human players. This is because there are many promising moves at the beginning position and the algorithm used up most of the elements of the heap memory searching in the wrong subtrees. However, PN search can solve some problems over 25 steps. Moreover, it can be possible to reduce the memory requirement per node to one third or less by packing the information of a position, then the ability of solving could be twice or thrice as high. Finally, it is remarked that by the application of Breuker's (1998) techniques, we expect to solve many more problems with PN search.

Iterative deepening solved 19 problems. It could solve all problems with less than 19 steps, but could not solve the long-sequence problems with roughly more than 21 steps by using up the entries of the transposition table.

5. CONCLUSION AND FUTURE WORK

PDS shows the highest solving ability both in 6x6 Othello and in Tsume-Shogi. In 6x6 Othello, PDS outperforms PN* by a factor of 2 to 5. The costs are nearly equal between proving the subtrees and disproving them. Therefore, PDS shows the ability disproving subtrees as well as proving them. However, PVS, the alpha-beta variant, also shows a high solving ability and speed. There probably are many non-forcing good moves rather than forcing moves in the game of 6x6 Othello, so PN-search variants cannot show the explicit predominance over standard depth-first search.

In Tsume-Shogi, there are many forcing sequences and sudden terminations in the solution tree, so PN-search variants almost show the predominance over variants of standard depth-first search, i.e., iterative deepening.

We have two conclusions. First, focusing on the solution speed for time critical cases such as endgame search, PDS is practical and relatively fast, but sometimes inefficient for short-sequence problems wandering further into wrong subtrees. PN* has the same tendency as PDS. PN* outperforms PDS in a few cases, but it is inferior to PDS in most cases. PN search is practical and fast for the problems with not so long sequences, but holds the same dangers as PN* and PDS. Depth-first iterative-deepening search is safe and practical for short-sequence problems, but cannot solve all the problems with long sequences.

Second, with regard to the solving ability and ignoring the solving speed, PDS has the highest proving ability as well as disproving ability. PN* requires more memory than PDS, so it is inferior in many cases. PN* has a less disproving ability. PN search has a good solving and disproving ability, but has a severe restriction of working memory. Good implementation or some modifications for PN search can be effective. Depth-first iterative-deepening search is powerless and impractical for long-sequence problems.

For PN*, PDS and iterative deepening, we have not yet implemented the smart management of transposition tables, i.e., neither the replacement of elements from unimportant elements by more important elements, nor garbage collection. For solving harder problems, better management of transposition tables is required by any means.

In Tsume-Shogi, proving and disproving are not symmetric. The costs of disproving the subtrees are generally several times larger than those of proving. In this case we expected that PDS is inferior to PN*. However, while searching PDS gave priority to proving the subtree rather than disproving and excelled over PN*. We should test the two algorithms in man-made Tsume-Shogi problems.

The implementation of using the dominance relation and omitting useless dropping pieces is also necessary. By implementing this, the search could have a much higher performance for many problems. Since our implementations are primarily based on the ideas of the algorithms, the efficiency can be much higher by making sufficient use of the domain-specific features.

There are some modifications of PN search and PDS. These are PN^2 search, PDS with PN^2 search, PDS*, and so on. We should implement these algorithms and examine them in the future.

6. REFERENCES

Allis, L. V. (1994a). *Searching for Solutions in Games and Artificial Intelligence.* Ph.D. thesis, University of Limburg, The Netherlands.

Allis, L. V., Meulen, M. van der, and Herik, H. J. van den (1994b). Proof-number search. *Artificial Intelligence*, Vol. 66, pp. 91–124. ISSN 0004–3702.

Breuker, D. M. (1998). *Memory versus Search in Games.* Ph.D. thesis, University of Maastricht, The Netherlands. ISBN 90–9012006–8.

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2001). The PN^2-Search Algorithm. *Advances in Computer Games 9* (eds. H. J. van den Herik and B. Monien), pp. xx–xx. Universiteit Maastricht, The Netherlands. ISBN 90–6216–5761.

Feinstein, J. (1993). Amenor Wins World 6x6 Championships! *British Othello Federation Newsletter, July*, pp. 6–9. Also availabel at URL `http://www.maths.nott.ac.uk/othello/Jul93/Amenor.html`.

Hiura, I. (1996). *What's True at the Positions Resigned? "Toryo no Shinso".* Mainichi Communications. ISBN 4–89563–650–X. (in Japanese).

Korf, R. E. (1985). Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, Vol. 27, pp. 97–109. ISSN 0004–3702.

Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, Vol. 62, No. 1, pp. 41–78. ISSN 0004–3702.

McAllester, D. A. (1988). Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, Vol. 35, pp. 287–310. ISSN 0004–3702.

Nagai, A. (1998). A new AND/OR Tree Search Algorithm Using Proof Number and Disproof Number. *Proceedings of Complex Games Lab Workshop*, pp. 40–45, ETL, Tsukuba.

Nagai, A. (1999). A New Depth-First-Search Algorithm for AND/OR Trees. M.Sc. thesis, Department of Information Science, The University of Tokyo, Japan.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA.

Schaeffer, J. (1990). Conspiracy Numbers. *Artificial Intelligence*, Vol. 43, pp. 67–84. ISSN 0004–3702.

Seo, M. (1995). The C* Algorithm for AND/OR Tree Search and its Application to a Tsume-Shogi Program. M.Sc. thesis, Department of Information Science, The University of Tokyo, Japan.

Seo, M., Iida, H., and Uiterwijk, J. W. H. M. (2001). The PN*-Search Algorithm: Application to Tsume-Shogi. *Artificial Intelligence.* ISSN 0004–3702. to be published.

Zobrist, A. L. (1970). A New Hashing Method with Application for Game Playing. Technical Report 88, Computer Science Department, The University of Wisconsin, Madison WI, USA. Reprinted in: *ICCA Journal*, Vol. 13, No. 2, pp. 69–73, 1990.
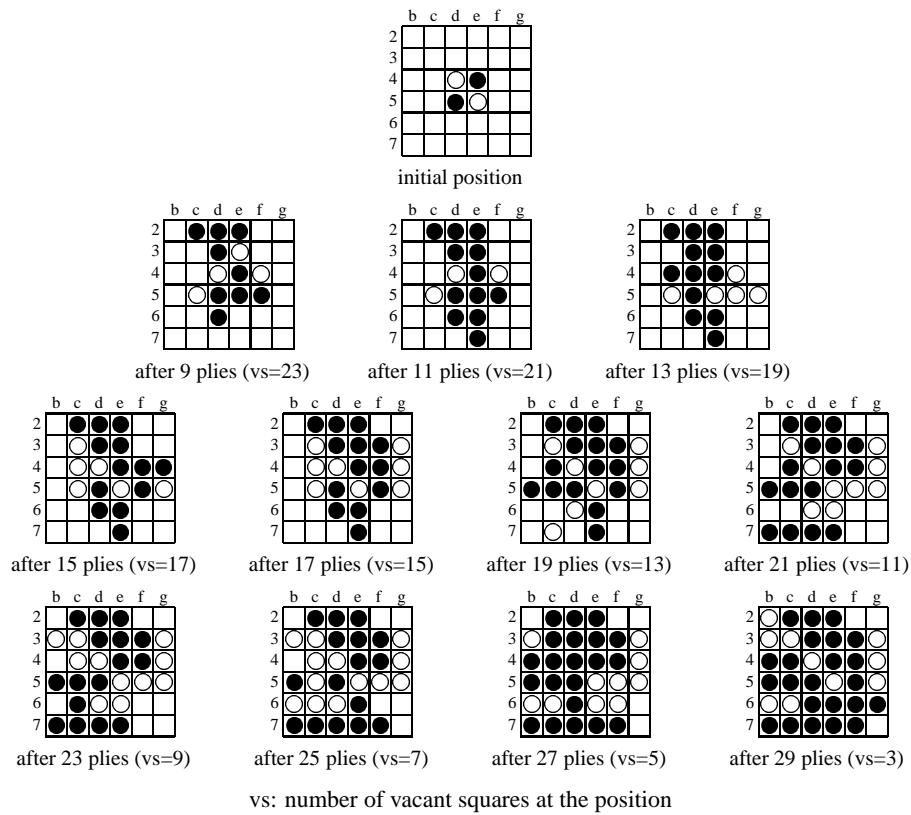
7. APPENDICES
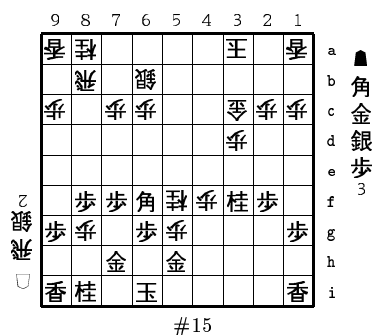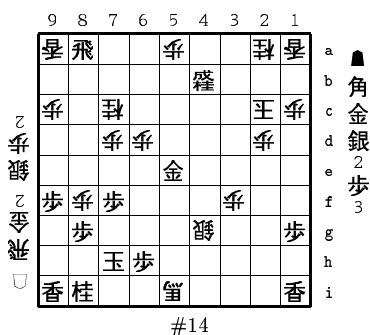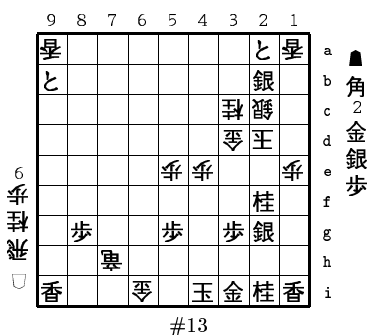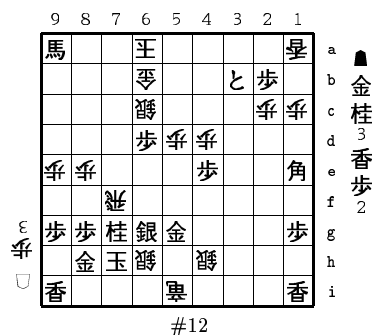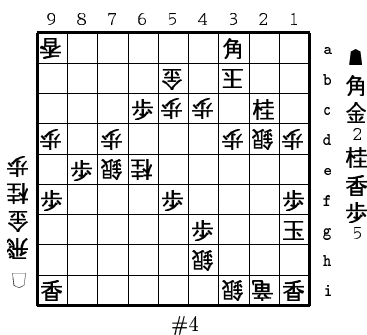
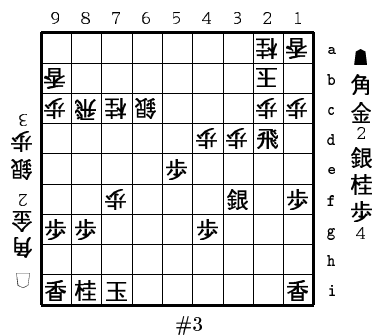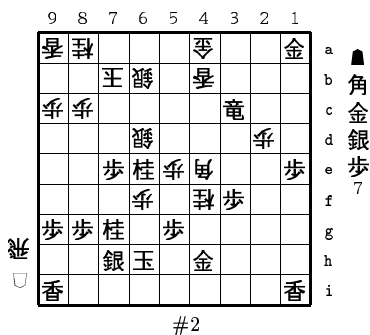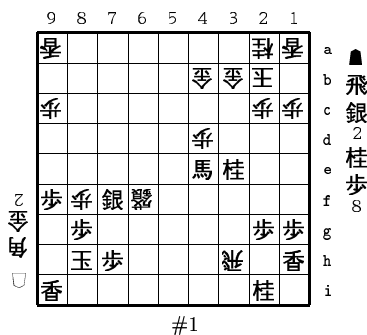### APPENDIX A: 6×6 OTHELLO POSITIONS ON PV

In 6x6 Othello, the second player (White) wins by 16 to 20 after 33 plies of best moves. Here is the principal variation (PV).

1.d3 2.c5 3.d6 4.e3 5.f5 6.f4 7.e2 8.d2 9.c2 10.e6 11.e7 12.g5 13.c4 14.c3 15.g4 16.g3 17.f3 18.c7 19.b5 20.d7 21.b7 22.b3 23.c6 24.b6 25.f7 26.f6 27.b4 28.b2 29.g6 30.g7 31.pass 32.f2 33.g2.

In this paper, the positions on PV have been used as the test positions, which are listed below. In 6x6 Othello, the 8x8 notation of moves is also followed. Therefore, each position is shown in a board using the notation of 8x8 board, though the file **a** and **h** and the rank **1** and **8** do not exist.

initial position

after 9 plies (vs=23)   after 11 plies (vs=21)   after 13 plies (vs=19)

after 15 plies (vs=17)   after 17 plies (vs=15)   after 19 plies (vs=13)   after 21 plies (vs=11)

after 23 plies (vs=9)   after 25 plies (vs=7)   after 27 plies (vs=5)   after 29 plies (vs=3)

vs: number of vacant squares at the position

APPENDIX B: TSUME-SHOGI PROBLEMS

#1

#2

#3

#4

#5

#6

#7

#8

#9

#10

#11

#12

#13

#14

#15
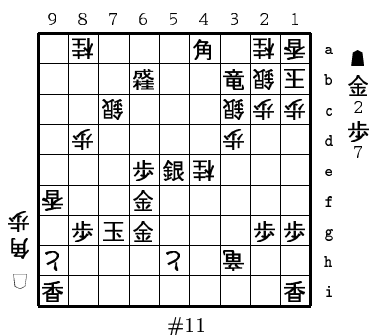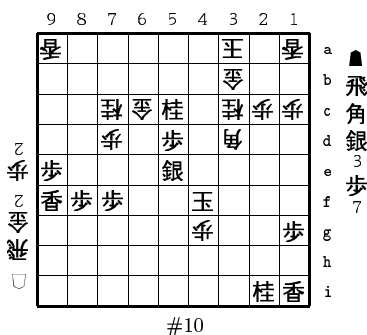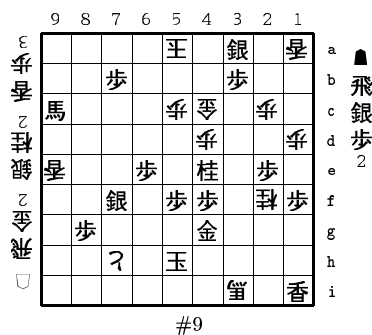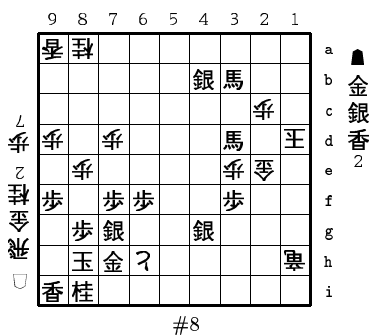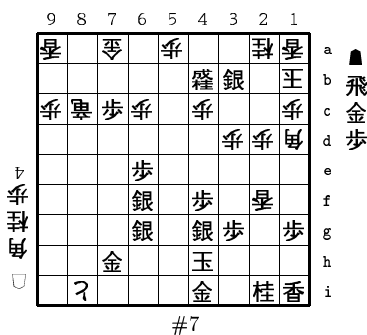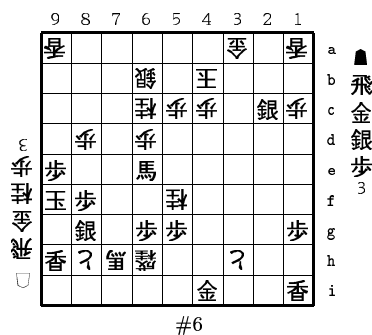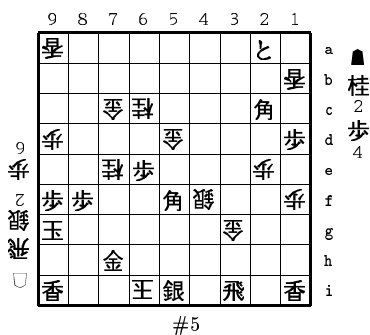
APPENDIX B: TSUME-SHOGI PROBLEMS (CONTINUED)

#16

#17

#18

#19

#20

#21

#22

#23

#24

#25

#26

#27

#28

#29

#30