Simulating Human Grandmasters: Evolution and Coevolution of Evaluation Functions

ABSTRACT

This paper demonstrates the use of genetic algorithms for evolving a grandmaster-level evaluation function for a chess program. This is achieved by combining supervised and unsupervised learning. In the supervised learning phase the organisms are evolved to mimic the behavior of human grandmasters, and in the unsupervised learning phase these evolved organisms are further improved upon by means of coevolution.

While past attempts succeeded in creating a grandmaster-level program by mimicking the behavior of existing computer chess programs, this paper presents the first successful attempt at evolving a state-of-the-art evaluation function by learning only from databases of games played by humans. Despite the underlying difficulty involved (in comparison to learning from chess programs), our results demonstrate that the evolved program outperforms a two-time World Computer Chess Champion.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*

General Terms

Algorithms

Keywords

Computer chess, Fitness evaluation, Games, Genetic algorithms, Parameter tuning

1. INTRODUCTION

Despite the many advances in Machine Learning and Artificial Intelligence, there are still areas where learning mechanisms have not yielded performance comparable to humans. Computer chess is a prime example of the difficulties in such fields.

It is well-known that computer games have served as an important testbed for spawning various innovative AI techniques in domains and applications such as search, automated theorem proving, planning, and learning. In addition, the annual World Computer Chess Championship (WCCC) is arguably the longest ongoing performance evaluation of programs in computer science, which has inspired other well-known competitions in robotics, planning, and natural language understanding.

Computer chess, while being one of the most researched fields within AI, has not lent itself to the successful application of conventional learning methods, due to its enormous complexity. Hence, top chess programs still resort to manual tuning of the parameters of their evaluation function, typically through years of trial and error. The evaluation function assigns a score to a given chess position and is thus the most critical component of any chess program.

Currently, the only successful attempt reported on automatic learning of the parameter values of an evaluation function is based on "mentor-assisted" evolution [12]. This approach evolves the parameter values by mimicking the evaluation function of an available chess program that serves as a "mentor". It essentially attempts to reverse engineer the evaluation function of this program by observing the scores it issues for a given set of positions. The approach relies heavily on the availability of the numeric evaluation score of each position, which is provided by the reference program.

In this paper, we deal successfully with a significantly more difficult problem, namely that of evolving the parameter values of an evaluation function by relying solely on the information available from games of human grandmasters, i.e., the moves played. Lacking any numeric information provided typically by a standard chess program, we combine supervised and unsupervised learning. The organisms are first evolved to mimic the behavior of these human grandmasters by observing their games, and the best evolved organisms are then further evolved through coevolution. The results show that our combined approach efficiently evolves the parameters of interest from randomly initialized values to highly tuned ones, yielding a program that outperforms a two-time World Computer Chess Champion.

In Section 2 we review past attempts at applying evolutionary techniques in computer chess. We also compare alternative learning methods to evolutionary methods, and argue why the latter are more appropriate for the task in question. Section 3 presents our new approach, including a detailed description of the framework of the GA as applied to the current problem. Section 4 provides our experimental results, and Section 5 contains concluding remarks and suggestions for future research.

2. LEARNING IN COMPUTER CHESS

While the first chess programs could not pose a challenge to even a novice player, the current advanced chess programs are on par with the strongest human chess players, as the recent man vs. machine matches clearly indicate. This improvement is largely a result of deep searches that are possible nowadays, thanks to both hardware speed and improved search techniques. While the search depth of early chess programs was limited to only a few plies, nowadays tournament-playing programs easily search more than a dozen plies in middlegame, and tens of plies in late endgame.

Despite their groundbreaking achievements, a glaring deficiency of today's top chess programs is their severe lack of a learning capability (except in most negligible ways, e.g., "learning" not to play an opening that resulted in a loss, etc.). In other words, despite their seemingly intelligent behavior, those top chess programs are mere brute-force (albeit efficient) searchers that lack true underlying intelligence.

2.1 Conventional vs. Evolutionary Learning in Computer Chess

During more than fifty years of research in the area of computer games, many learning methods such as reinforcement learning [29] have been employed in simpler games. Temporal difference learning has been successfully applied in backgammon and checkers [26, 30]. Although temporal difference learning has also been applied to chess [4], the results showed that after three days of learning, the playing strength of the program was merely 2150 Elo (see Appendix B for a description of the Elo rating system), which is a very low rating for a chess program. Wiering [32] provided formal arguments for the failure of these methods in more complicated games such as chess.

The issue of learning in computer chess is basically an optimization problem. Each program plays by conducting a search, where the root of the search tree is the current position, and the leaf nodes (at some predefined depth of the tree) are evaluated by some static evaluation function. In other words, sophisticated as the search algorithms may be, most of the knowledge of the program lies in its evaluation function. Even though automatic tuning methods, that are based mostly on reinforcement learning, have been successfully applied to simpler games such as checkers, they have had almost no impact on state-of-the-art chess engines. Currently, all top tournament-playing chess programs use hand-tuned evaluation functions, since conventional learning methods cannot cope with the enormous complexity of the problem. This is underscored by the following four points.

(1) The space to be searched is huge. It is estimated that there are about 10^{46} possible positions that can arise in chess [11]. As a result, any method based on exhaustive search of the problem space is infeasible.

(2) The search space is not smooth and unimodal. The evaluation function's parameters of any top chess program are highly co-dependent. For example, in many cases increasing the values of three parameters will result in a worse performance, but if a fourth parameter is also increased, then an improved overall performance would be obtained. Since the search space is not unimodal, i.e., it does not consist of a single smooth "hill", any gradient-ascent algorithm such as hill climbing will perform poorly. In contrast, genetic algorithms are known to perform well in large search spaces which are not unimodal.

(3) The problem is not well understood. As will be discussed in detail in the next section, even though all top programs are hand-tuned by their programmers, finding the best value for each parameter is based mostly on educated guessing and intuition. (The fact that all top programs continue to operate in this manner attests to the lack of practical alternatives.) Had the problem been well understood, a domain-specific heuristic would have outperformed a general-purpose method such as GA.

(4) We do not require a global optimum to be found. Our goal in tuning an evaluation function is to adjust its parameters so that the overall performance of the program is enhanced. In fact, a unique global optimum does not exist for this tuning problem.

In view of the above points it seems appropriate to employ GA for automatic tuning of the parameters of an evaluation function. Indeed, at first glance this appears like an optimization task, well suited for GA. The many parameters of the evaluation function (bonuses and penalties for each property of the position) can be encoded as a bit-string. We can randomly initialize many such "chromosomes", each representing one evaluation function. Thereafter, one needs to evolve the population until highly tuned "fit" evaluation functions emerge.

However, there is one major obstacle that hinders the above application of GA, namely the fitness function. Given a set of parameters of an evaluation (encoded as a chromosome), how should the fitness value be calculated? For many years, it seemed that the solution was to let the individuals, at each generation, play against each other a series of games, and subsequently record the score of each individual as its fitness value. (Each "individual" is a chess program with an appropriate evaluation function.)

The main drawback of this approach is the unacceptably large amount of time needed to evolve each generation. As a result, severe limitations were imposed on the length of the games played after each generation, and also on the size of the population involved. With a population size of 100 and a limit of 10 seconds per game, and assuming that each individual plays each other individual once in every generation, it would take 825 minutes for each generation to evolve. Specifically, reaching the 100th generation would take up to 57 days. As we see in the next section, past attempts at applying this process resulted in weak programs, which were far inferior to state-of-the-art programs.

In Section 3 we present our GA-based approach for using GA in evolving state-of-the-art chess evaluation functions. Before that, we briefly review previous work of applying evolutionary methods in computer chess.

2.2 Previous Evolutionary Methods Applied to Chess

Despite the abovementioned problems, there have been some successful applications of evolutionary techniques in computer chess, subject to some restrictions. Genetic programming was successfully employed by Hauptman and Sipper [17, 18] for evolving programs that can solve Mate-in-N problems and play chess endgames.

Kendall and Whitwell [21] used evolutionary algorithms for tuning the parameters of an evaluation function. Their approach had limited success, due to the very large number of games required (as previously discussed), and the small number of parameters used in their evaluation function. Their evolved program managed to compete with strong programs only if their search depth (lookahead) was severely limited.

Similarly, Aksenov [2] employed genetic algorithms for evolving the parameters of an evaluation function, using games between the organisms for determining their fitness. Again, since this method required a very large amount of games, it evolved only a few parameters of the evaluation function with limited success. Tunstall-Pedoe [31] also suggested a similar approach, without providing an implementation.

Gross *et al.* [16] combined genetic programming and evolution strategies to improve the efficiency of a given search algorithm using a distributed computing environment on the Internet.

David-Tabibi, Koppel, and Netanyahu [12] used "mentor-assisted" evolution for reverse engineering the evaluation function of a reference chess program (the "mentor"), thereby evolving a new comparable evaluation function. Their approach takes advantage of the evaluation score of each position considered (during the training phase), that is provided by the reference program. In fact, this numeric information is key to simulating the program's evaluation function. In other words, notwithstanding the high-level performance of the evolved program, the learning process is heavily dependent on the availability of the above information.

In this paper, we combine supervised evolution and unsupervised coevolution for evolving the parameter values of the evaluation function to simulate the moves of a human grandmaster, without relying on the availability of evaluation scores of some computer chess program. As will be demonstrated, the evolved program is on par with today's strongest chess programs.

3. EVOLUTION AND COEVOLUTION OF EVALUATION FUNCTIONS

Encoding the parameters of an evaluation function as a chromosome is a straightforward task, and the main impediment for evolving evaluation functions is the difficulty of applying a fitness function (a numerical value representing how well the organism performs). However, as previously noted, establishing the fitness evaluation by means of playing numerous games between the organisms in each generation (i.e., single-population coevolution) is not practical.

As mentioned earlier, the fitness value in mentor-assisted evolution is issued as follows. We run both an organism and a grandmaster-level chess program on a given set of positions; for each position the difference between the evaluation score computed by the organism and that computed by the reference program is recorded. We take the fitness value to be inversely proportional to this difference.

In contrast, no evaluation scores of any chess program are assumed available in this paper, and we only make use of (widely available) databases of games of human grandmasters. The task of evolution, in this case, is thus significantly more difficult than that based on an existing chess program, as the only information available here consists of the actual moves played in the positions considered.

The evaluation function is evolved by learning from grandmasters according to the steps shown in Figure 1.

1. Select a list of positions from games of human grandmasters. For each position store the move played.
2. For each position, let the organism perform a 1-ply search and store the move selected by the organism.
3. Compare the move suggested by the organism with the actual move played by the grandmaster. The fitness of the organism will be the total number of "correct" moves selected (where the organism's move is the same as the grandmaster's move).

Figure 1: Fitness function for supervised evolution of evaluation functions.

Although performing a search for each position appears to be a costly process, in fact it consumes little time. Conducting a 1-ply search amounts to less than a millisecond for a typical chess program on an average machine, and so one thousand positions can be processed in one second. This allows us to use a large set of positions for the training set.

The abovementioned process, which will be discussed below in greater detail, results in a grandmaster-level evaluation function (see next section). Due to the random initialization of the chromosomes, each time the above process is applied, a different "best evolved organism" is obtained. Comparing the best evolved organisms from different runs, we observe that even though they are of similar playing strength, their evolved parameter values differ, and so does their playing style.

After running the supervised evolution process a number of times, we obtain several evolved organisms. Each organism is the best evolved organism from one complete run of the evolutionary process. We next use a coevolution phase for further improving upon the obtained organisms. During this single-population coevolution phase the evolved organisms play against each other, and the fitness function applied is based on their relative performance. Completing this phase for a predetermined number of generations, the best evolved organism is selected as the best overall organism. According to the results in the next section, this "best of best" organism improves upon the organisms evolved from the supervised phase. As noted before, previous attempts at applying coevolution have failed to produce grandmaster-level evaluation functions. The difference here is that the population size is small (we used 10), and the initial organisms are already well tuned (in contrast to randomly initialized).

In the following subsections, we describe in detail the chess program, the implementation of the supervised and unsupervised evolution, and the GA parameters used.

3.1 The Chess Program and the Evaluation Function

Our chess program uses NegaScout/PVS [9, 23] search, in conjunction with standard enhancements such as null-move pruning [5, 13, 14], internal iterative deepening [3, 27], dynamic move ordering (history + killer heuristic) [1, 15, 24, 25], multi-cut pruning [7, 8], selective extensions [3, 6] (consisting of check, one-reply, mate-threat, recapture, and passed pawn extensions), transposition table [22, 28], and futility pruning near leaf nodes [19].

The evaluation function of the program (which we are interested in tuning automatically) consists of 35 parameters (see Figure 3). Even though this is a small number of parameters in comparison to other top programs, the set of parameters used does cover all important aspects of a position, e.g., material, piece mobility and centricity, pawn structure, and king safety.

The parameters of the evaluation function are represented as a binary bit-string (chromosome size: 224 bits), initialized randomly. The value of a pawn is set to a fixed value of 100, which serves as a reference for all other parameter values. Except for the four parameters representing the material values of the pieces, all the other parameters are assigned a fixed length of 6 bits per parameter. Obviously, there are many parameters for which 3 or 4 bits suffice. However, allocating a fixed length of 6 bits to all parameters ensures that *a priori* knowledge does not bias the algorithm in any way.

Note that the program's evaluation function is randomly initialized, i.e., other than knowing the rules of the game, the program has essentially no game skills at all at this point.

3.2 Supervised Evolution using Human Grandmaster Games

As indicated, our goal is to evolve the parameters of a program's evaluation function, so as to simulate the moves played by grandmasters for a given set of positions.

For our experiments, we use a database of 10,000 games by grandmasters of rating above 2600 Elo, and randomly pick one position from each game. We pick winning positions only, i.e., positions where the side to move ultimately won the game (e.g., if it is white's turn to move, the game was won eventually by white). Of these 10,000 positions, we select 5,000 positions for training and 5,000 for testing.

In each generation, for each organism we translate its chromosome bit-string to a corresponding evaluation function. For each of the N test positions (in our case, $N = 5,000$), the program performs a 1-ply search using the decoded evaluation function, and the best move returned from the search is compared to that of the grandmaster in the actual game. The move is deemed "correct" if it is the same as the move played by the grandmaster, and "incorrect" otherwise. The fitness of the organism is calculated as the square of the total number of correct moves.

Note that unlike the mentor-assisted approach for mimicking an existing chess program, which provides numeric values for each position, here we only have 1-bit of information for each processed position (correct/incorrect). This underscores why relying on human games is much more difficult than using computers as mentors.

Other than the special fitness function described above, we use a standard GA implementation with Gray coded chromosomes, fitness-proportional selection, uniform crossover, and elitism (the best organism is copied to the next generation). The following parameters are used:

population size = 100
crossover rate = 0.75
mutation rate = 0.005
number of generations = 200

3.3 Coevolution of the Best Evolved Organisms

Rerunning the supervised evolution ten times, we obtain ten "best organisms" corresponding to the various runs. The evaluation functions of these evolved organisms do not have the same evolved parameter values, since each run produces different results (due to the random initialization). Although the ten programs are of similar playing strength, their playing style is somewhat different. At any rate, the above ten best organisms are used for the coevolution phase described below. Note that selecting, instead, the top ten evolved organisms from one of the supervised runs is not desirable, as it could result in "interbreeding", in the sense that the parameter values of these organisms tend to be fairly similar.

Consider, on the other hand, generating multiple evolved organisms using different training sets for each run. Specifically, for each run we might pick games of a specific grandmaster, in the hope of obtaining organisms that mimic the individual styles of the various grandmasters. Preliminary tests suggest, however, that this variant provides no additional insight or improvement. Apparently, the 1-ply searches enable mimicking only a "generic" grandmaster style, rather than the style of a specific player.

In the coevolution phase, the ten best organisms selected serve as the initial population, which is then coevolved over

50 generations. In each generation, each organism plays four games against each other organism (to obtain a more reliable result). At the end of each generation, rank-based selection is applied for selecting the organisms for breeding. Elitism is used here as well, which ensures that the best organism survives for the next generation. This is especially critical in light of the small population size. Other GA parameters remain unchanged, that is, uniform crossover with crossover rate of 0.75 and mutation rate of 0.005.

In the following section we present our experimental results, both in terms of the learning efficiency and the performance gain of the best evolved individual.

4. EXPERIMENTAL RESULTS

We now present the results of running the evolutionary process described in the previous section. We also provide the results of several experiments that measure the strength of the evolved program in comparison to Crafty, a former two-time World Computer Chess Champion that is commonly used as a baseline for testing chess programs.

4.1 Results of Supervised Evolution

Figure 2 shows the number of positions solved (i.e., the number of correct moves found) for the best organism and the population average for 200 generations. Specifically, the results indicate that the average number of solved positions is about 800 (out of 5,000) in the first generation. Note that even without any chess knowledge an organism would occasionally select the correct move by random guessing. Additionally, since the randomly initialized parameters contain only positive values, an organism can find some basic captures of the opponent's pieces without possessing any real chess knowledge.

The average number of solved positions increases until stabilizing at around 1500, which corresponds to 30% of the positions. The best organism at generation 200 solves 1621 positions, which corresponds to 32.4% of the positions. Due to the use of elitism, the number of solved positions for the best organism is monotonously increasing, since the best organism is preserved. The entire 200-generation evolution took approximately 2 hours on our machine (see Appendix A).

At first glance, a solution rate of 32% might not seem too high. However, considering that the evolved organism selects successfully the "correct" move in one out of three cases, by applying merely a 1-ply search (as opposed to the careful analysis of a position by the grandmaster), this is quite satisfactory.

With the completion of the learning phase, we used the additional 5,000 positions set aside for testing. We let our best evolved organism perform a 1-ply search on each of these positions. The number of correctly solved positions was 1538 (30.7%). This indicates that the first 5,000 positions used for training cover most types of positions that can arise, as the success rate for the testing set is close to the success rate for the training set.

To measure the performance of the best evolved organism after the supervised evolution phase (we call this program

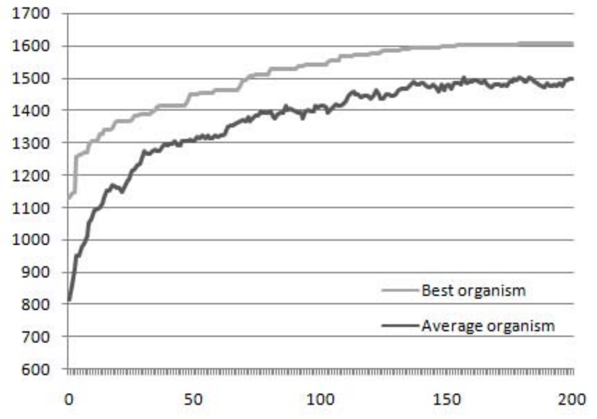

Figure 2: Number of positions solved (out of 5,000) for the best organism and the population average at each generation (total time for 200 generations ≈ 2 hours).

Evol*), we conducted a series of matches against the chess program Crafty [20]. Crafty has successfully participated in numerous World Computer Chess Championships (WCCC), and is a direct descendent of Cray Blitz, the WCCC winner of 1983 and 1986. It is frequently used in the literature as a standard reference.

Table 1 provides the results of 500 games between Evol* and Crafty. The results show that the evolved organism (Evol*) is on par with Crafty, clearly demonstrating that the supervised evolution has succeeded in evolving a grandmaster-level evaluation function by purely mimicking grandmaster moves.

Match	Result	W%	RD
Evol* - Crafty	254.5 - 245.5	50.9%	+6

Table 1: Results of the games between COEVOL* and CRAFTY (W% is the winning percentage, and RD is the Elo rating difference (see Appendix B)). Win = 1 point, draw = 0.5 point, and loss = 0 point.

4.2 Results of Coevolution

Repeating the supervised evolutionary process, we obtained each time a "best evolved organism" with a different set of evolved parameter values. That is, each run produced a different grandmaster-level program. Even though the performance of these independently evolved best organisms is fairly similar, our goal was to improve upon these organisms and create an enhanced "best of best" organism.

We applied single-population coevolution to enhance the performance of the program. After running the supervised evolution ten times (which ran for about 20 hours), ten different best organisms were obtained. Using these ten organisms as the starting population, we applied GA for 50 generations, where each organism played each other organism four times in every round. Each game was limited to ten seconds (5 seconds per side). In practice, this coevolution phase ran for approximately 20 hours.

Figure 3 provides the evolved values of the best individual obtained. Conventionally, knight and bishop are valued at 3 pawns, and rook and queen are valued at 5 and 9 pawns, respectively [10]. These values are used in most chess programs. However, it is well-known that grandmasters frequently sacrifice pawns for positional advantages, and that in practice, a pawn is assigned a lower value. Interestingly, the best organism assigns a pawn about half the value it is usually assigned, relative to the other pieces, which is highly unconventional for chess programs. This implies that the evolved organism, which learns its parameter values from human grandmasters, ends up adopting also their less materialistic style of play. This is also reflected in the playing style of the ultimate evolved program, as it frequently sacrifices pawns for gaining positional advantages.

```
PAWN_VALUE                          100
KNIGHT_VALUE                        523
BISHOP_VALUE                        568
ROOK_VALUE                          813
QUEEN_VALUE                        1699
PAWN_ADVANCE_A                        3
PAWN_ADVANCE_B                        6
PASSED_PAWN_MULT                     10
DOUBLED_PAWN_PENALTY                 14
ISOLATED_PAWN_PENALTY                 9
BACKWARD_PAWN_PENALTY                 4
WEAK_SQUARE_PENALTY                   6
PASSED_PAWN_ENEMY_KING_DIST           7
KNIGHT_SQ_MULT                        6
KNIGHT_OUTPOST_MULT                   9
BISHOP_MOBILITY                       4
BISHOP_PAIR                          24
ROOK_ATTACK_KING_FILE                48
ROOK_ATTACK_KING_ADJ_FILE             7
ROOK_ATTACK_KING_ADJ_FILE_ABGH       25
ROOK_7TH_RANK                        31
ROOK_CONNECTED                        5
ROOK_MOBILITY                         4
ROOK_BEHIND_PASSED_PAWN              38
ROOK_OPEN_FILE                       28
ROOK_SEMI_OPEN_FILE                  10
ROOK_ATCK_WEAK_PAWN_OPEN_COLUMN      15
ROOK_COLUMN_MULT                      6
QUEEN_MOBILITY                        2
KING_NO_FRIENDLY_PAWN                33
KING_NO_FRIENDLY_PAWN_ADJ             9
KING_FRIENDLY_PAWN_ADVANCED1          8
KING_NO_ENEMY_PAWN                   16
KING_NO_ENEMY_PAWN_ADJ                9
KING_PRESSURE_MULT                    4
```

Figure 3: Evolved parameters of the best individual.

We measured the performance of the best evolved organism after coevolution (we call this program COEVOL*) by conducting a series of matches against CRAFTY and also against EVOL*. Table 2 provides the results of 500 games between COEVOL* and EVOL*, and between COEVOL* and CRAFTY.

Match	Result	W%	RD
COEVOL* - CRAFTY	304.5 - 195.5	60.9%	+77
COEVOL* - EVOL*	293.0 - 212.0	58.6%	+60

Table 2: Results of the games of COEVOL* against CRAFTY and EVOL*.

The results demonstrate that the coevolution phase further improved the performance of the program, resulting in the superiority of COEVOL* to both CRAFTY and EVOL*.

In a second experiment, we measured the tactical strength of the programs using the Encyclopedia of Chess Middlegames (ECM) test suite, consisting of 879 positions. Each program was given 5 seconds per position. Table 3 provides the results. As can be seen, both EVOL* and COEVOL* solve significantly more problems than CRAFTY.

EVOL*	COEVOL*	CRAFTY
622	651	593

Table 3: Number of ECM positions solved by each program (time: 5 seconds per position).

The results of both tests establish that even though the parameters of our program are evolved from scratch (with chromosomes initialized randomly), the resulting organism substantially outperforms a grandmaster-level chess program.

5. CONCLUDING REMARKS AND FUTURE RESEARCH

In this paper we presented a novel approach for evolving grandmaster-level evaluation functions by combining supervised and unsupervised evolution. In contrast to the previous successful attempt which focused on mimicking the evaluation function of a chess program that served as a mentor, the approach presented in this paper focuses on evolving the parameters of interest by observing solely games of human grandmasters, where the only available information to guide the evolution consists of the moves made in these games.

Learning from games of human grandmasters in the supervised phase of the evolution, we obtained several grandmaster-level evaluation functions. Specifically, running the procedure ten times, we obtained ten such evolved evaluation functions, which served as the initial population for the second coevolution phase.

While previous attempts at using coevolution have failed due to the unacceptably large amount of time needed to evolve each generation, the use of coevolution succeeded in our case because the initial population was not random, but relatively well tuned due to the first phase of supervised evolution.

According to our experiments, organisms evolved from randomly initialized chromosomes to sets of highly tuned parameters. The coevolution phase further improved the performance of the program, resulting in an evolved organism which resoundingly defeats a grandmaster-level program. Note that this performance was achieved despite the fact that the evaluation function of the evolved program consists of a considerably smaller number of parameters than that of CRAFTY, of which the evaluation function consists of over 100 parameters.

In summary, we demonstrated how our approach can be used for automatic tuning of an evaluation function from scratch. Furthermore, the approach can also be applied for enhancing existing highly tuned evaluation functions. Starting from several sets of tuned parameter values of the evaluation function, the coevolution phase can be applied to refine these values, so as to further improve the evaluation function.

Running the supervised evolution phase ten times, together with coevolution, took a total of about 40 hours. Both the supervised and unsupervised phases can be easily parallelized for obtaining linear scalability. During the supervised evolution each organism can be evaluated independently on a different processor, without having to share any information with the other organisms. Also, during coevolution, multiple games can be run in parallel. In this work we ran the experiments on an average single processor machine. Running these tests on an 8-core processor (which is readily available today) would reduce the overall running time from 40 hours to as little as 5 hours.

Finally, the results presented in this paper point to the vast potential in applying evolutionary methods for learning from human experts. We believe that the approach presented in this paper for parameter tuning could be applied to a wide array of problems for essentially "reverse engineering" the knowledge of a human expert.

6. REFERENCES

[1] S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *Proceedings of the Fifth Annual ACM Computer Science Conference*, pages 466–473. ACM Press, Seattle, WA, 1977.

[2] P. Aksenov. *Genetic algorithms for optimising chess position scoring*. M.Sc. Thesis, University of Joensuu, Finland, 2004.

[3] T.S. Anantharaman. Extension heuristics. *ICCA Journal*, 14(2):47–65, 1991.

[4] J. Baxter, A. Tridgell, L. and Weaver. Learning to play chess using temporal-differences. *Machine Learning*, 40(3):243–263, 2000.

[5] D.F. Beal. Experiments with the null move. *Advances in Computer Chess 5*, ed. D.F. Beal, pages 65–79. Elsevier Science, Amsterdam, 1989.

[6] D.F. Beal and M.C. Smith. Quantification of search extension benefits. *ICCA Journal*, 18(4):205–218, 1995.

[7] Y. Bjornsson and T.A. Marsland. Multi-cut pruning in alpha-beta search. In *Proceedings of the First International Conference on Computers and Games*, pages 15–24, Tsukuba, Japan, 1998.

[8] Y. Bjornsson and T.A. Marsland. Multi-cut alpha-beta-pruning in game-tree search. *Theoretical Computer Science*, 252(1-2):177–196, 2001.

[9] M.S. Campbell and T.A. Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20(4):347–367, 1983.

[10] J.R. Capablanca. *Chess Fundamentals*, ed. N. de Firmian, Random House, Revised ed., 2006.

[11] S. Chinchalkar. An upper bound for the number of reachable positions. *ICCA Journal*, 19(3):181–183, 1996.

[12] O. David-Tabibi, M. Koppel, and N.S. Netanyahu. Genetic algorithms for mentor-assisted evaluation function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1469–1476. Atlanta, GA, 2008.

[13] O. David-Tabibi and N.S. Netanyahu. Extended null-move reductions. In *Proceedings of the 2008 International Conference on Computers and Games*, eds. H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, pages 205–216. Springer (LNCS 5131), Beijing, China, 2008.

[14] C. Donninger. Null move and deep search: Selective search heuristics for obtuse chess programs. *ICCA Journal*, 16(3):137–143, 1993.

[15] J.J. Gillogly. The technology chess program. *Artificial Intelligence*, 3(1-3):145–163, 1972.

[16] R. Gross, K. Albrecht, W. Kantschik, and W. Banzhaf. Evolving chess playing programs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 740–747. New York, NY, 2002.

[17] A. Hauptman and M. Sipper. Using genetic programming to evolve chess endgame players. In *Proceedings of the 2005 European Conference on Genetic Programming*, pages 120–131. Springer, Lausanne, Switzerland, 2005.

[18] A. Hauptman and M. Sipper. Evolution of an efficient search algorithm for the Mate-in-N problem in chess. In *Proceedings of the 2007 European Conference on Genetic Programming*, pages 78–89. Springer, Valencia, Spain, 2007.

[19] E.A. Heinz. Extended futility pruning. *ICCA Journal*, 21(2):75–83, 1998.

[20] R.M. Hyatt, A.E. Gower, and H.L. Nelson. CRAY BLITZ. *Computers, Chess, and Cognition*, eds. T.A. Marsland and J. Schaeffer, pages 227–237. Springer-Verlag, New York, 1990.

[21] G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 995–1002. IEEE Press, World Trade Center, Seoul, Korea, 2001.

[22] H.L. Nelson. Hash tables in CRAY BLITZ. *ICCA Journal*, 8(1):3–13, 1985.

[23] A. Reinfeld. An improvement to the Scout tree-search algorithm. *ICCA Journal*, 6(4):4–14, 1983.

[24] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[25] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[26] J. Schaeffer, M. Hlynka, and V. Jussila. Temporal difference learning applied to a high-performance game-playing program. In *Proceedings of the 2001*

International Joint Conference on Artificial Intelligence, pages 529–534. Seattle, WA, 2001.

[27] J.J. Scott. A chess-playing program. *Machine Intelligence 4*, eds. B. Meltzer and D. Michie, pages 255–265. Edinburgh University Press, Edinburgh, 1969.

[28] D.J. Slate and L.R. Atkin. CHESS 4.5 - The Northwestern University chess program. *Chess Skill in Man and Machine*, ed. P.W. Frey, pages 82–118. Springer-Verlag, New York, 2nd ed., 1983.

[29] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[30] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277, 1992.

[31] W. Tunstall-Pedoe. Genetic algorithms optimising evaluation functions. *ICCA Journal*, 14(3):119–128, 1991.

[32] M.A. Wiering. *TD learning of game evaluation functions with hierarchical neural architectures.* Master's Thesis, University of Amsterdam, 1995.

APPENDIX

A. EXPERIMENTAL SETUP

Our experimental setup consisted of the following resources:

- CRAFTY 19 chess program running as a native ChessBase engine.
- Encyclopedia of Chess Middlegames (ECM) test suite, consisting of 879 positions.
- FRITZ 9 interface for automatic running of matches, using SHREDDER opening book.
- AMD Athlon 64 3200+ with 1 GB RAM and Windows XP operating system.

B. ELO RATING SYSTEM

The Elo rating system, developed by Arpad Elo, is the official system for calculating the relative skill levels of players in chess. The following statistics from the January 2009 FIDE rating list provide a general impression of the meaning of Elo ratings:

- 21079 players have a rating above 2200 Elo.
- 2886 players have a rating between 2400 and 2499, most of whom have either the title of International Master (IM) or Grandmaster (GM).
- 876 players have a rating between 2500 and 2599, most of whom have the title of GM.
- 188 players have a rating between 2600 and 2699, all of whom have the title of GM.
- 32 players have a rating above 2700.

Only four players have ever had a rating of 2800 or above. A novice player is generally associated with rating values of below 1400 Elo. Given the rating difference (RD) of two players, the following formula calculates the expected winning rate (W, between 0 and 1) of the player:

$$W = \frac{1}{10^{-RD/400} + 1}.$$

Given the winning rate of a player, as is the case in our experiments, the expected rating difference can be derived from the above formula:

$$RD = -400 \log_{10}(\frac{1}{W} - 1).$$