2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing

A System-Design Outline of the Distributed-Shogi-System Akara 2010

Kunihito Hoki
The Center for Frontier Science and Engineering
The University of Electro-Communications
Tokyo, Japan
hoki@cs.uec.ac.jp

Tomoyuki Kaneko
The Department of Multi-Disciplinary Science
Graduate School of the University of Tokyo
Tokyo, Japan
kaneko@graco.c.u-tokyo.ac.jp

Daisaku Yokoyama
Department of Informatics and Electronics,
Institute of Industrial Science, the University of Tokyo
Tokyo, Japan
yokoyama@tkl.iis.u-tokyo.ac.jp

Takuya Obata
Canon Inc.
Kanagawa, Japan
obata.taku@gmail.com

Hiroshi Yamashita
Freelance
Kanagawa, Japan
yss@bd.mbn.or.jp

Yoshimasa Tsuruoka
Department of Electrical Engineering and Information Systems
Graduate School of Engineering, the University of Tokyo
Tokyo, Japan
tsuruoka@logos.t.u-tokyo.ac.jp

Takeshi Ito
Department of Information and Communication Engineering
The University of Electro-Communications
Tokyo, Japan
ito@minerva.cs.uec.ac.jp

***Abstract*— This paper describes Akara 2010, the distributed shogi system that has defeated a professional shogi player in a public game for the first time in history. The system employs a novel design to build a high-performance computer shogi player for standard tournament conditions. The design enhances the performance of the entire system by means of distributed computing. To utilize a large number of computers, a majority-voting method using four existing programs is combined with a distributed-search method. Although the performance of the entire system could not be tested, the majority-voting component increased the winning percentage from 62% to 73%, and the distributed-search component increased it from 50% to 70% or more.**

Keywords—chess variant; shogi; distriuted search; majority voting; Akara 2010

I. INTRODUCTION

Human-computer chess matches have provided an index of advances in modern artificial intelligence. One of the most prominent pieces of historical evidence has been provided by Deep Blue, which won the six-game match against Garry Kasparov by one point in 1997 [1]. Since then, chess machines have started playing chess better than top-level human players. Nowadays, programs running on mobile phones seem to have reached the grandmaster level, and Pocket Fritz 4 won the Copa Mercosur tournament in 2009 [2].

In creating artificial intelligence, shogi (a Japanese chess variant) is probably a more difficult target than chess [3]. Because shogi has a dropping rule, i.e., a captured piece can be dropped back onto the board, the average number of legal moves is greater than in chess (the raw branching factor in shogi is around 80) [4]. This suggests that a shogi program needs to search a much larger game tree than a chess program to reach the grandmaster level. In fact, human-computer shogi matches are still challenging for a computer player in 2013.

In chess and shogi, the performance of game-tree searches is considerably improved by pruning techniques, such as alpha-beta pruning [5], with the use of a large transposition table [6, 7].

This research has been supported by the Information Processing Society of Japan (IPSJ).

978-0-7695-5005-3/13 $26.00 © 2013 IEEE
DOI 10.1109/SNPD.2013.24

Therefore, a distributed search should visit more nodes in the tree than a single-processor search does without losing the efficiency of such pruning techniques. Compared to SMP parallelization such as dynamic tree splitting (DTS) [8], the difficulty of distributed computing comes from the limitation in network-connection speed between multiple processors: (i) synchronization of individual searches at each processor when one processor experiences a beta cutoff, and (ii) management of the large transposition table that has to be updated and probed frequently by the multiple processors.

In this paper, we discuss distributed computing in shogi to play grandmaster-level games. We propose a novel design to build a high-performance yet fault-tolerant system for demonstrations in standard tournament conditions of shogi. Our system design utilizes multiple existing shogi programs. It also utilizes massive computer resources, i.e., a lot of existing computers of various architectures at various places with internet connections.

Our system uses the majority-voting method proposed by Obata *et al*. [9]. The voters are four different existing programs, Gekisashi[1] [10], GPS Shogi[2], Bonanza[3] [11], and YSS[4]. Their performance is enhanced by state-of-the-art techniques such as machine learning of evaluation functions, which has now become common practice in computer shogi due to the successful application in Bonanza [12]. Given a large number of available computers, the majority-voting method is combined with the distributed-game-tree-search method [13].

The shogi program Akara 2010 has been developed on the basis of this proposal and defeated one of the top female players, Ichiyo Shimizu, on October 11, 2010. Here, the game used a standard time control, i.e., 180 minutes plus a 60-second countdown. This was the first time a shogi machine defeated a professional in a public game. Moreover, some new results of the majority-voting method are presented in this paper.

II. Related Work

Distributed computing of game-tree searches has been extensively studied. One prominent strategy for using a massive computer resource is to build specialized hardware for the game-playing program to lift the limitation in network-connection speed. Deep Blue [1] and Hydra [14] are famous examples of massively parallelized chess machines.

The second strategy for distributed computing is to improve the divisions of the tasks on the bases of alpha-beta pruning, where the entire game tree is decomposed into multiple sub-trees. One naive way would be to divide the tasks at the root position. Here, all children of the root node are partitioned among the multiple processors, and each processor takes charge of the subset of the children by using a local transposition table. Since this naive way does not change the subsets throughout an entire search, all separate tasks across multiple processors can be independently computed without frequent network communication. One important extension from such a naive parallelization would be dividing the tree not only at the root node but also at multiple interior nodes of different depths. Such an extension can be found in early work, e.g., the young brothers wait concept (YBWC) [15], asynchronous parallel game-tree search (APHID) [16], and transposition-table driven scheduling alpha-beta (TDSAB) [17].

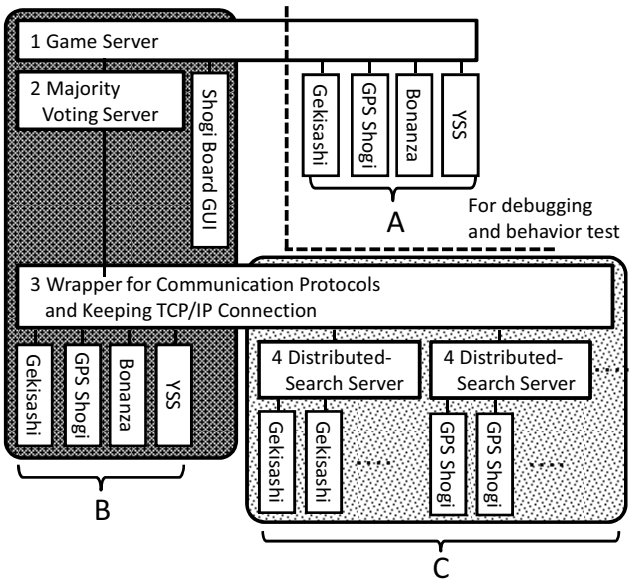

Fig. 1 Outline of entire system.

The third strategy is the extended use of pondering methods [18, 19]. In an ordinary pondering method, the expected move of the opponent is considered as already played, and a computer player starts searching during the opponent's thinking time to avoid the computer being idle while the opponent is thinking. The reported method is to start searching speculatively ahead on the basis of the expected move sequence during both the player's and opponent's thinking time.

Althöfer and Snatzke have reported 3-Hirn in two-player games [20], where two programs provide move opinions and a human selects a single move from these two. Therefore, the majority-voting method can be regarded as a reduction from 3-Hirn, where the human intervention in two computer programs is removed. The majority voting has been applied to shogi [9] and chess [21]. Moreover, it has been reported that majority voting is even effective in the Monte-Carlo tree search of Go programs [22].

The majority-voting method utilizes the distributed environment for better performance. This was demonstrated in tournament conditions in 2009 [23]. The distributed shogi-tree search has also been demonstrated in tournament conditions in 2010 [13]. The notable advantage of these methods is simplicity, and most existing shogi programs can be made into worker programs with minimal modifications.

Recently, an artificial intelligence system has challenged human experts in games other than variants of two-player games.

[1]Developed by Y. Tsuruoka, D. Yokoyama *et al*., commercial software from Mynavi corporation.
[2]T. Kaneko, T. Tetsuro *et al*., source codes are available online from http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/.
[3]Developed by K. Hoki, commercial software from Magnoria, Inc., source codes are available online from http://www.geocities.jp/bonanza_shogi/.
[4]Developed by H. Yamashita, commercial software (named AI Shogi) from e frontier, Inc.

TABLE I. THE LIST OF COMPUTERS USED IN AKARA2010

Set	*Processor*	*Number*
Hongo Campus, the University of Tokyo[a, b, c, d]		
B	Intel Xeon W3680 3.33GHz, 6 cores	4
istbs Cluster, Graduate School of Information Science and Technology, The University of Tokyo[a, b, c]		
C	Intel Xeon 2.80GHz, 2 cores	106
C	Intel Xeon 2.40GHz, 2 cores	60
InTrigger[5] ***Cluster***[a, b, c, d]		
C	Intel Xeon X5560 2.80GHz, 8 cores	10
C	AMD Opteron 2380 2.4GHz, 8 cores	2
C	Intel Xeon E5410 2.33GHz, 8 cores	9
Komaba Campus, the University of Tokyo[b]		
C	Intel Xeon X5570 2.93GHz, 8 cores	1
C	Intel Xeon X5470 3.33GHz, 8 cores	1
C	Intel Xeon X5365 3.00GHz, 8 cores	1
C	AMD Opteron 2376 2.3GHz, 8 cores	4
C	AMD Opteron 280 2.4GHz, 4 cores	1
C	Intel Core2 Quad Q6700 2.66GHz, 4 cores	1
The University of Electro-Communications[c]		
C	Intel X5680, 12 cores	4
C	Intel W3680, 6 cores	1
C	Intel Core i7 980X, 6 cores	1
C	Intel W3440, 4 cores	1
Kanagawa[d]		
C	Intel Core i7 980X 3.3GHz, 6 cores	1

Used by [a]Gekisashi, [b]GPS Shogi, [c]Bonanza, and [d]YSS

In 2011, Watson defeated two former winners of the quiz show Jeopardy! [24]. Watson is a question answering system that can carry out natural language processing and reasoning to answer questions on a quiz show [25].

III. SETUP OF ENTIRE SYSTEM

Figure 1 shows the outline of the entire system. The workers in the three sets (A, B, and C) individually compute their best move for a given position. Each worker is one of the four programs running on a single computer. Because the computer has multiple cores, each worker carries out a shared-memory parallel search of the given game tree. Sets A, B, and C connect to servers 1, 2, and 4, respectively.

Server 1 continuously makes and runs shogi games between two workers. For the server program, we used the shogi-server program package that is available online. This program had been maintained and run on the internet shogi server, Floodgate, for the past two years [26]. For the purpose of programs and the hardware test, game-server 1 had worker-set A and majority-voting-server 2 connected for 24 hours. The game records and log messages were continuously uploaded to the Web, and the authors were able to check the activity of the system any time.

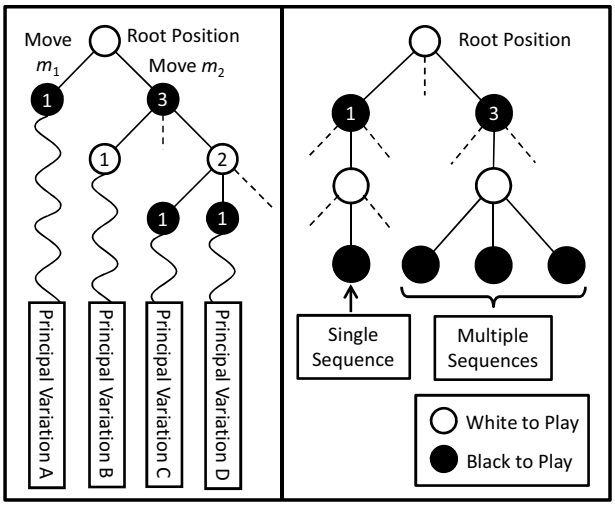

Fig. 2 An example of the game tree and majority voting method.

The system is able to play a game with human players using a shogi-board Graphical User Interface (GUI) client connected to game-server 1. We used the SFICP GUI program with some modifications to suspension and resumption functions [27].

Majority-voting-server 2 and the following play the central role in the entire system. The main role of server 2 is to gather move opinions on the current game position from workers and decide the move to play on the basis of majority voting, where the weights of votes are adjustable parameters. Majority-voting server 2 allows shutdowns and connections from workers anytime in the middle of a game. However, at least one worker has to be connected to guarantee the continuation of a game.

Unit 4 is a wrapper program that absorbs the difference in communication protocols. Moreover, unit 4 manages to restart each worker and establish a new connection again to the majority-voting-server 2 when the old internet connection experiences an interruption.

The dark-shaded box indicates the indispensable units to continue a shogi game. These were necessary to display the move decision of the system to the GUI client. Moreover, the pondering function while the opponent is thinking was provided by majority-voting-server 2, and the suspend-and-resume function is provided by game-server 1, majority-voting-server 2, and the GUI client. These indispensable units were composed using reliable hardware and software that have passed more than 1000 game tests (see Table I).

On the other hand, the light-shaded box indicates optional units that are designed to boost the performance of the entire system. The purpose of the separation from the dark-shaded ones is to utilize whatever hardware and system software are available at that moment. The requirements for the workers inside the light-shaded box are to result in a freezing or termination state without sending low-quality moves when they

experience abnormal behavior. Distributed-search-server 4 divides the game-tree into multiple sub-trees and assigns one of these sub-trees to each worker in set C. These workers ran on the cluster istbs, InTrigger[5], and the other computers (see Table 1).

IV. MAJORITY VOTING METHOD

The primary goal of the majority-voting method is to make each base program stronger by composing an ensemble of the four programs. Because these programs are different, we are able to have four different principal variations. Some variations may branch off at the root node, while the others branch off at some interior nodes. In the example shown in the left panel of Fig. 2, the number in the nodes indicates votes, and move m_2 is the majority. The selection of m_2 is unfounded in the strict sense. However, one can speculate that the selection of m_2 supported by three programs is less risky than the other one supported by a single program.

The right panel of Fig. 2 shows an example of a game tree, where the majority voting is effective. There are four playable move sequences. Because of the huge size of the game tree and the limitation of the time control, we have no hope to verify these sequences by in-depth consideration. Under this circumstance, the selection of move m_2 supported by three playable sequences is reasonable. However, a single base program has a probability of 0.25 to select move m_1. Majority voting is able to reduce this probability.

There are two more favorable properties of the majority-voting method in addition to the enhancement of strength. The first is fault tolerance. The standard tournament conditions of shogi games are real-time games. Therefore, stable computations and input/output functions are necessary to win a game. However, to the best of our knowledge, existing distributed-search methods do not take into account hardware and network troubles. That is, the more computers, the more accidents. In contrast, the majority-voting method is fault tolerant. The majority-voting server allows the shutdown and connection of a worker at any point in time. Therefore, whatever the number of computers we use in the light-shaded box in Fig. 1, the probability of having accidents is mostly determined only by those in the dark-shaded box.

The second favorable property of the majority-voting method is that it is difficult for a human player to predict the result of votes in advance. This property is expected to help disturb the opponent modeling of a human player. That is, even if she/he knows a tactical deficiency of a specific program in advance, she/he cannot use this knowledge to win the game when the other programs do not have the same deficiency.

We design the majority-voting server to be as simple as possible to avoid unexpected accidents. The server program is written by using about 1000 lines of Perl script. The server program had the following three phases:

Phase 1: The system considers a move to play. After waiting for thinking-time t, the majority-voting server sends the majority move to the game server and all workers. The system proceeds to phase 2.

TABLE II. THE PERFORMANCE OF MAJORITY VOTING

Player	*Win*
The majority voting of these four players	73%
Gekisashi	50%
GPS Shogi	36%
Bonanza	62%
YSS	37%

Phase 2: The system predicts the opponent's move. As soon as it receives the opponent's move from the game server, the systems forward it to all workers and returns to phase 1. After waiting for a shorter time than t, the opponent is assumed to play the majority move. The system sends the predicted move to all workers and proceeds to phase 3.

Phase 3: The system considers a response move to the predicted move. It waits for the opponent's move from the game server. If the opponent's move and the predicted move are the same, then it returns to phase 1. Otherwise, the system sends the opponent's move as an alternating move to all workers and then returns to phase 1.

If the system is the player who moves first, the phase starts from 1; otherwise the phase starts from 2. To resume a suspended game, the majority-voting server sends a move sequence to the suspended position to all workers. Then the phase starts from 1 if the system is the player who moves first from the suspended position. Otherwise, it starts from 2.

The thinking-time t is determined in terms of the extent of voting agreements. When all workers vote for the same move, the time t is reduced by half from a standard thinking time. On the other hand, when all workers vote for different moves, the time t is increased up to twice from the standard thinking time. The standard thinking time is set to consume 180 minutes by 60 moves.

The workers are required to carry out the following processing:

- They keep on considering the best move for the current position. As soon as the best move at the moment changes, they send the new best move to the majority-voting server.
- When a move arrives from the majority-voting server, the workers update the current position in accordance with the move. Also, when an alternation move arrives from the majority-voting server, they play back the last move of the position and then update the current position in accordance with the alternation move.

The workers also take care of the following auxiliary functions:

[5]New IT Infrastructure for the Information-explosion Era, MEXT Grant-in-Aid for Scientific Research on Priority Areas.

- They send the total nodes searched to the majority-voting server at the right interval to keep network connections alive.
- They send the opening signal when a worker recognizes the current position an opening. This signal is used to reduce the thinking time in early opening.
- They send the thinking-stop signal. This signal is sent when a worker finds the current position in the opening database or the position is solved. The majority-voting server is able to play a move immediately when the majority is achieved from workers that send this signal.

Table II shows the performance of the majority-voting method. Each winning percentage is measured by 1000 games whose opening moves are randomly chosen according to their opening databases. The opponent of all games is Gekisashi, and the winning percentage, 50%, of Gekisashi is a theoretical value. For each move, all players use four seconds of the same ordinary desktop computer with a sufficient memory allocation for the transposition tables. To resolve tied votes, the weight of the vote of Gekisashi is set to a value slightly higher than 1. The majority voting provides the highest percentage, 73%. Although these results are produced using the short time control, they indicate that the majority voting using these four programs is effective.

V. Distributed Search for Game Trees

Distributed searches of the game tree remain challenging, and a definitive method is still unknown. Because a distributed computing environment needed to be practically used to build the entire system, we adopted a simple master/slave model of communication in this paper. In this section, we use the terminologies master and slaves instead of server and worker. That is, the distributed-search server is the master program, and the workers in set C are the slave programs.

The master/slave model behaves as described by Kaneko and Tanaka [13], and we summarize the model below:

- The master program expands a shallow and narrow tree T rooted at the current node and assigns each slave to search for a large tree rooted at a node contained in T. The small tree T is composed on the basis of the move ordering of the slaves.
- The slave assigned a leaf node of T takes charge of all children of the leaf node. On the other hand, the slave assigned an internal node of T takes charge of all children nodes that are not contained in T.
- Each slave sends its best child and the corresponding evaluation value every time these are updated through their tree searches and the master program updates and sends the best move to the majority-voting server.

This distributed-search method is effective because each slave is able to start a tree search from a deeper node than the current node if the master program expands a proper small tree T. The server program is written by about 2,500 lines of Perl script.

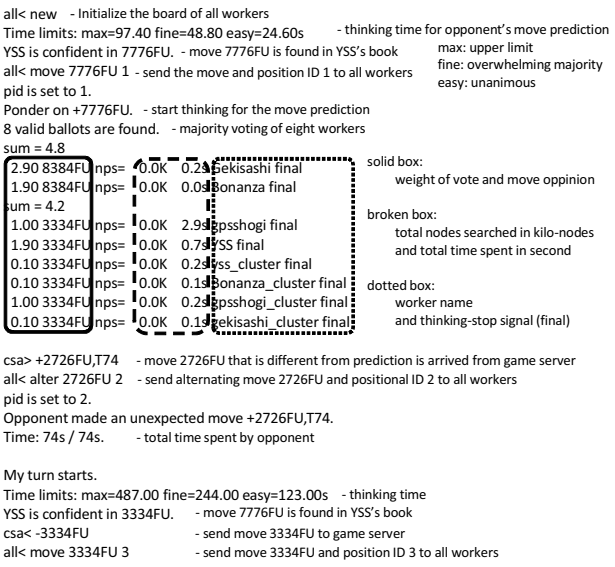

Fig. 3 Extract from log file of majority-voting server of game on October 11, 2010 and some explanations [28].

This distributed-search method has two advantages. The first is the less-frequent communication between master and slave programs. The size of messages that each slave has to send to the master is less than 1000 bytes, and the communication frequency is about once per second. The second is the applicability to an existing program. The requirements for a slave are to compute a good move ordering, the best child, and an evaluation value. Note that any practical shogi program has these functions.

The performance test using GPS Shogi and ordinary desktop computers with the time control of 15 seconds per move shows the effectiveness of our implementation. The distributed search using eight slaves has a winning percentage of more than 70% against a single slave program.

In the game on October 11, 2010, because most computers had not been available for tests, the weights of votes from the light-shaded box in Fig. 1 were set sufficiently smaller than 1 to stay on the safe side. Although the distributed-search components did not strongly influence the move decision, the components were effectively used to determine thinking-time t as described in Sec. IV.

VI. Conclusion

We have presented a system design to build a high-performance computer shogi player. In the system design, we have considered enhancing the performance by means of a distributed computing environment. At the same time, we have also considered the stabilization of the entire system to play a game under standard tournament conditions. To utilize a large

number of computers, a majority-voting method has been combined with the distributed-search method. On the basis of this system design, we have developed a shogi-machine, Akara 2010. The move decision has been made by the majority voting of four programs: Gekisashi, GPS Shogi, Bonanza, and YSS. Because the entire system of Akara 2010 had been operational for only a few days, a statistically reliable test of the performance was not available. We have carried out partial tests of the entire system, where the majority-voting component increased the winning probability from 62% to 73%, and the distributed-search component increased it from 50% to 70% or more. Moreover, Akara 2010 defeated a professional player in a public game, the log file of which is available online (see Fig. 3). These results indicate the effectiveness of our system design.

ACKNOWLEDGMENT

We are grateful to Professor Hitoshi Matsubara for the fruitful discussion on the human-computer shogi match project. We are also grateful to associate Professor Tsuyoshi Hashimoto for the fruitful discussion on the shogi opening database. We are grateful to Professor Hideyuki Nakashima for his support of this work.

REFERENCES

[1] M. Campbel, A.J. Hoane, Jr., and F.-h. Hsu, "Deep blue", Artificial Intelligence, Elsevier, vol. 134, pp. 57-83, 2002.

[2] M. Crowther, The Week in Chess (TWIC) 771, August 17, 2009, url: http://www.chess.co.uk/twic/twic771.html (last access: 2013).

[3] H. Iida, M. Sakuta, and J. Rollason, "Computer Shogi", Artificial Intelligence, Elsevier, vol. 134, pp. 121-144, 2002.

[4] H. Matubara, H. Iida, and R. Grimbergen, "Chess, Shogi, Go, A Natural Development in Game Research", ICCA Journal, International Computer Chess Association (ICCA), vol. 19, pp. 103-112, 1996.

[5] D.E. Knuth and R.W. Moore, "An Analysis of Alpha-Beta Pruning", Artificial Intelligence, Elsevier, vol. 6, pp. 293-326, 1975.

[6] A.L. Zobrist, "A New Hashing Method with Application for Game Playing", ICCA Journal, International Computer Chess Association (ICCA), vol. 13, pp. 69-73, 1990.

[7] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., Prentice Hall, 2002.

[8] R. Hyatt, "The DTS high-performance parallel tree search algorithm", url: http://www.cis.uab.edu/info/faculty/hyatt/search.html (last access: 2013).

[9] T. Obata, T. Sugiyama, K. Hoki, and T. Ito, "Consultation algorithm in computer Shogi - a move decision by majority", Computers and Games, 7th International Conference (CG2010), Kanazawa, Japan, 2010, H.J. van den Herik, H. Iida, and A. Plaat Eds., Lecture Notes in Computer Science (LNCS) 6515, Springer, 2011, pp. 156-165.

[10] Y. Tsuruoka, D. Yokoyama, T. Chikayama, "Game-tree search algorithm based on realization probability", ICGA Journal, International Computer Games Association (ICGA), vol. 25, pp. 145-152, 2002.

[11] K. Hoki and M. Muramatsu, "Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and Late Move Reduction (LMR)", Entertainment Computing, Elsevier, vol. 3, pp. 51-57, 2012.

[12] K. Hoki, "Optimal control of minimax search results to learn positional evaluation", 11th Game Programing Workshop, Kanagawa, Japan, 2006, pp. 78–83 (in Japanese).

[13] T. Kaneko and T. Tanaka, "Distributed game-tree search based on prediction of best moves", IPSJ Journal, Information Processing Society of Japan (IPSJ), vol. 53, pp. 2517-2524, 2012 (in Japanese).

[14] C. Donninger and U. Lorenz, "The chess monster Hydra", Proc. of 14th international conference on field-programmable logic and applications (FPL), Antwerp, Belgium. Lecture Notes in Computer Science (LNCS) 3203, Springer-Verlag, 2004, pp. 927-932.

[15] R. Feldmann, P. Mysliwietz, and B. Monien, "A fully distributed chess program", Advances in Computer Chess 6, D.F. Beal Ed., Ellis Horwood Series in Artificial Intelligence, 1991.

[16] M.G. Brockington and J. Schaeffer, "The APHID parallel alpha-beta search algorithm", IEEE Symposium of Parallel and Distributed Processing (SPDP'96), New Orleans, USA, 1996.

[17] A. Kishimoto and J. Schaeffer, "Transposition table driven work scheduling in distributed game-tree search", 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence (AI 2002), Calgary, Canada, R. Cohen and B. Spencer Eds., Lecture Notes in Artificial Intelligence (LNAI) 2338, Springer-Verlag, 2002, pp. 56-68.

[18] K. Himstedt, U. Lorenz, D.P.F. Möller, "A twofold distributed game-tree search approach using interconnected clusters", 14th International Euro-Par Conference, Lecture Notes in Computer Science (LNCS) 5168, Springer, 2008, pp. 587-598.

[19] K. Himstedt, "GridChess: combining optimistic pondering with the young brothers wait concept", ICGA Journal, International Computer Games Association (ICGA), vol. 35, pp. 67-79, 2012.

[20] I. Althöfer, and R.G. Snatzke, "Playing games with multiple choice system", Computer and Games, third international conference, (CG2002), Edmonton, Canada, 2002, J. Schaeffer, M. Müller, and Y. Björnssonpp Eds., Lecture Notes in Computer Science (LNCS) 2883, Springer, 2003, pp. 142-153.

[21] S. Omori, K. Hoki, and T. Ito, "Performance Analysis of Consultation Methods in Computer Chess", The 2012 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2012), Tainan, Taiwan, 2012.

[22] Y. Soejima, A. Kishimoto, and O. Watanabe, "Evaluating root parallelization in Go", IEEE Transactions on Computational Intelligence and AI in Games, IEEE, vol. 2, pp. 278-287, 2010.

[23] T. Ito, "Consultation Player "Monju" - A Majority Rule to Increasing the Winning Rate", IPSJ Magazine, IPSJ Journal, Information Processing Society of Japan (IPSJ), vol. 50, pp. 887-894, 2009 (in Japanese).

[24] J. Markoff, "Computer wins on 'Jeopardy!': Trivial, It's Not", The New York Times, 2011, url: http://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html (last access 2013).

[25] D. Ferrucci *et al*., "Building watson: an overview of the DeepQA project", AI Magazine, Association for the Advancement of Artificial Intelligence (AAAI), vol. 31, pp. 59-79, 2010.

[26] Internet computer shogi server Floodgate, url: http://wdoor.c.u-tokyo.ac.jp/shogi/ (last access: 2013) (in Japanese).

[27] J. Takada, Shogi Framework Implements CSA Protocol, url: http://www.junichi-takada.jp/sficp/ (last access: 2013) (in Japanese).

[28] Utl: http://www.ipsj.or.jp/50anv/shogi/20101012-2.html (last access: 2013)