Multiple-paths Search with Concurrent Thread Scheduling for Fast AND/OR Tree Search

Fumiyo Takano
Department of Computer Science and Engineering,
Waseda University,
Tokyo, Japan.
Japan Society for the Promotion of Science.
takano@kasahara.cs.waseda.ac.jp

Yoshitaka Maekawa
Department of Computer Science,
Chiba Institute of Technology,
Chiba, Japan.
maekawa@cs.it-chiba.ac.jp

Hironori Kasahara
Department of Computer Science and Engineering,
Waseda University,
Tokyo, Japan.
kasahara@waseda.jp

Abstract

This paper proposes a fast AND/OR tree search algorithm using a multiple-paths concurrent search method. Conventional heuristic AND/OR tree search algorithms expand nodes in only a descending order of heuristic evaluation values. However, since the evaluation values are heuristic, a solution node group sometimes includes nodes with lower evaluation values. The tree which has a solution node group including nodes with lower evaluation values requires a long time to be solved by the conventional algorithms. The proposed algorithm allows us to search paths including nodes with lower evaluation values and paths including nodes with higher evaluation values concurrently. For searching various paths concurrently, the proposed algorithm uses pseudo-threads and a pseudo-thread scheduler managed by a user program with low overhead compared with the OS thread management. The pseudo-thread scheduler can weight the amount of search on each path and schedule the pseudo-threads. The proposed algorithm can solve trees which have solutions including nodes with lower evaluation values also quickly. For performance evaluation, the proposed algorithm was applied to a tsume-shogi (Japanese chess problem) solver as a typical AND/OR tree search problem. In tsume-shogi, players can reuse captured pieces. Performance evaluation results on 385 problems show that the proposed algorithm is 1.67 times faster on the average than the previous algorithm df-pn.

1 Introduction

AND/OR tree searches are used for artificial intelligence such as games, puzzles, route searches, logic programming language Prolog[5], and the boolean satisfiability problem (SAT)[9]. The number of searching nodes on the AND/OR tree increases exponentially with the tree growth; therefore, the time for solving the tree lengthens exponentially as well. Accordingly, fast search algorithms are desired. Various AND/OR tree search algorithms have been studied [1, 7, 13, 2, 8], and df-pn (depth-first proof number search) [7] is especially known as a fast AND/OR tree search algorithm which uses proof and disproof numbers[1] as heuristic evaluation values. Df-pn λ search[8] using proof and disproof numbers and threats[13, 2] has been also proposed. These conventional algorithms search nodes in only a descending order of evaluation values because of higher search priority for highly evaluated nodes. However, since the evaluation values are heuristic, a solution node group sometimes does not include nodes with higher evaluation values. When a solution node group of a tree consists of nodes with lower evaluation values, searching in an ascending order of evaluation values sometimes solves the given tree quicker than searching in a descending order. We do not know whether evaluation values are not effective before searching, and whether to search in a descending order of evaluation values or to search in an ascending order of evaluation values.

Therefore, this paper proposes a new AND/OR tree search algorithm which searches nodes in both descending

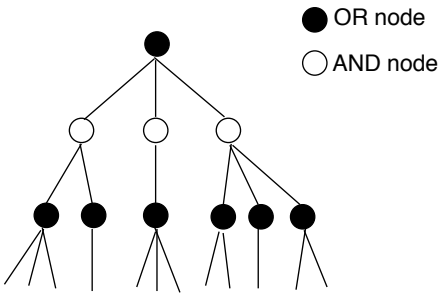

Figure 1. Example of an AND/OR tree.

and ascending orders of evaluation values concurrently. The proposed algorithm searches multiple-paths concurrently by using proposed pseudo-threads. To use the pseudo-threads enables weighting the amount of search on paths, and enables scheduling properly pseudo-threads searching the paths with low overhead. The proposed algorithm can solve trees which have solution node groups including nodes with lower evaluation values also quickly.

The outline of this paper is as follows. Section 2 describes AND/OR trees and conventional search algorithms for AND/OR trees. Section 3 proposes the fast AND/OR tree search algorithm which searches nodes in both descending and ascending orders of evaluation values with a concurrent pseudo-thread scheduling method. Section 4 evaluates the proposed algorithm using tsume-shogi problems (Japanese chess mating problems) as a typical example of an AND/OR tree search problem. Finally, Section 4 concludes this paper.

2 AND/OR Tree Search

An AND/OR tree consists of AND nodes and OR nodes. An OR node is true when at least one of its children is true. An AND node is true when all of its children are true. Figure 1 shows a typical example of an AND/OR tree. In the problems treated in this paper, the lengths of the solution sequences of trees are not given. Because the tree size is unknown before searching, search algorithms sometimes take a long computation time to solve the tree.

Since the tree growth lengthens the search time, AND/OR tree search algorithms widely have been studied to solve larger tree. Depth-first search and best-first search which are basic algorithms are often used to search AND/OR trees. Depth-first search expands the deepest leaf, and best-first search expands the most highly evaluated leaf. The trees which are solvable with best-first search are larger than the trees which are solvable with depth-first search while all of memory is not exhausted. However, best-first search uses larger memory than depth-first search uses, and best-first search cannot search a tree any more when all of memory are exhausted. To reap the advantages of depth-first search and best-first search, a modified depth-first search which expands nodes in the same order of best-first search, df-pn[7], has been proposed. The df-pn is a depth-first iterative-deepening search algorithm with thresholds of evaluation values[7].

The df-pn uses proof and disproof numbers[1] as evaluation values of a node. The proof number of a node is the number of proved posterity nodes that are required for proving the node. The disproof number of a node is the number of disproved posterity nodes that are required for disproving the node. Therefore, if the proof and disproof numbers are smaller, then the evaluation value is higher. Therefore, the proof and disproof numbers, as the evaluation value, are effective on most AND/OR trees.

The proof number pn(n) and the disproof number dn(n) at a node n are calculated as follows:

1. n is a leaf node.
   (a) node n is true.
   pn(n)=0
   dn(n)=∞
   (b) node n is false.
   pn(n)=∞
   dn(n)=0
   (c) node n is unknown.
   pn(n)=1
   dn(n)=1
2. n is a interior node.
   (a) n is OR node.
   pn(n)= min(pn of child nodes)
   dn(n)= sum(dn of child nodes)
   (b) n is AND node.
   pn(n)= sum(pn of child nodes)
   dn(n)= min(dn of child nodes)

The df-pn is a depth-first iterative-deepening search algorithm with thresholds of proof and disproof numbers. The df-pn expands nodes until the thresholds by depth-first search. When the proof number or the disproof number exceeds the thresholds at a node, the df-pn does not expand the node. Then, the df-pn begins the search again after increasing the thresholds. The df-pn searches nodes in an ascending order of the proof number in OR nodes and in an ascending order of the disproof number in AND nodes. The iterative-deepening algorithms, searching same nodes iteratively, requires prevention of unneeded re-expansion of nodes with a transposition table saved small information of the searched nodes.

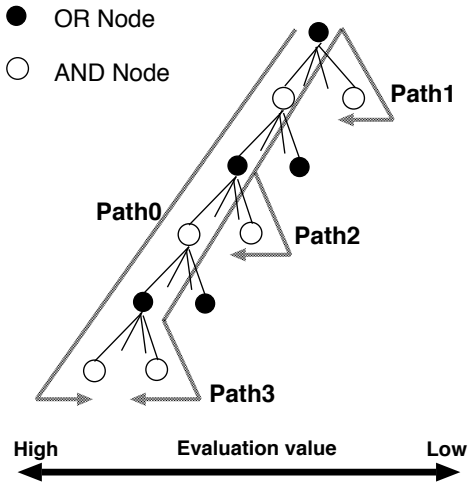

Figure 2. Multiple-paths search with concurrent thread scheduling.

However, since the evaluation values such as proof and disproof numbers are heuristic, a solution node group of a tree often does not include nodes with higher evaluation values. When a solution node group consists of nodes with lower evaluation values, searching in an ascending order of evaluation values solves the tree quicker than searching in a descending order.

3 AND/OR Tree Search with Concurrent Multiple-Paths Search

This section proposes a fast AND/OR tree search algorithm with a concurrent thread scheduling method. The proposed algorithm searches nodes in both descending and ascending orders of evaluation values concurrently for finding quickly also solutions including nodes with lower evaluation values.

Figure 2 shows a search by the proposed algorithm. Child nodes of any nodes are arranged in descending order of evaluation values, in Figure 2. The proposed algorithm searched multiple-paths such in Figure 2 concurrently. For the proper concurrent multiple-paths search, the proposed algorithm uses pseudo-threads and a pseudo-thread scheduler. Since the depth-first iterative-deepening search is used for the base of the proposed algorithm, the base algorithm is modified to search nodes in an ascending order of evaluation values.

The proposed algorithm can solve problems quickly which are solved slowly by the conventional search algorithms.

This paper defines the search in a descending order of evaluation values such as the best-first search or the df-pn as a *highest evaluation search*.

3.1 Search paths for concurrent search

The proposed algorithm searched multiple-paths such in Figure 2 concurrently. This section describes the paths searched concurrently.

The proposed algorithm searches concurrently a *highly evaluated path* and *lowly evaluated paths* for searching highly evaluated nodes and lowly evaluated nodes concurrently. The highly evaluated path always selects child nodes in a descending order of evaluation values. The highly evaluated path is a same path of the highest evaluation search. The lowly evaluated path is a path including lowly evaluated nodes. The lowly evaluated paths branch from the highly evaluated path at the each OR node included in the highly evaluated path. Because an OR node is true when one of its children is true, branching at OR nodes can reduce the search time of the tree when the lowly evaluated path is the part of the solution of the tree. In contrast, because an AND node is true when all of its children are true, branching at AND nodes does not affect the search time of the tree significantly. Therefore, the lowly evaluated paths branch at the only OR nodes. The branching nodes are prioritized in a deepening order of depth[11]. After branching, the paths pass from the lowly evaluated child node of the branching nodes to the highly evaluated child node. The search after branching is detailed in Section 3.3. Shallower and highly evaluated nodes are passed by more paths.

In Figure 2, Path0 which always selects nodes with the highest evaluation values at the point in time, is the highly evaluated path. The other paths are the lowly evaluated paths.

Because of the concurrent search of the paths such in Figure 2, the proposed algorithm can search as follows: 1) search highly evaluated nodes and lowly evaluated nodes concurrently, 2) search more highly evaluated nodes, 3) search less lowly evaluated nodes[5, 6, 12].

3.2 Concurrent search by pseudo-threads

This section describes the method for the proper concurrent search.

The proposed algorithm uses *pseudo-threads* and a *pseudo-thread scheduler* for searching multiple-paths concurrently. The pseudo-threads, instead of the threads created by OS, search the paths such in Figure 2. The pseudo-threads used in the proposed algorithm are classified into two types. One of the pseudo-threads is a *leader*, and the others are *workers*. The leader searches the highly evaluated path, Path0 in Figure 2. The workers search lowly evaluated paths. When the workers complete the search of

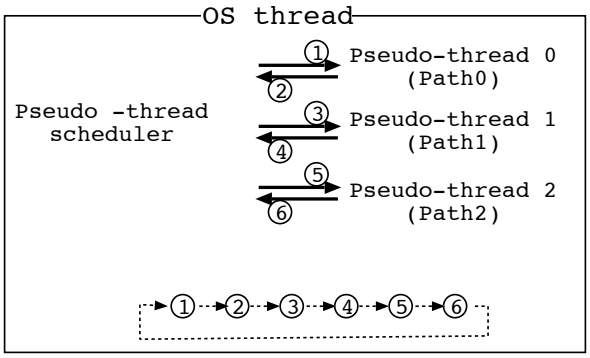

Figure 3. Pseudo-thread scheduling.

assigned paths, then they start searching other unsearched lowly evaluated paths.

The pseudo-thread scheduler manages the pseudo-threads with round-robin scheduling. Figure 3 shows the pseudo-thread scheduling on the proposed algorithm. The proposed algorithm uses the number of searched nodes as a quantum for round-robin scheduling to reduce costs of measurement of a quantum instead of time. Pseudo-thread switching at the proper point by the pseudo-thread scheduler enables the concurrent processing without mutual exclusions.

When a worker has proved or disproved a node included in the highly evaluated path, the pseudo-thread scheduler switches the pseudo-thread to the leader despite the processing order of pseudo-threads and the remained quantum for reduction of the unneeded search.

Quantum and ratios of amount of the search among pseudo-threads are changeable by users easily. Equally quantum among all paths reduces the search rate of the highly evaluated path. Therefore, the proposed algorithm equalizes the amount of the search on the highly evaluated path and the total amount of the search on the lowly evaluated paths to increase the search rate of the highly evaluated path. That is, the worker quantum is calculated using Equation (1).

$$\text{Worker quantum} = \frac{\text{Leader quantum}}{\text{Number of workers}} \quad (1)$$

The validity of this search rate is evaluated in Section 4.3. The two parameters in the proposed algorithm, the quantum and the number of pseudo-threads, are adjustable.

3.3 Iterative-deepening using evaluation values

This paper applies proof and disproof numbers[1] to evaluation values on the proposed algorithm. Smaller proof and disproof numbers mean higher evaluation values. Therefore, the leader in the proposed algorithm searches with the df-pn[7], which is depth-first iterative-deepening search with thresholds of proof and disproof numbers. The leader and the df-pn search nodes in an ascending order of proof or disproof number. The workers search the paths branching from the leader path to nodes with large proof numbers. In detail, workers select nodes in an ascending order of evaluation values (in a descending order of proof number) among the children of the branching nodes, which are included in the leader path. Then, the workers search the subtrees; the roots of the subtrees are the selected nodes. The search algorithm in the subtrees is equivalent to the search algorithm of the leader.

However, the evaluation values of the selected nodes are not in the range of the thresholds of the leader search. Therefore, the roots of the subtrees are unable to search in the leader search, which is the ordinary iterative-deepening with thresholds of the evaluation values. To search these subtrees with the workers, the proposed algorithm modifies the thresholds on the root nodes to cover the evaluation values of the root nodes. The subtrees are searched with the modified thresholds.

4 Performance Evaluation

This section evaluates performances of the proposed algorithm. The proposed algorithm was implemented as a solver for Tsume-Shogi problems. In this evaluation, we compare the search time of the proposed algorithm using proof and disproof numbers as evaluation values with the search time of the df-pn which is a typical search algorithm.

4.1 Evaluation environment

The proposed algorithm is evaluated on a computer with AMD DualCore Opteron 2.6 GHz and 4 GByte memory. This evaluation uses a core of the dual-core.

The proposed algorithm was applied to a tsume-shogi, or Japanese chess mating problem, solver. Tsume-shogi is a typical example of an AND/OR tree search problem. The evaluation used three tsume-shogi benchmarks, Zoku-tsumuya-tsumayaruya[4] (benchmark A), Shogi-Muso[3] (benchmark B) and Shogi-Zuko[3] (benchmark C). These are often used as benchmarks of solvers of Tsume-Shogi. Benchmark A consists of 202 problems, benchmark B consists of 100 problems and benchmark C consists of 100 problems. Because five problems in benchmark B and six problems in benchmark C are also in benchmark A, duplicated problems are excluded. The lengths of solution sequences of the problems in the benchmarks are between 9 and 873.

The search trees of tsume-shogi often require a long time to be solved, because the lengths of the solution sequences of the trees are unknown before searching. Therefore, this paper defines the *upper limit search time* as 18000 seconds. This evaluation did not use the results of the problems which are not solved by both the traditional and proposed algorithms within the upper limit search time.

The average speedup ratio of the proposed algorithm to the df-pn is calculated using Equation (2) shown below.

$$\text{Average speedup ratio} = \frac{\text{Total search time of df-pn}}{\text{Total search time of proposed algorithm}} \quad (2)$$

Execution with a single pseudo-thread means execution with df-pn.

4.2 Tsume-shogi (Japanese chess mating problem)

The evaluation of the proposed algorithm is performed using tsume-shogi problems as a typical example of an AND/OR tree search problem. Tsume-shogi problems are mating problems in shogi (Japanese chess).

The major difference between chess and shogi is that in shogi a player is able to reuse pieces which have been captured from the ones opponent, while in chess a captured piece is discarded. Therefore, a search tree of shogi is explosively larger than that of chess.

Tsume-shogi is a single player puzzle game based on the shogi rules, where a player is presented with a game state (a problem), and must checkmate the King of the Defender in the least number of moves possible. The Attacker plays first. The Attacker's moves must yield a check (a move by which a piece directly attacks the opponent's King), the Defender must select a move which extends the length of the mating sequence by avoiding being checkmated as long as possible. The length of a solution sequence in tsume-shogi is dependent on the number of plies (a half of a move) that lead from the given state to a checkmate. A solution sequence of tsume-shogi is unique. Furthermore, the length of a solution sequence is unknown before solving the problem. Evaluation of AND/OR tree search algorithms using tsume-shogi trees is useful because a search tree of tsume-shogi problem is a massive AND/OR tree.

Figure 4 shows an example of a tsume-shogi problem. Each piece in the upper part of Figure 4 moves in the fashion shown in the lower part of the same figure. The Attacker moves the Gold at 3c to 2b; that will be checkmate the King. The length of the solution sequence in this problem is one (one ply to check mate).

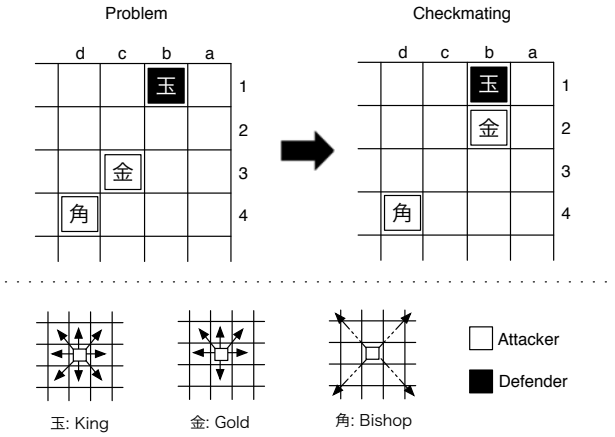

Figure 4. Example of tsume-shogi.

4.3 Quantum of pseudo-threads

In the proposed algorithm, the leader quantum is equal to the worker quantum (shown in Equation 1). Therefore, this section confirms the pseudo-threads quantum in the proposed algorithm. We compare *"Proposal"* which calculates the quantum with Equation (1), with *"Equally"* which uses same quantum among all pseudo-threads, and *"10-times"* which calculates the quantum with Equation (3) shown below.

$$\text{Worker quantum} = \frac{\text{Leader quantum}}{\text{Number of workers}} \times \frac{1}{10} \quad (3)$$

The the leader quantum in "Equally" is lower than that in "Proposed", and the the leader quantum in "10-times" is higher than that in "Proposed". Figure 5 shows speedup ratio of "Equally" , "10-times", and "Proposal" to the df-pn. The data in Figure 5 are the average speedup ratios among the three benchmarks. In case of two pseudo-threads, "Equally" is equal to "Proposed".

Figure 5 illustrates that "Proposed" outperforms the others. In any number of pseudo-threads and any quantum, "Proposal" is faster than "Equally", and "10-times" and also the df-pn. "Proposal" is 47% faster than "Equally" in the case of "100000nodes-4threads", and 35% faster than "10-times" in the case of "1000nodes-8threads" The geometric means of speedup ratios against the df-pn by "Proposed" 1.28, that by "10-times" is 1.08, and that by "Equally" is 1.03. The reason of the low speedup ratios of "Equally" is that the number of leader's search nodes in "Equally" is too small relatively. The speedup ratios of "10-times" approximate 1 because searching too many nodes by the leader approximates searching with the df-pn. These results show that "Proposal" is the most effective method because "Proposed" is over 20% faster than "Equally" and "10-times".

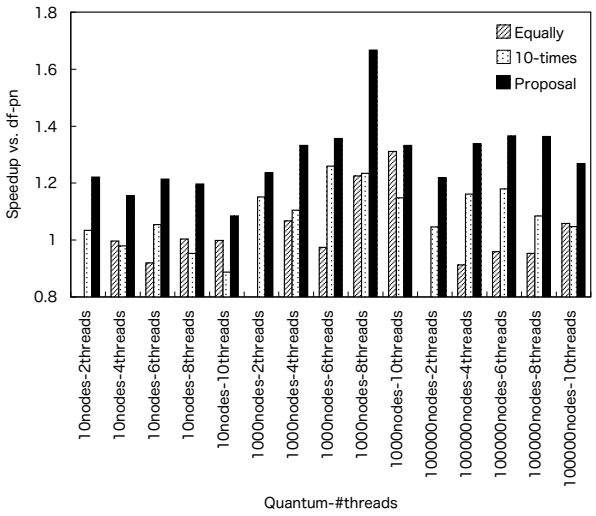

Figure 5. Search rate and speedup ratio compared with df-pn.

Table 1. Speedup ratio and parameters on benchmark A.

leader	#pseudo-threads				
quantum	2	4	6	8	10
10	0.99	0.94	1.02	1.06	0.90
100	1.04	1.14	0.99	1.15	1.04
1000	1.01	1.06	1.10	1.48	1.05
10000	1.23	1.04	1.37	1.00	1.29
100000	1.00	1.21	1.17	1.16	1.06

4.4 The number of pseudo-threads and quantum

This section evaluates the influence of the parameters and benchmarks on performances. The two parameters (the number of pseudo-threads and the quantum) are variable, in the proposed algorithm. Table 1, Table 2 and Table 3 show relation between speedup ratios to the df-pn and the parameters on the benchmark A, B and C, respectively.

Table 1, Table 2 and Table 3 illustrate that the proposed algorithm is up to 2.11 time faster than the df-pn, when the quantum is over 100 nodes. The maximum achieved speedup ratio is 2.32 when the number of pseudo-threads is 4 and the quantum is 100 nodes in benchmark C. The speedups and the parameters are not strongly correlated, in Table 1, Table 2 and Table 3. Because, a small change of the quantum or the number of pseudo-threads can lead to large and variable changes of search nodes and search time.

Table 2. Speedup ratio and parameters on benchmark B.

leader	#pseudo-threads				
quantum	2	4	6	8	10
10	1.06	1.05	1.44	1.30	0.91
100	1.18	1.65	1.16	1.66	1.14
1000	1.11	1.24	1.14	1.95	1.22
10000	1.70	1.10	1.69	1.15	1.89
100000	1.03	1.69	1.81	1.65	1.32

Table 3. Speedup ratio and parameters on benchmark C.

leader	#pseudo-threads				
quantum	2	4	6	8	10
10	1.96	1.70	1.71	1.42	1.45
100	2.06	2.32	2.03	2.19	1.85
1000	1.75	1.92	1.98	2.11	2.07
10000	1.68	1.71	1.33	1.62	1.53
100000	1.83	1.60	1.81	1.17	1.68

Table 1, Table 2 and Table 3 show that the most effective parameters vary from benchmark to benchmark. Therefore, we discuss relation between search times by the df-pn and influence of the parameters. Table 4 shows speedup ratios and geometric mean speedups (GM) on the all of benchmark A, B and C. Figure 6 shows ratios of search times with the proposed algorithm with the best parameter set (quantum: 1000, pseudo-threads: 8) to the worst parameter set (quantum: 10, pseudo-threads: 10) in Table 4. In Figure 6, plots over 1 means that search with the best parameter set is faster than that with the worst parameter set. Figure 6 illustrates that superiority of parameters and search times are not correlated. Therefore, optimum parameters cannot be determined before searching. 96% of plots shows the best parameter set is faster than the worst parameter set.

Table 4 shows the proposed algorithm with the best parameter set is 1.67 times faster than the df-pn. The peaks of geometric mean speedup ratios are obtained in case that the quantum is 1000 or 10000, and in case of 8 pseudo-threads. The proposed algorithm with the best parameter set is fastest also in benchmark A and B.

The results shown this section show that the proposed algorithm with any parameter except the case of the too small quantum is almost faster than the df-pn, although optimal parameters are not determinable before searching. The proposed algorithm in the all 385 problems is 1.67 times faster at the maximum, and 1.08 times faster at the minimum than df-pn.

Table 4. Speedup ratio and parameters on all problems.

leader	#pseudo-threads					
quantum	2	4	6	8	10	GM
10	1.22	1.16	1.21	1.20	1.08	1.17
100	1.30	1.41	1.25	1.40	1.32	1.34
1000	1.24	1.33	1.36	1.67	1.33	1.38
10000	1.38	1.26	1.56	1.23	1.50	1.38
100000	1.22	1.34	1.36	1.36	1.27	1.31
GM	1.27	1.30	1.35	1.36	1.29	-

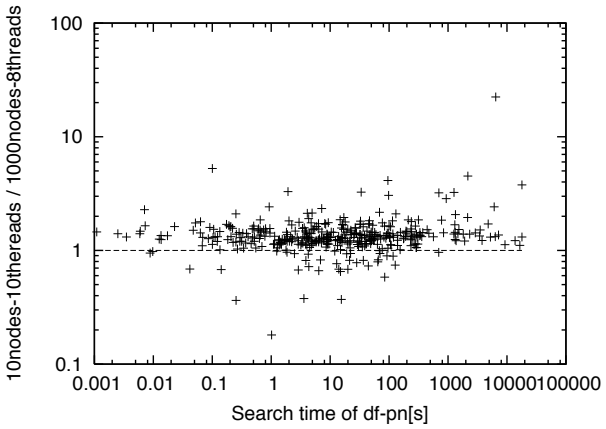

Figure 6. Influence of parameters and search time of df-pn.

4.5 Relation between speedup ratio and search time of df-pn

We evaluate the relation between the search times and the speedup ratios individually.

One of the purposes of the proposed algorithm is large speedup on the problems which are searched for a long time by the conventional algorithm. Figure 7 shows the search times of the df-pn and the speedup ratios of the proposed algorithm with 8 pseudo-threads and 1000 node quantum on the all benchmarks. Figure 7 shows that 63% of the speedup ratios of the problems are over 1. Earlier searching of many various paths including the lowly evaluated nodes results in fast finding of the solution nodes which require a long time to be searched by searching only highly evaluated path. The maximum achieved speedup ratio is 20.03. In this case, the proposed algorithm requires just only 323.14 seconds to solve the problem, though the traditional search algorithm df-pn requires 6471.12 seconds to solve the same problem. On the problems which are solved for a short time by the

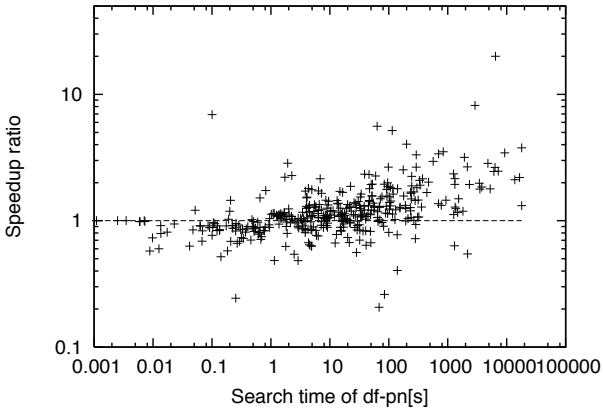

Figure 7. Search time of df-pn and speedup ratio compared with df-pn.

df-pn, speedup ratios are lower than 1. The reason of the low speedup ratio is an increase in the number of searched node because of searching the lowly evaluated paths. However, both the proposed and the conventional algorithms can solve the problems for just a short time although speedup ratios are low. These tendencies of the graph are the same as that using of other parameter sets.

Moreover, two unsolvable problems by the df-pn within the upper limit search time 18000 seconds could be solved within the upper limit search time by the proposed algorithm. In the two cases, the proposed algorithm could solve one of the problem in 4774 seconds, and the other in 13693 seconds. In Figure 7, the speedup ratios of these problems are calculated using the resulting search time of the df-pn that was bounded by the upper limit search time, 18000 seconds. Therefore, the actual speedup ratios are higher than 3.77, 1.67; this values are the speedup ratios shown in Figure 7.

The results in Section 4.4 and Section 4.5 show that the proposed algorithm outperforms the df-pn. According to the average speedup ratios, the speedup ratios of the proposed algorithm are almost more than 1 regardless of the parameters. Moreover, the proposed algorithm with proper parameters is 1.67 times faster than the df-pn. According to the individual speedup ratios, the speedup ratio is achieved up to 20.03.

5 Conclusions

This paper proposes a fast AND/OR tree search algorithm with a multiple-paths concurrent search method. To solve trees faster, the proposed algorithm searches concurrently paths including nodes with lowly evaluated nodes and

a path including highly evaluated nodes. The concurrent search is realized by using pseudo-threads and a pseudo-thread scheduler for weighting the amount of the search of the highly evaluated path and managing the pseudo-threads properly. The proposed algorithm was applied to a tsume-shogi solver as a typical example of an AND/OR tree search problem. The evaluation results shows as follows. The proposed algorithm is 1.67 times faster than the typical conventional search algorithm df-pn on the average for 385 problems on a single processor. The maximum achieved the individual speedup ratio obtained by the proposed algorithm is 20.03 compared with the df-pn. Moreover, the proposed algorithm can achieve speedup without prior parameter determination.

The proposed algorithm is not specialized for a specific tree nor a specific evaluation function. Therefore, the proposed algorithm can be applied to other AND/OR tree problems, search algorithms, and evaluation functions.

Also, we have proposed a parallelized version of the proposed algorithm[10]. This parallelized algorithm executed on four processor cores allows us to solve AND/OR trees 4.17 times faster than the sequential algorithm on an embedded multicore processor, NEC Electronics NaviEngine.

Acknowledgement

This research was partly supported by Grant-in-Aid for JSPS Fellows, JSPS Global COE Ambient SoC program, and NEDO Heterogeneous Multicore for Consumer Electronics project.

References

[1] L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. *Artificial Intelligence*, 66:91–124, 1994.

[2] T. Cazenave. A generalized threats search algorithm. In *Computers and Games*, pages 75–87, 2002.

[3] Y. Kadowaki. *Tsumuya Tsumazaruya*. Heibonsha, 1975.

[4] Y. Kadowaki. *Zoku-Tsumuya-Tsumazaruya*. Heibon-Sha, 1978.

[5] M. Kai, K. Kobayashi, and H. Kasahara. An OR parallel processing scheme of PROLOG using hierarchical pincers attack search. *Trans. of IPSJ*, 29(7):16–23, 1988.

[6] H. Kasahasa, A. Itoh, H. Tanaka, and K. Itoh. A parallel optimization algorithm for minimum execution-time multiprocessor scheduling problem. *Systems and Computers in Japan*, 23(13):54–65, 1992.

[7] A. Nagai and H. Imai. Proof for the equivalence between some best-first algorithms and depth-first algorithms for AND/OR trees. *IEICE TRANS.*, (10):1645–1653, 2002.

[8] S. Soeda, K. Yoshizoe, A. Kishimoto, T. Kaneko, T. Tanaka, and M. Muller. λ search based on proof and disproof numbers. *Trans. of IPSJ*, 48(11):3455–3462, 2007.

[9] W. M. Spears. A NN algorithm for boolean satisfiability problems. *Neural Networks, 1996., IEEE International Conference on*, 2:1121–1126 vol.2, Jun 1996.

[10] F. Takano, Y. Maekawa, and H. Kasahara. Parallel and concurrent search for fast AND/OR tree search on multicore processors. In *Proc. of Parallel and Distributed Computing and Networks (PDCN 2009)*, 2009.

[11] F. Takano, H. Sata, Y. Maekawa, K. Rokusawa, and N. Miyazaki. Adding parallel and node search to AND/OR tree hierarchical pincers attack search. *Transactions of Information Processing Society of Japan*, 46(SIG-12):319–329, 2005.

[12] F. Takano, A. Sekine, H. Sata, Y. Maekawa, and K. Rokusawa. Hierarchical pincers attack search using proof numbers and disproof numbers for AND/OR tree. *Transactions of Information Processing Society of Japan*, 45(SIG-11):280–289, 2004.

[13] T. Thomsen. Lambda-search in game trees - with application to Go. In *Computers and Games 2000*, pages 19–38, 2002.