ELEKTROTEHNIŠKI VESTNIK 78(3): 153–158, 2011
ENGLISH EDITION

Chess Program Umko

Borko Bošković, Janez Brest

University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, 2000 Maribor, Slovenia

E-mail: borko.boskovic@uni-mb.si

Abstract. Umko is a strong open-source chess program developed to collect good concepts from literature and other open-source projects. Using these concepts, we want to implement an optimally chess program. To do this, Umko has implemented a bitboard representation, move generator, parallel search algorithm, multiple principal variation search, transposition table, universal chess interface, evaluation function, usage of endgame tablebases and usage of the opening book. The paper provides details of these concepts.

Umko is a program running on several platforms inside different graphical user interfaces and using the modern processor technology. It has a parallel search algorithm allowing its program to simultaneously use more processors or cores and the new SSE4.2 CPU instruction set. Both the parallel search algorithm and the new instruction set enable the program to be a faster and stronger player. Having been tested on different independent rating lists, the program is rated among the top ten open-source chess programs.

Key words: chess program, bitboard representation, evaluation function, search algorithm, move generator, transposition table, endgame tablebases, opening book

1 INTRODUCTION

Computer chess has a long history of research in the field of artificial intelligence. With computer strength ever growing to a remarkable degree, we are witnessing more and more matches between computers and humans where computers usually win. The reasons for this are mainly hardware improvements and chess algorithm optimizations. The first computer that won against a human world champion chess player was Deep Blue which defeated Garry Kasparov in 1997 [12]. In 2006 the Deep Fritz 10 computer program which runs on a PC, defeated world champion Vladimir Kramnik [18]. In 2005 the Rybka chess program surprised everyone with its strength. It was top-rated on all notable chess program rating lists and the first chess program that obtained a rating over 3000 rating points [9]. Moreover, open-source and free programs are becoming stronger, some of them even stronger than their commercial counterparts. Currently, Stockfish is the top open-source chess program that has been rated over 3200 rating points on most chess rating lists. In 2010 the free chess program Houdini became a new top ranked. Its strength was approximately 50 rating points ahead of Rybka and 70 rating points ahead of Stockfish.

So why are developers trying to improve the already very strong chess programs even further? Many professional human chess players use chess programs to improve their own playing skills. Chess programs are also very useful in correspondence and freestyle chess. Matches between programs are also gaining popularity. As far as artificial intelligence is concerned, chess is regarded as a very useful environment for testing different approaches.

We, too, want to develop a strong chess program. In order to do this, we collected good concepts from literature and open-source projects. According to these sources, Umko has implemented a bitboard representation, move generator, parallel search algorithm, multiple principal variation search, transposition table, universal chess interface, evaluation function, usage of endgame tablebases inside the search algorithm and usage of the opening book. It has a parallel search algorithm allowing the program to use more processors or cores at the same time. On the other hand, it can use the new SSE4.2 CPU instruction set. Both the parallel search algorithm and the new instruction set enable the program to use the new processor technology and become a faster and stronger player.

Our program, being based on open-source projects, is published under the GNU General Public License version 3 on the following SourceForge web sites http://umko.sourceforge.net/. SourceForge is one of the largest open-source software development web site. It provides free services that help people to build software and share it with a global audience. Umko is implemented with the C++ programming language thus enabling us to compile it with the GNU C++ compiler for Linux, Android, Mac OS X and Windows operating systems for the i586, x64 and ARM architectures.

Received 6 June 2010
Accepted 15 July, 2011

The paper is organized as follows. Introduction is following by Section 2 describing our chess program in more detail. Our experimental results and obtained ratings are presented in Section 3 and our conclusions are given in Section 4.

2 Description of the program

Umko does not have its own graphical user interface (GUI). It is a console application that communicates with a GUI via a standard universal chess interface (UCI). The GUI runs our program as an external process and according to the UCI protocol with standard I/O streams it manages our program. To communicate with the GUI and to analyze a specific position at the same time, our program has at least two threads. One is responsible for communication while the other one or more of them are used for analyzing specific positions and searching for the best move. Wanting to develop a program for Windows and Unix-based operating systems, we use the POSIX and Windows threads. At the compile time, the compiler detects the target operating system according to which, the corresponding threads or libraries are used to build a program.

The basic components of modern chess programs are: representation of the game, search algorithm, move generator, transposition table, evaluation function, opening book, and endgame tablebases. Representation of the game enables the program to manage positions and moves. It also affects the speed of a chess program. The move generator is a mechanism that generates legal moves as fast as possible. The search algorithm is responsible for finding the best move according to the evaluation-function values. The evaluation function contains chess knowledge which enables it to evaluate positions. The transposition table (TT) enables the search algorithm to avoid multiple searches of the same position. The opening book enables the chess program to use knowledge and experiences of human and computer chess players in the opening-phase game. Similarly, endgame tablebases enable strong move choices in the final stage of the game.

Chess games are time-limited. This means that in deciding which move is the best in a specific position, the chess program is time-limited. Therefore, the chess program has to be as fast as possible thus enabling the search algorithm to examine more positions in a given time and consequently the program becomes a better chess player. On the other hand, the chess program contains chess knowledge that also affects the speed. More knowledge makes the program slower and vice versa, less knowledge makes the program faster. Therefore, chess program developers have to balance the amount of chess knowledge and the speed to get a strong chess program.

2.1 *Game representation*

The game representation enables the chess program to manage positions and moves. It affects the speed of the move generator, evaluation function and generally the speed of almost any other component. There are many different ways of representing positions and moves. A chess position has information about the piece placement, player on turn, castling availability, en-passant target square, number of half-moves since the last pawn advanced or piece captured and number of all moves that have been played to reach this position. The most important for game representation is the piece placement or board representation. To represent the board, we used the bitboard data structure. Bitboard is a 64-bit unsigned integer where each bit presents a square of a chess board. Because chess contains different pieces, a set of bitboards is needed, one bitboard for each piece-type and color. The main advantage of the bitboard is its enabling fast calculation of moves and evaluation function expressions [11], [19], [4], [3].

Move representation is also important. Moves change positions and as such they are used inside the search algorithm where they are also stored in a transposition table. Therefore, move representation must be compact. We used a 16-bit unsigned integer for move representation. Inside this integer, the following data are coded: move type, square from and square to. According to these data, the search algorithm can do and undo moves. In the chess program, it is also important to calculate a unique number for each position that enables comparison between them. To do this, we used the Zobrist keys [24] that are almost unique numbers for any chess position. These keys are also used in the transposition table and opening book.

2.2 *Search algorithm*

Due to the complicated nature of the chess game, there is no way to make a program that will play a perfect chess. However, it is possible to develop a search algorithm and an evaluation function [23] which together enable a chess program to become a strong chess player. The evaluation function is responsible for evaluating positions statically while the search algorithm is responsible for evaluating positions dynamically and selecting moves which will be played.

Generally, search algorithms of chess programs are based on the minimax or alpha-beta algorithm. In our program, we implemented the Principal Variation Search (PVS) algorithm [17], [2], [3] which is an improved version of the alpha-beta algorithm. Its main idea is to separate search nodes (positions) to the principal variation nodes (PV-nodes) and non PV-nodes. The PV-nodes are searched with a specific window while the non PV-nodes are searched with the minimal window. The window is defined with the alpha and beta arguments. For a minimal window difference between these arguments is one. We used this algorithm for enabling different pruning techniques between the PV-nodes and the non PV-nodes. The principal variation will probably be played and it is better to use less aggressive pruning techniques under the PV-nodes and more aggressive pruning techniques under the non PV-nodes.

Quiescence search (QS) is used to evaluate leaf nodes of the search tree. The purpose of this technique is to avoid the horizon effect [22]. This means that if the algorithm stopped searching when reaching the leaf node and if its position were evaluated, the evaluation value might be considerably erroneous. This can happen when the last move in the search tree branch is a capture move and the next possible move is a recapture. Therefore, in leaf nodes the quiescence search continues to search and tries to avoid the horizon effect and to evaluate the obtained position correctly.

Extensions and pruning techniques implemented in our search algorithm are: futility pruning [10], null move pruning [6], late move reductions [16], transposition table [5], singular extensions [1], internal iterative deepening, single move extensions, check extensions, passed pawns extensions and recapture extensions. Futility pruning is a technique that prunes nodes close to leaf nodes of the tree search according to their evaluation values. Null move search makes a null move (changes the player on the move) and by minimising the search depth recognizes the nodes that can be pruned. Late move reductions is a technique that increases reduction in the depth for quiet moves that are closer to the end of the generated moves. The transposition table stores the search data and enables the search algorithm to avoid multiple searches of the same position. Internal iterative deepening is a technique similar to iterative deepening but it is used inside the search tree nodes. Singular extensions is an extension technique when one move seems to be a lot better than all other moves. Single move extensions and check extensions are extensions when only one legal move exists or king is in check. All other extensions passed pawn extensions and recapture extensions are self-explained.

The whole algorithm was implemented together with iterative deepening, aspiration search and root search. Iterative deepening is a technique that enables the program to iteratively increment the search depth [15]. This technique together with other chess program techniques, such as transposition table and history heuristic, iteratively improve move ordering. Although some nodes are examined repeatedly, efficiency of the whole search algorithm is improved because the alpha-beta algorithm is sensitive to move ordering. The aspiration search is a technique that reduces the search space [14]. Instead of searching the entire search space, the algorithm guesses the evaluation of a position and searches around this value with a specific window. If the evaluation is outside this window, then an examine must be made around the new guess value. This is repeated until the algorithm returns the evaluation in the searched window. The root search is a technique to search the root node. This search returns the evaluation of the search algorithm as well as the move which will be played. In our program, a multiple principal variation search is included into the root search. This means that the program can search more principal variations at the same time. The principal variation moves are searched as PV-nodes with less aggressive pruning and are shown in the GUI via the UCI protocol. This enables users to analyze more variants at the same time and get differences of their evaluations.

To increase the search speed or search depth by using additional processors or cores, our program implemented “Young Brothers Wait“ search [7], [8]. This is a parallel search algorithm. In its specific nodes the first move is searched without parallelization and after that the remaining moves can be searched in the parallel mode. Because in the parallel search threads use the transposition table at the same time, data can be corrupted. Therefore, the transposition table has to be adapted to avoid this problem [13].

The techniques used in our program were improved according to the open-source Toga II and Stockfish chess programs.

2.3 Move generator

As mentioned above, the search algorithm is very sensitive to move ordering. Good move ordering improves efficiency of the search algorithm and also the strength of the program. In our program we implemented a magic bitboard move generator which is a technique that generates moves for sliding pieces very fast [19]. With only a few instructions it already enables the program to calculate the index of a bitboard database with attacks of both lines for pieces bishop and rook stored. To improve the move ordering transposition table, static exchange evaluation, killer moves and history heuristic are used in the move generator.

The implemented move generator has four schemes that were taken from the Toga II chess program. Two schemes are used for the PVS-nodes and two schemes for QS-nodes. In the PVS-nodes, according to information of check, one of the schemes is used. If a position is not in check, the following scheme is used:

- transposition table move
- good capture (static exchange evaluation)
- killer moves
- quiet moves (history heuristic)
- bad captures (static exchange evaluation)

If a position is in check, the following scheme is used:

- transposition table move
- good evasion of check (static exchange evaluation)
- bad evasion of check (static exchange evaluation)

Similar schemes are used for the QS-nodes. If a position is not in check, the following scheme is used:

- transposition table move
- good captures (static exchange evaluation)
- check moves (only if search depth is 0)

If a position is in check, the following scheme is used:

- transposition table move
- all evasions of check

The first technique to improve move ordering is the transposition table. It is a hash table that contains some data about positions. As mentioned above, these data are used in the search algorithm to avoid a multiple search of the same position. The good moves of earlier searches are also stored in the transposition table. If a move exists in the transposition table for a specific position, it will be searched first and as such will improve move ordering. The second technique that was used in our program to improve move ordering is a static exchange evaluation. It separates good and bad captures. The static exchange evaluation is a technique evaluating the consequence of a series of exchanges on a single square for a specific move [20]. If this evaluation is greater than or equal to 0, the given move is a good capture, otherwise the given move is a bad capture. Killer moves are the third technique used in the move generator to improve move ordering. These moves were good quiet moves in nodes of earlier branches in the search tree with the same distance to the root. And the last technique that was used in the move generator to improve move ordering is a history heuristic [21]. This technique is used for dynamic ordering of quiet moves. Through the search, quiet moves are evaluated according to the number of cutoffs and its search depth. These evaluations are then used for ordering of quiet moves. The term cutoff, that was mentioned above, is a situation when a specific move is researched and the obtained evaluation is larger than the upper bound of the searched window (argument beta). This means the remaining moves do not need to be searched.

2.4 *Evaluation function*

The evaluation function is an important part of chess programs. It contains chess knowledge and statically evaluates positions. These evaluations are integer numbers used in the search algorithm to evaluate the root position. Our evaluation function takes most of its ideas from the Toga II chess program updated with some ideas from the Stockfish chess program. Therefore, our evaluation function discovers the following chess knowledge:

- simple endgame positions (KK, KNK, KBK, KRK, KQK, KNKN, KBKB, KRKR, KQKQ, KNNK, KBBK, KBNK, KBKN, KRRK, KQQK, KQRK)
- patterns (trapped and blocked pieces)
- pieces (material, mobility, position values)
- king (storm of pawns, shelter squares, attack of pieces)
- pawn structure (chain of pawns, doubled pawns, isolated pawns, backward pawns, candidates for promotion, passed pawns, unstoppable passed pawns)
- threats (attack of pieces)

During evaluation, this knowledge is calculated for opening and endgame phases. At the end of evaluation, the final evaluation is calculated using interpolation according to the game phase.

2.5 *Opening book and endgame tablebases*

A program that uses the opening book and endgame tablebases is a stronger chess player. The opening book contains human and chess program experiences and knowledge of the opening game phase. That enables the program to play stronger opening moves quickly. Our chess program can use the Polyglot opening books. These books for specific positions offer possible moves and their corresponding probabilities. That means the program selects offered moves with specific probabilities and plays different opening variants. Without the opening book, the program would play only a few opening variants and it would be predictable.

Just as the opening book improves the strength of the chess program in the opening game phase, the endgame tablebases improve its strength in the endgame phase. Those tablebases present databases of endgame positions and enable the program to evaluate some positions perfectly and consequently play stronger moves in the endgame phase. In our program, the Gaviota's tablebases are used. Therefore, our chess program is able to play perfect moves for positions which have only five or less pieces. The tablebases are also used in the search algorithm when the root position has more than five pieces. This can happen if positions in the search tree nodes have five or less pieces. In this way, tablebases also help the search algorithm to select stronger moves, even when positions contain more than five pieces.

3 Results

3.1 *Experimental results*

Umko was also tested by solving test-position suites. These tests were done on an Intel(R) Core(TM)2 CPU 6600 2.40GHz processor. The operating system was Linux and two threads were used for the search algorithm. The obtained results are shown in Table 1. The first column shows names of test suites, the second column shows the available time for solving the problem in seconds, the third column shows the size and usage of the transposition table. The average speed (the number of nodes per second) is shown in the fourth column. The obtained average search depth and average selective depth (the maximum number of half moves from the root

Table 1. Test-position suites (Umko 1.1b).

Test Suites	Time	Transposition Table	Speed	Depth	Solved	Rating time	Rating
Win at Chess	5	64 (70.7%)	1.74e+06	26.1 (36.2)	294/300	54.8	-
1001 Winning Chess Sacrifices	5	64 (80.1%)	1.73e+06	21.7 (37.9)	872/1001	738.3	-
1001 Brilliant Ways to Checkmate	5	64 (7.8%)	1.34e+06	55.1 (16.1)	988/1001	89.4	-
Strategic Test Suite	10	128 (94.1%)	1.63e+06	18.0 (40.1)	996/1300	3752.6	-
Encyclopedia of Chess Middlegame	20	128 (94.0%)	1.68e+06	20.8 (47.0)	677/770	2429.0	-
Bratko-Kopec	60	256 (95.6%)	1.56e+06	23.8 (46.8)	18/24	404.0	-
LCT II	600	512 (98.0%)	1.75e+06	27.4 (65.6)	29/35	3777.3	2740
BT 2630	900	512 (96.7%)	1.67e+06	29.8 (69.7)	27/30	2889.4	2534
BS 2830	900	512 (96.1%)	1.54e+06	26.1 (71.6)	19/27	7847.6	2771
Nolot	3600	512 (99.8%)	1.54e+06	27.8 (83.4)	6/11	18935.7	-

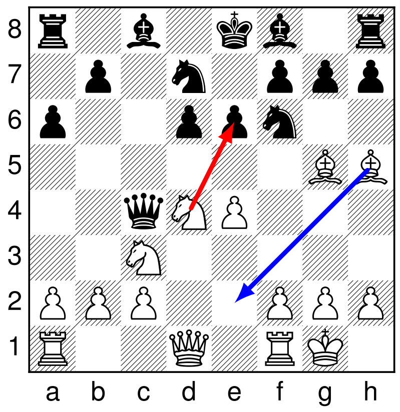

Figure 1. Bronstein - Kotov, Budapest 1950.

Table 2. Position analysis (Nolot: problem 4).

Depth	Evaluation	Time	Speed	TT	Move
23 (43)	0.83	315	1.48e+6	99.8	h5e2
23 (50)	1.04	370	1.51e+6	99.8	d4e6
23 (52)	1.78	488	1.50e+6	99.8	d4e6
29 (65)	2.11	3003	1.47e+6	99.8	d4e6

node to the leaves) are shown in column five. The sixth column shows the number of the solved positions and the number of all positions in the test suite. The last two columns show the rating time and the obtained rating. Specific test suites according to the rating time and the number of the solved positions enable rating calculation.

The way that the program works is shown the Table 2. Position 4 (Figure 1) from the Nolot chess suite was analysed. The first column in the table shows the search depth and the selective search depth. The remaining columns show evaluation, time in seconds, search speed (million nodes per second), usage of transposition table (percent), and selected move. As seen from the obtained results, the program found the best move when the search depth was 23. Evaluation was 1.04 or advantage of one pawn for white. Reaching this move takes 370 seconds. The search speed was 1.5 million nodes per second and the usage of the transposition table was 99.8%. The final search depth was 29 and evaluation was increased to 2.11 (the value of two pawns).

3.2 *Obtained rating*

Umko was tested and obtained different ratings on different independent rating lists as shown in Table 3.1. In this table, ratings with a 95% confidence interval are shown. CCRL (Computer Chess Rating Lists) is a club of people inspired by watching computers play chess. They also compare the strengths of different chess programs. CCRL has two lists: 40/40 and 40/4. On the first list 40 moves in 40 minutes repeatedly and on the second list 40 moves in 4 minutes repeatedly time control is used. Time control on both lists is adjusted to the AMD64 X2 4600+ (2.4GHz). On both rating lists, Umko is ranked as the 25th (June, 2011), while within open-source programs it is ranked as the 6th on the 40/4 rating list and as the 7th on the 40/40 rating list.

Similar to CCRL, CEGT (Chess Engines Grand Tournament) is a team that has fun to test chess programs and share their results and information to all chess enthusiasts with two lists: 40/4 and 40/20. IPON is a rating list where programs play in a pondering mode (thinking during its opponent's time) and time control is five minutes per game plus three seconds per move. On this rating list Umko with the SSE4.2 instruction set was tested and was ranked as the 20th.

Umko also plays games on an Internet Chess Server (freechess.org) against humans and computers from all over the world. On this server, Umko obtained blitz rating of 2416 in 858 games, standard rating of 2500 in 911 games and lightning rating of 2619 in 155 games. These ratings are obtained on the same computer as mentioned above for solving test suites.

During this section, ratings of the Umko chess pro-

Table 3. Obtained ratings on different rating lists.

Rating list	Program	Rating
CCRL 40/4	Umko 1.1 64-bit 2CPU	2912±17
CCRL 40/40	Umko 1.0 64-bit	2908±29
CEGT 40/20	Umko 1.0 x64 4CPU	2848±62
CEGT 40/4	Umko 1.0 x64 1CPU	2811±13
IPON	Umko 1.1 SSE42	2635±11

gram are shown and they are a bit different, because rating is a relative strength of chess players. From the presented results you can see three different versions of Umko that were tested. All versions have similar strength and the main differences are a few minor improvements.

4 Conclusion

The paper presents the best Slovenian chess program named Umko. It has implemented concepts and ideas that were taken from literature or open-source projects. It also uses technologies that have been included in modern processors. It is able to use more cores or processors at the same time and to use the new SSE4.2 processor instruction set. It is developed as an open-source project on the SourceForge web system. It is freely available and can run on many platforms. We tested it on the Linux, Android, Mac OS X and Windows operating systems. According to the obtained results, it is a strong chess player. It was ranked on independent rating lists among the top 25 chess programs and among the top 10 open-source chess programs. It is also interesting that Umko outperforms some commercial chess programs such as the well-known Chessmaster 11 chess program.

In future, we will try to add more chess knowledge to the evaluation function and optimise the parallel search algorithm. In order to improve the strength of the whole chess program. We will also take an effort to add new features that will enable the program to play with different strengths. This way the program will be interesting and useful for different chess players, from beginners to the grand masters.

Acknowledgements

The authors would like to thank Thomas Gaksch and Fabien Letouzey for the open-source chess program Toga II, Tord Romstad, Marco Costalba, Joona Kiiski for open-source chess program Stockfish, Miguel A. Ballicora for endgame tablebases, Fabien Letouzey for ”UCI adapter“ Polyglot, CCRL, CEGT and IPON teams and SourceForge web system.

References

[1] Thomas Anantharaman, Murray S. Campbell in Feng-hsiung Hsu. Singular extensions: adding selectivity to brute-force searching. *Artificial Intelligence*, 43:99–109, 1990.

[2] Yngvi Björnsson. *Selective Depth-First Game-Tree Search*. doktorska disertacija, University of Alberta, 2002.

[3] B. Bošković. Implementacija računalniškega šaha, 2004.

[4] B. Bošković, S. Greiner, J. Brest in V. Žumer. The Representation of Chess Game. V *Proceedings of the 27th International Conference on Information Technology Interfaces*, str. 381–386, 2005.

[5] D.M. Breuker, J. W. H. M. Uiterwijk in H. J. Van Den Herik. Replacement Schemes for Transposition Tables. *ICCA Journal*, 17:183–193, 1994.

[6] Omid David-Tabibi in Nathan Netanyahu. Extended Null-Move Reductions. V H. van den Herik, Xinhe Xu, Zongmin Ma in Mark Winands, uredniki, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, str. 205–216. Springer Berlin / Heidelberg, 2008.

[7] R. Feldmann, P. Mysliwietz in B. Monien. A Fully Distributed Chess Program. *Advances in Computer Chess 6*, 1991.

[8] Rainer Feldmann. *Game Tree Search on Massively Parallel Systems*. doktorska disertacija, 1993.

[9] Harald Fietz. Beyond the 3000 Elo barrier A glance behind the scenes of the Rybka chess engine. *Chess*, str. 18–21, 2007.

[10] E.A. Heinz. Extended Futility Pruning. 21(2):75–83, 1998.

[11] Ernst A. Heinz. How DarkThought Plays Chess. *ICCA Journal*, (3):166–176, 1997.

[12] Feng-hsiung Hsu. IBM's Deep Blue Chess Grandmaster Chips. *IEEE Micro*, 19:70–81, 1999.

[13] Robert M. Hyatt in Timothy Mann. A lockless Transposition-Table Implementation for Parallel Search. *ICGA Journal*, 25(1):36–39, 2002.

[14] Hermann Kaindl, Reza Shams in Helmut Horacek. Minimax Search Algorithms With and Without Aspiration Windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13:1225–1235, December 1991.

[15] Richard E. Korf. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97–109, 1985.

[16] D. Levy, D. Broughton in M Taylor. The SEX algorithm in Computer Chess. *ICCA Journal*, 12(1):10–21, 1989.

[17] T. A. Marsland in M. Campbell. Parallel Search of Strongly Ordered Game Trees. *ACM Computing Surveys*, 14:533–551, 1982.

[18] Dylan Loeb Mcclain. Once Again, Machine Beats Human Champion at Chess. *New York Times*, 2006.

[19] Fritz Reul. *New Architectures in Computer Chess*. doktorska disertacija, Tilburg University, 2009.

[20] Fritz Reul. Static Exchange Evaluation with $\alpha\beta$-Approach. *ICGA Journal*, 33(1):3–17, 2010.

[21] Jonathan Schaeffer. The History Heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[22] Günther Schrüfer. A Strategic Quiescence Search. *ICCA Journal*, 12:3–9, 1989.

[23] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256, 1950.

[24] Albert Zobrist. A New Hashing Method with Application for Game Playing. *ICCA Journal*, 13(2):69–73, 1970.

Borko Bošković received BS and PhD degrees in computer science from the University of Maribor, Maribor, Slovenia, in 2004 and 2010. He is currently a teaching assistant at the Faculty of Electrical Engineering and Computer Science, University of Maribor. His research is focused on chess algorithms and evolutionary computing. His areas of expertise also include programming languages, integrative programming and natural language processing.

Janez Brest received BS, MSc, and PhD degrees in computer science from the University of Maribor, Maribor, Slovenia, in 1995, 1998, and 2000, respectively. He is currently a full professor at the Faculty of Electrical Engineering and Computer Science, University of Maribor. His research interests include evolutionary computing, artificial intelligence, and optimization. His fields of expertise embrace also programming languages, web oriented programming, parallel and distributed computing research.