PARALLEL AND CONCURRENT SEARCH FOR FAST AND/OR TREE SEARCH ON MULTICORE PROCESSORS

Fumiyo Takano
Department of Computer Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku, 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
2-17-1 Tsudanuma, Narashino, Chiba, Japan.
maekawa@cs.it-chiba.ac.jp

Hironori Kasahara
Department of Computer Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku, Tokyo, Japan.
kasahara@waseda.jp

ABSTRACT
This paper proposes a fast AND/OR tree search algorithm using a multiple paths parallel and concurrent search scheme for embedded multicore processors. Currently, not only PCs or supercomputers but also information appliances such as game consoles, mobile devices and car navigation systems are equipped with multicore processors for better cost performance and lower power consumption. However, the number of processor cores and the amount of memories in embedded multicore processors are limited for lowering power consumption and chip costs. Development of parallel application programs on embedded multicore processors requires exploitation of parallelism and effective utilization of small memories. The proposed algorithm allows us to search in parallel many paths including lowly evaluated nodes and paths including highly evaluated nodes. The algorithm also uses depth-first search, working on small memories. The proposed algorithm is applied for a tsume-shogi (Japanese chess problem) solver as a typical AND/OR tree search problem on an embedded multicore processor system, NEC Electronics NaviEngine with four ARM processor cores. Evaluation results for 100 problems show that the proposed algorithm executed on two processor cores is 2.36 times faster, and executed on four processor cores is 4.17 times faster than the sequential algorithm on the average.

KEY WORDS
Parallel Processing, Parallel AND/OR Tree Search Algorithms, Multicore Processors, Embedded Systems

1 Introduction

Currently, multicore processors[1, 2, 3, 4], which integrate multiple processor cores onto a chip, are widely used for better cost performance and lower power consumption. Not only PCs, high-end servers and supercomputers but also information appliances such as game consoles, mobile devices and car navigation systems are equipped with multicore processors. Multicore processors embedded in information appliances require a small number of processor cores and small memories for lowering power consumption and chip costs. Development of effective parallel application programs on embedded multicore processors requires exploitation of parallelism and effective utilization of small memory. Automatic parallelization of scientific computation and multimedia applications, which have large parallelism, has been studied[5, 6]. For difficulty in extraction of parallelism from programs for basic algorithms such as sort and search, manual parallelization of the algorithms has been researched. Map Sort[7], an example of parallelization of a sort algorithm for multicore processors, improves performance scalably.

Speedup of AND/OR tree searches is desired. AND/OR tree searches are used in processing such as games, puzzles, route searches, logic programming language Prolog[8], and the boolean satisfiability problem (SAT)[9]. Df-pn (depth-first proof number search) [10] is known as a fast sequential AND/OR tree search algorithm. Df-pn uses proof and disproof numbers[11] as evaluation values. Df-pn λ search[12] using proof and disproof numbers and threats[13, 14] has been also proposed. These conventional sequential algorithms search nodes in only a descending order of evaluation values. Conventional parallel search algorithms also search nodes in only a descending order of evaluation values. However, since the evaluation values are heuristic, nodes with higher evaluation values often are not included in solution nodes. When a group of solution nodes consists of nodes with lower evaluation values, to search in an ascending order of evaluation values solves the tree quicker than to search in descending order. Consequently, the conventional parallel search algorithms hardly achieve super-linear speedup. 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 parallel AND/OR tree search algorithm which searches nodes in both descending and ascending orders of evaluation values in parallel. Moreover, the proposed algorithm is applied for em-

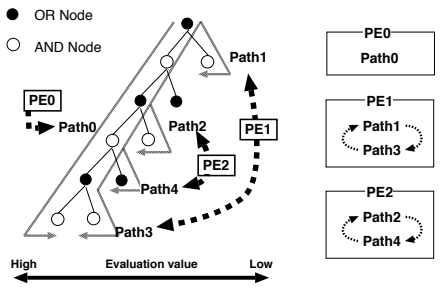

Figure 1. Proposed parallel search

bedded multicore processors equipped with a small number of processor cores and small memories. To use effectively a small number of processor cores, each processor core searches multiple paths concurrently. To reduce memory usage, processor cores search in depth-first search; depth-first search uses a small amount of memory because it does not keep all active searched nodes exponentially increasing during search in a memory.

The outline of this paper is as follows. Section 2 proposes the fast parallel AND/OR tree search algorithms for embedded multicore processors. Section 3 evaluates the proposed algorithm using tsume-shogi problems (Japanese chess mating problems) as a typical example of an AND/OR tree search problem on a multicore processor, NEC Electronics NaviEngine with four ARM cores. Finally, Section 4 concludes this paper.

2 AND/OR Tree Search with Multiple Paths Parallel and Concurrent Search

This section proposes a fast parallel AND/OR tree search algorithm for embedded multicore processors, which have a small number of processor cores and small memories. The proposed algorithm searches nodes in both descending and ascending orders of evaluation values in parallel for finding quickly also solutions including nodes with lower evaluation values. Moreover, in the proposed algorithm the parallel search of paths more than processor cores can solve trees faster than the parallel search of a small number of paths. To reduce memory usage, processor cores search with depth-first search.

2.1 Parallel search path

The proposed algorithm searches a *highly evaluated path* and *lowly evaluated paths* in parallel for searching highly evaluated nodes and lowly evaluated nodes. The highly evaluated path always selects child nodes in a descending order of evaluation values. The highly evaluated path is a same path of conventional sequential search algorithms which start searching from the highest evaluated node. The lowly evaluated path is a path including lowly evaluated nodes. The lowly evaluated paths branch from the highly evaluated path at the each node included in the highly evaluated path. The branch nodes are prioritized in the following order: 1) a deepening order of depth in OR nodes, 2) a shallowing order in AND nodes[15]. After branching, the paths pass from the lowly evaluated child node of the branch node to the highly evaluated child node. The search after branching is detailed in Section 2.4. Shallower and highly evaluated nodes are passed by more paths.

Figure 1 shows a search by the proposed algorithm. Child nodes of any nodes are arranged in descending order of evaluation value, in Figure 1. Path0, always selecting nodes with highest evaluation values at the point in time, is the highly evaluated path. The other paths are the lowly evaluated paths.

Because of the parallel search of the paths such in Figure 1, the proposed algorithm can search as follows: 1) search highly evaluated nodes and lowly evaluated nodes in parallel, 2) search highly evaluated nodes by a large number of processor cores, 3) search lowly evaluated nodes by a small number of processor cores.

2.2 Parallel and concurrent search

Each processor core on a multicore processor system searches multiple paths concurrently so that the multicore processor system searches in parallel more paths than the number of processor cores. The processor cores used in the proposed algorithm are classified into two types. One of the processor cores is a *leader*, and the others are *workers*. The leader searches the highly evaluated path only. The workers search multiple lowly evaluated paths concurrently. The leader searches only the highly evaluated path which is the same path as the best sequential search algorithm, to prevent performance degradation from the sequential algorithm. The concurrent search of multiple paths by the workers allows the parallel search of paths more processor cores in the multicore processor systems as a whole.

The parallel and concurrent search is described with Figure 1. In Figure 1, PE0 is the leader, and PE1, PE2 are the workers. Each worker searches two paths, therefore five paths in total are searched. Search paths are assigned to the workers cyclically. For example, Path1 and Path3 are assigned to PE1. Meanwhile, Path2 and Path4 are assigned to PE2.

Synchronization among processor cores is unneeded because each processor core can search the paths individually. The parallel search of paths more than processor cores on a small number of processor cores solves trees more quickly.

2.3 Concurrent search by pseudo-threads

The workers search multiple paths concurrently. The proposed algorithm uses *pseudo-threads* and *pseudo-thread-schedulers* for the concurrent search.

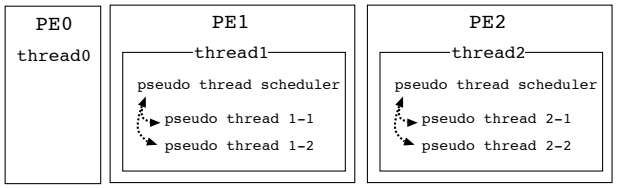

Figure 2. Threads and pseudo-threads

The pseudo-threads, instead of the threads created by OS, search the paths concurrently on the workers. The pseudo-threads search the different paths in a thread which is created by OS for each processor core. Pseudo-thread 1-1 in Figure 2 searches Path1 on PE1 in Figure 1, and pseudo-thread 1-2 in Figure 2 searches Path3 on PE1 in Figure 1. When the pseudo-threads complete the search of assigned paths, then they start searching other unsearched lowly evaluated paths.

The pseudo-thread-scheduler manages the pseudo-threads. The pseudo-threads and the pseudo-thread-scheduler operate on a worker thread (see Figure 2). The pseudo-thread-scheduler can schedule pseudo-threads as follows; 1) although actual concurrently searchable paths are less than requested paths, (pseudo-) threads which are not searching any paths do not use resources; 2) mutual exclusions among pseudo-threads in a same thread are unneeded; 3) grains of the concurrent search and ratios of amount of the search among pseudo-threads are changeable with small overheads.

2.4 Iterative-deepening using evaluation values

This paper applies proof and disproof numbers[11] to evaluation values on the proposed algorithm. Smaller proof and disproof numbers mean higher evaluation values. The path searched by the leader is the same path searched by df-pn[10], which is iterative-deepening depth-first search with threshold of proof and disproof numbers. The leader and the df-pn search in an ascending order of proof or disproof number. The workers search the paths branching from the leader path to nodes with large proof or disproof number. In detail, workers select nodes in an ascending order of evaluation values (in a descending order of proof or disproof number) among the children of the branch 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. The evaluation values of the selected nodes are lower than 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 evaluation values. To search these subtrees by 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.

The iterative-deepening algorithm, searching same nodes iteratively, requires prevention of unneeded re-expansion of nodes with a transposition table saved small information of the searched nodes.

2.5 Memory usage

The proposed algorithm consumes only a small amount of memory.

The space complexity of df-pn, the sequential algorithm in depth-first, is $O(d)$ (d means the maximum search depth). Although the transposition table generally requires larger space than others such as nodes require, the iterative-deepening is able to search also on a small amount of memory for the transposition table.

The space complexity of the proposed algorithm, parallelized df-pn, is $O(d \cdot p)$ (p means the number of the pseudo-threads). All processor cores are able to share the transposition table. The shared transposition table requires a memory as same amount of that used by df-pn. Sharing the transposition table can improve efficiency of the search, because a processor core refers information obtained by other processor cores.

3 Performance Evaluation

The performance of proposed algorithm was evaluated on an embedded multicore processor system, NEC Electronics NaviEngine with four ARM MP11 cores. The proposed algorithm was implemented as a solver for Tsume-Shogi (Japanese chess mating) problems. In this evaluation, we compare the search time of the proposed algorithm with the search time of df-pn, the sequential algorithm.

3.1 Evaluation environment

The proposed algorithm is evaluated on an embedded multicore processor system, NEC Electronics NaviEngine. The NaviEngine is equipped with MPCore with four ARM MP11 399 MHz cores, and is an SMP system. Figure 3 shows an architecture of the NaviEngine, and Table 1 shows a specification of the NaviEngine. The proposed algorithm used 16 MB of the memory for the transposition table.

The proposed algorithm was applied for 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 a tsume-shogi benchmark, Shogi-Zuko[16]. The benchmark consists of 100 problems. The lengths of solution sequences of the problems in the benchmark are between 9 and 611.

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 the search is conducted. Therefore, this paper defines the *upper limit search*

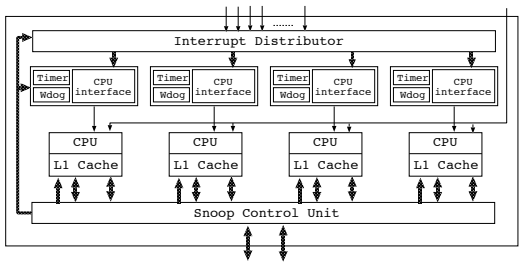

Figure 3. Block Diagram of MPCore

Table 1. Specification of NaviEngine

ARM11 MPCore	
Clock frequency	399 MHz
Instruction cache	32 KByte
Data cache	32 KByte
Main memory	256 MByte

time as 3600 seconds. Unsolvable problems within the upper limit search time are not used on the evaluation.

Each worker searches up to four paths concurrently, and searches 1000 nodes while a pseudo-thread turn. Execution by one processor core means execution with the sequential algorithm, df-pn. The relations of the number of the worker paths to the number of the total search paths including the leader path are shown in Table 2.

3.2 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 chess.

Tsume-shogi is a single player puzzle game based on the shogi rules, where a player is presented with a game state (problem), and must checkmate the King of the Defender in the least number of moves (solution sequence) possible. The Attacker plays first. The Attacker's moves must yield a check, 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 moves) 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.

Table 2. Number of search paths searched by each worker processors and by all processors

# core	# worker path	# total path
2	1	2
	2	3
	3	4
	4	5
3	1	3
	2	5
	3	7
	4	9
4	1	4
	2	7
	3	10
	4	13

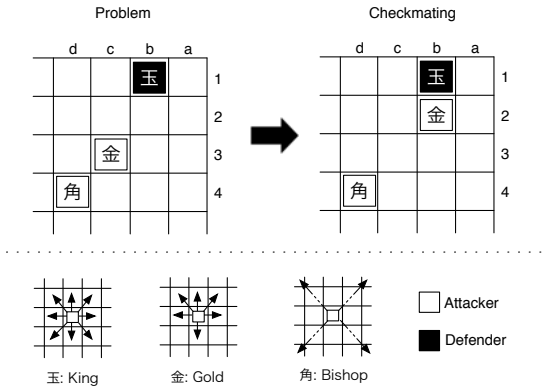

Figure 4. Example of Tsume-Shogi

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).

3.3 Evaluated results

This section evaluates the performance of the proposed algorithm.

First, we discuss the average speedup ratios. The average speedup ratio of the proposed algorithm is calculated using Equation (1) shown below.

$$\text{Speedup ratio} = \frac{\text{Total search time of df-pn}}{\text{Total search time of proposal}} \qquad (1)$$

The average speedup ratios of the proposed algorithm to df-pn are shown in Figure 5. Figure 5 shows that the pro-

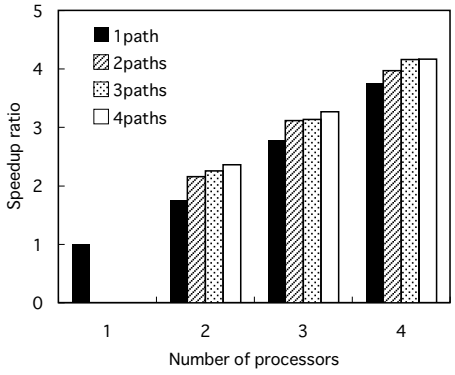

Figure 5. Average speedup ratio

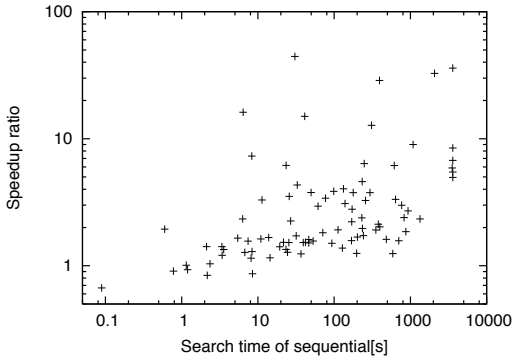

Figure 6. Sequential search time and speedup ratio

posed algorithm, each worker searches one path, is 1.76 times faster for two processor cores than the sequential algorithm, 2.78 times faster for three processor cores than the sequential algorithm, and 3.77 times faster for four processor cores than the sequential algorithm. The proposed algorithm scales almost linearly. The reason of the speedups is to search highly evaluated nodes and lowly evaluated nodes in parallel. Therefore, the proposed algorithm can find solution nodes early. Moreover, the increases in the worker paths searched concurrently lead to the increases in the speedup ratios. On two processor cores, execution with two worker paths is 23% faster, execution with three worker paths is 28% faster, and execution with four worker paths is 34% faster than that with one worker path. On four processor cores, execution with two worker paths is 5% faster than that with one worker path, execution with three worker paths is 10% faster than that with one worker path, and execution with four worker paths is 11% faster than that with one worker path. The reason of the speedups is that the increases in the search paths conduce to earlier search of the solution nodes, which are unfindable with the parallel search of a small number of paths. The decreases in the number of processor cores grow the effects of the increase in the search path because to search too many paths diminishes earlier findablity of solution nodes newly. In case of four worker paths, the proposed algorithm executed on two processors is 2.36 times faster, executed on three processors is 3.27 times faster, and executed on four processors is 4.17 times faster than the sequential algorithm on the average. These speedup ratios are larger than the number of processor cores, although the conventional parallel search algorithms hardly achieve super-linear speedup. The proposed algorithm executed by four processor cores with four worker paths reduces the average search time from 446.10 seconds in the sequential to 106.99 seconds.

The speedup ratios change according to problems. Next, we evaluate the relation between the search times and the speedup ratios individually. Figure 6 shows the search times of the sequential algorithm and the speedup ratios of the proposed algorithm. In Figure 6, the proposed algorithm is executed on four processor cores, and each worker searches four paths concurrently. Figure 6 shows that 23% of the speedup ratios of the problems are over 4. Earlier searches of many paths including the lowly evaluated nodes result in the fast find of the solution nodes which require a long time to be searched by searching only highly evaluated nodes. The maximum achieved speedup ratio is 44.39. The problem achieving the maximum speedup ratio requires 30.53 seconds to be searched by the sequential algorithm, and 0.69 second to be searched by the proposed algorithm. In 5% of the problems, the proposed algorithm is slower than the sequential search. However, these problems require just a short time to be solved by both the sequential and the proposed algorithms. For example, the problem of the minimum speedup ratio requires 0.09 second to be searched by the sequential algorithm and 0.13 second to be searched by the proposed algorithm. The extended time is only 0.04 second. Therefore, the small speedup ratios are practically ignorable.

Finally, the maximum search depth d on the evaluated problems is about 150. The proposed algorithm is effective also on embedded multicore processors, which have small memories, because the space complexity of the proposed algorithm is $O(d \cdot p)$.

These results show that the proposed algorithm can solve problems quickly on embedded multicore processors, which have a small number of processor cores and small memories. This conclusion is supported by as follows; 1) the average speedup ratios are larger than the number of processor cores, 2) the search time of a problem which the maximum speedup is achieved on is reduced from 30.53 seconds in the sequential algorithm to 0.69 second in the proposed algorithm, 3) the memory requirement of the proposed algorithm is linear complexity.

4 Conclusions

This paper proposes a fast parallel AND/OR tree search algorithm for embedded multicore processors, which have a small number of processor cores and small memories.

To solve trees faster on embedded multicore processors, the proposed algorithm searches more than processor cores paths including lowly evaluated nodes and a path including highly evaluated nodes in parallel. The concurrent search by each processor realizes the parallel search of paths more than processor cores. Processor cores search paths based on depth-first search to reduce memory usage. The proposed algorithm was applied for a tsume-shogi solver as a typical example of AND/OR tree search problems. The performance of the proposed algorithm was evaluated on NaviEngine with four processor cores and 256 MB memory. The evaluation results shows as follows. The proposed algorithm executed on four processor cores is 4.17 times faster than the sequential algorithm on the average. The average search time is reduced from 446.10 seconds in the sequential algorithm to 106.99 seconds in the proposed algorithm. The maximum achieved individual speedup ratio obtained by the proposed algorithm executed on four processor cores is 44.39. The increases in the number of the concurrent search path allow us the increases in the speedup ratios. The concurrent search of four paths by each processor core is 34% faster than the search of one path by each processor core. For the reasons stand above, the proposed algorithm can be applied for search on handheld game consoles and mobile devices.

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] M. Ito, T. Hattori, Y. Yoshida, K. Hayase, T. Hayashi, O. Nishii, Y. Yasu, A. Hasegawa, M. Takada, H. Mizuno, K. Uchiyama, T. Odaka, J. Shirako, M. Mase, K. Kimura, and H. Kasahara. An 8640 MIPS SoC with independent power-off control of 8 CPUs and 8 RAMs by an automatic parallelizing compiler. *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 90–598, Feb. 2008.

[2] A. Tanoka, A. Suga, F. Hayakawa, S. Tago, and S. Imai. Single-chip multi-processor integrating quadruple 8-way VLIW processors. *IEICE technical report. Image engineering*, 105(354):25–29, 2005.

[3] S. Maeda, S. Asano, T. Shimada, K. Awazu, and H. Tago. A real-time software platform for the cell processor. *Micro, IEEE*, 25(5):20–29, Sept.-Oct. 2005.

[4] K. Hirata and J. Goodacre. ARM MPCore; the streamlined and scalable arm11 processor core. In *ASP-DAC '07: Proceedings of the 2007 conference on Asia South Pacific design automation*, pages 747–748, Washington, DC, USA, 2007. IEEE Computer Society.

[5] K. Ishizaka, M. Obata, and H. Kasahara. Cache optimization for coarse grain task parallel processing using inter-array padding. *Transactions of Information Processing Society of Japan*, 45(4):1063–1076, 2004.

[6] Y. Wada, A. Hayashi, T. Masuura, J. Shirako, H. Nakano, H. Shikano, K. Kimura, and H. Kasahara. Parallelization of MP3 encoder using static scheduling on a heterogeneous multicore. *Transactions of Information Processing Society of Japan*, 1(1):105–119, 2008.

[7] M. Edahiro and Yamashita Y. Map sort : A scalable sorting algorithm for multi-core processors. *IEICE technical report. Dependable computing*, 2007(DC2006-94):19–24, 2007.

[8] 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.

[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] 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.

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

[12] 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.

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

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

[15] 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.

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