Evolutionary approach to the game of checkers

Magdalena Kusiak, Karol Walędzik, and Jacek Mańdziuk*

Faculty of Mathematics and Information Science, Warsaw University of Technology,
Plac Politechniki 1, 00-661 Warsaw, POLAND

Abstract. A new method of genetic evolution of linear and nonlinear evaluation functions in the game of checkers is presented. Several practical issues concerning application of genetic algorithms for this task are pointed out and discussed. Experimental results confirm that proposed approach leads to efficient evaluation functions comparable to the ones used in some of commercial applications.

1 Introduction

In our previous work [1] a comparison was made between two evolutionary heuristic generators applied to the game of give-away checkers. In order to check the quality of developed heuristics they were tested against themselves and against TD-GAC program [2, 3], which uses temporal difference learning [4].

In this paper the idea of applying evolutionary heuristic generators in two player games domain is evaluated using the game of US checkers. The game is well known and several AI-based approaches to playing checkers has been published, with the most famous ones being Jonathan Schaeffer's Chinook [5] and TDL-Chinook [6]. The paper is particularly motivated by the two "grand Computational Intelligence achievements" - the Samuel's checkers program [7] and Chellapilla and Fogel's *Blondie 24* program [8]. In short the former work empirically proves the efficacy of a combination of self-play and a variant of reinforcement learning as a suitable learning tool for checkers. On the other hand, the other inspiring work - *Blondie 24* - is an apparent example of learning from scratch, without any human guidance, in a knowledge-free regime.

The reminder of the paper is organized as follows: in the next section basic board features used as the components of the evaluation functions are introduced. Section 3 presents different types of both linear and nonlinear heuristical evaluation functions that were generated using the method described in the paper. In Section 4 the evolutionary process is presented in detail together with achieved experimental results. Conclusions and directions for future research are placed in Section 5.

2 Components of the Evaluation Functions

The heuristics described in this paper are relatively simple and made use of some or all of the following parameters, calculated separately for each player:

* Corresponding author. E-mail: mandziuk@mini.pw.edu.pl; fax: (48 22) 625-74-60.

Eight **simple features**: numbers of **(1)** pawns and **(2)** kings; Numbers of safe - i.e. adjacent to the edge of the board - **(3)** pawns and **(4)** kings; Numbers of moveable - i.e. able to perform a move other than capturing - **(5)** pawns and **(6)** kings (these two last parameters were calculated taking no notice of capturing priority); **(7)** Aggregated distance of the pawns to promotion line; **(8)** Number of unoccupied fields on promotion line.

Eleven **layout features**: **(9)** number of defender pieces - i.e. together pawns and kings situated in the two lowermost rows; **(10)** Number of attacking pawns - i.e. positioned in three topmost rows; Numbers of centrally positioned - i.e. situated on the eight central squares of the board - **(11)** pawns and **(12)** kings; Numbers of **(13)** pawns and **(14)** kings positioned on the main diagonal; Numbers of **(15)** pawns and **(16)** kings situated on double diagonal; Numbers of loner **(17)** pawns and **(18)** kings (a loner piece was defined as the one not adjacent to any other piece); **(19)** Number of holes - i.e. empty squares adjacent to at least three pieces of the same color.

Six **pattern features** - described below using common notation presented in Fig. 1. All patterns are described from the white player's point of view. Since at most one instance of each pattern can exist for each player in a given board position, features **(20)**-**(25)** can take only boolean values. **(20)** A Triangle - white pawns on squares 27, 31 and 32; **(21)** An Oreo - white pawns on squares 26, 30 and 31; **(22)** A Bridge - white pawns on squares 30 and 32; **(23)** A Dog - white pawn on square 32 and a black one on square 28; **(24)** A Pawn in the Corner - white man on square 29; **(25)** A King in the Corner - white king on square 4;

Heuristics could also consider sums of or differences in respective parameters **(1)**-**(25)** for both players rather than raw numbers for each player separately.

Two types of heuristics were considered: linear and nonlinear ones. Each linear heuristic consisted of linear combination of the parameters listed above:

$$LinCombOfParam = a_1 \cdot param_1 + a_2 \cdot param_2 + \ldots + a_j \cdot param_j, \quad (1)$$

where $1 \leq j \leq 25$ and $param_1, \ldots, param_j$ were freely chosen from the above 25 parameters (being either raw numbers or sums or differences calculated for both players).

Nonlinear heuristics were composed of a small number (in our tests 3) of IF-conditions which divided the entire game into disjoint stages (c.f. definitions of heuristics 3Ph and E3Ph in Sect. 3). In each stage the respective linear combination of parameters of the form (1) was considered. As it was the case with linear heuristics sums or differences of particular parameters calculated for both players could be used instead of raw numbers. For either type of heuristics coefficients $a_1, \ldots, a_j$ in (1) were optimized in the evolutionary process described in the next section.

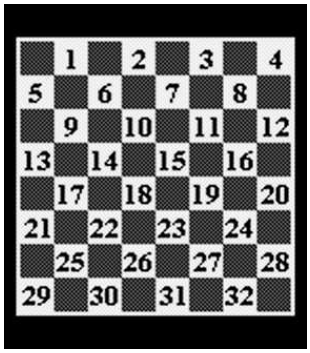


Fig. 1. Notation used for describing patterns. White pawns are at the bottom.

3 Types of Heuristics

Initial, simplified tests were carried out in order to find out whether values of respective parameters for both players in linear heuristics exhibited symmetry (i.e. they evolve to approximately opposite values). Pawns counts and kings counts weights were found to be the most asymmetrical. It was therefore obvious that for the symmetrical parameters differences in respective values should be used in order to decrease the number of genes. As for the two asymmetrical parameters, it was suggested that the asymmetry was a result of dependencies between parameters and therefore using differences instead of individual values might affect the quality of heuristics in a negative way. The following linear and nonlinear heuristics were defined.

Linear Heuristics.

8 Factors (8F). This heuristic took into consideration differences in values of eight first features (**(1)** - **(8)**) listed in sect. 2.

10 Factors (10F). This heuristic took into account the same parameters as 8F, the only exception being the fact that it considered raw numbers of pawns and kings for each player separately (instead of differences), as suggested by the initial symmetry tests described above.

15 Factors (15F). All factors of this heuristic were defined as differences of the respective parameters calculated for both sides. The following features were considered: **(1)**-**(4)**, **(9)**-**(12)**, **(19)**-**(25)**.

19 Factors (19F). All components of this heuristic were in the form of differences between the respective values calculated for both players. It took into account all parameters **(1)**-**(19)** described in Sect. 2. It should be noticed that patterns (Dog, Bridge, Oreo, ...) were not included in this heuristics' definition.

25 Factors (25F). This heuristic took into consideration differences in values of all available parameters described in Sect. 2.

Nonlinear Heuristics.

The general idea of nonlinear heuristics was based on the fact that it was proved to be advantageous to divide the entire game into several stages and

to use different heuristics for different stages. Some crucial moments requiring changing of the evaluation function were identified. These included presence of kings, which clearly indicated that the game was relatively advanced. End-game positions might also require applying different heuristic. Some tests were carried out which showed that introducing nonlinear components with conditions that were not disjoint hindered the genetic algorithm significantly, which resulted from the fact that having several overlapping conditions made it possible to achieve very similar results in many ways, each time with very different values of parameters. Therefore the game was divided into *disjoint stages*.

3Phase (3Ph). In this heuristic the game was divided into three stages: *Beginning*: each player has more than 3 pawns and no kings are present on the board; *Kings*: both players have more than 3 pieces and at least one king is present; *Ending*: one or both players have 3 pieces or less. Linear heuristics in each stage took into consideration the differences in exactly the same 8 parameters as in 8F heuristics.

Expert 3 Phase (E3Ph). In this heuristic the game was divided into stages in the same way as it was the case of 3Ph. Linear heuristics used in each of the three stages took into consideration differences (white vs. black) in the following ten features : **(3)**, **(10)-(12)**, **(16)**, **(20)-(23)** and **(25)**. The above set of parameters was chosen based on the results obtained in initial runs of the generator − choosing parameters which proved to be the most significant.

In both 3Ph and E3Ph parameters concerning kings were, of course, only considered if kings were present on the board.

4 Evolutionary Process and Results

In order to generate coefficients $a_1, \ldots, a_j$ of the above linear combinations genetic algorithms were used. All coefficients of the heuristics were represented as a vector of real numbers whereby each number denoted a single gene (one coefficient). In case of nonlinear heuristics, the conditions that nonlinear components consisted of were not modified by the process of evolution.

Population consisted of 300-400 specimens. In each phase the weakest 75%-80% of the population were regenerated.

Selection − selection was done by means of tournaments. For each tournament a number of specimens were randomly chosen from the population. Their fitness assessments were compared in order to determine the winner of the tournament. Winners of two such tournaments were coupled and went on to crossbreed. Depending on the number of genes of each specimen the size of tournament between specimens was set between two and five.

Crossover − each pair of respective linear combinations in a heuristic (i.e. each nonlinear component and base heuristic function) was crossed over independently. The genotype of each linear combination was randomly partitioned into two. Descendant inherited values of each part of the genotype from respective parent. The value of the gene on which the division was placed was randomly chosen from the interval defined by the values of this gene in parent specimens.

The weakest specimen in the population was to be replaced by the newly created descendant, however only if the new specimen's fitness evaluation was greater than that of the one to be replaced.

The ratio of *effective* crossovers (i.e. the ones in which a resulting specimen was actually added to the population) to *potential* crossovers turned out to remain fairly stable throughout the process and ranged between $80\% - 90\%$.

Mutation. Three kinds of mutation could occur in each of the genes of every newly created specimen: multiplying a value of a gene by two, dividing it by two or changing its sign. Multiplying or dividing a value by two were twice as probable as changing the sign.

Heuristic Generator (HG). One of the difficulties encountered while designing a genetic algorithm was defining fitness function for the heuristics. In order to solve this problem the general idea presented in [9] was followed. The game was partitioned into several disjoint stages according to the number of moves already performed. During the first phase of the algorithm a heuristic that would be able to assess correctly situations close to the end of the game was to be obtained. In order to achieve this, alpha-beta algorithm with no heuristic was used to assess a number of randomly generated positions close to the leaves of the game tree. All positions beyond the depth of alpha-beta algorithm were considered a draw. Subsequently, each specimen assessed the same positions and its fitness was calculated according to the formulae $n/\sum (h_i - a_i)^2$, where n denotes the number of test situations, h_i- assessment of the i-th test situation by the heuristic specimen and a_i- by the alpha-beta algorithm.

Once the initial stage had ended, worst fitted fraction of the population was replaced by new random specimens. New board situations closer to the root of the game tree were generated and they were assessed by the alpha-beta algorithm with the fittest specimen of the previous phase used as its heuristic function. The process continued until the root of the game tree was reached. In other words, the evaluation function of HG was evolved step-by-step starting from positions achieved far from the initial position and moving backwards. In each phase a constant fraction of all the test boards came from the stage of the game closest to the beginning (i.e. the stage considered most recently).

In the first step positions obtained in between 82 and 86 moves were considered. This interval was determined based on preliminary tests calculating the average number of moves necessary to finish a game performing random moves. The difference in depths between subsequent phases was set to 6. The number of positions considered in each phase ranged from $3,000$ to $4,500$.

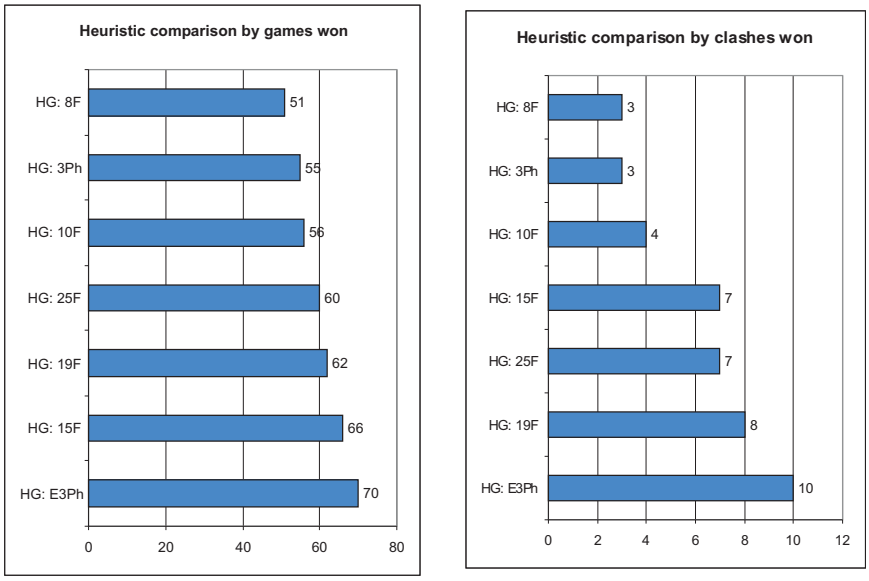
Heuristics Comparison.

In order to determine the relative quality of heuristics a tournament was held, where each heuristic played 10 games with each other with sides swap after each game. Due to some randomness in searching the game tree implemented in alpha-beta, games played between any two heuristics were pairwise different.

Two classifications were used. The first one was based on the total number of games won, tied and lost by each heuristic. The other one took into consideration results of whole 10-game clashes between heuristics. In each case, heuristic would

gain 2 points for a win and 1 point for a draw. All the games were played with alpha-beta's search depth limit of 6. The results are presented in Fig. 2.

As might be expected the E3Ph heuristic proved to perform better than any other one, no matter which comparison criteria were considered. More advanced heuristics, considering more sophisticated parameters, also tended to perform well in the tournament, which again was in line with the expectations.



(a) Comparison by Games (b) Comparison by Clashes

Fig. 2. Heuristic comparison.

Comparison with public domain and commercial programs.

The piece of research described in this paper was not intended to create a program that could compete against professional or commercial checkers programs. Its main purpose was to assess credibility of evolutionary approach to heuristic generation, in particular with HG method. It was developed with flexibility rather than speed in mind and as a result assessed between $55,000$ and $100,000$ nodes per second whereas professional checkers applications were sometimes more than one order of magnitude faster. Nevertheless, in order to assess the quality of evolved heuristics, some comparison games were played. In order to make them fair, most of the features of the commercial programs (opening books, endgame databases, hashtables, killer moves identification etc.) had to be disabled or reduced as much as possible. Four best heuristics were chosen for those tests: E3Ph, 15F, 19F and 25F.

The first checkers engine that was tested against the generated heuristics was Simple Checkers (*http://www.fierz.ch/checkers.htm*). Although it is a relatively simple program with publicly available source code, it is at the same time one of the best engines among those not making use of opening books and end-game databases and it surpasses many commercial applications (c.f. *http://www.bobnewell.net/checkers/checkerprograms.html*). In order to make the comparison fair, Simple Checkers (SC) algorithm was slightly modified so that it would always perform search exactly to the depth of 6 plies, i.e. the same depth as our alpha-beta algorithm. Each heuristic played 50 games against SC. Generally the program appeared to be too strong opponent for developed heuristics. It turned out that the only heuristic capable of winning at least one of 50 games played was E3Ph, achieving a final score of 37 points (1 win, 35 ties, 14 loses). All the other heuristics were capable to draw a few games each: 15F accomplished 9 draws, 25F - 6 draws and 19F - 2 draws.

In the next tests E3Ph heuristic (the undoubtedly best of all heuristics generated) played against shareware version of commercial application Actual Checkers 2000A (*http://www.atlantsoft.com/ach2000a/download.htm*). Due to shareware edition limitation, our heuristic would move first during all of the games (during all other comparisons, sides were swapped after each game). Actual Checkers (AC) program was set to the highest possible difficulty level and was expected to perform a move in 3 seconds on average (which for some reason it wouldn't, taking usually around 5 seconds for each move), which resulted in searches of depth 12 to 18. The number of nodes analyzed by our alpha-beta algorithm was usually well below 1 million per move, which was equivalent to search depth of approximately 9 plies during most of the mid-game but significantly less during end-game. Despite lower search depth E3Ph managed to draw 3 of 4 games played. Single games played with search depths of 9 and 10 ended in draws as well. During the tests it was observed that E3Ph would perform noticeably worse during end-games, often failing to take full advantage of its upper-hand or defend a draw well enough. This probably stemmed from the fact that AC's hashtables proved especially useful in this phase of the game.

Finally, in order to once again confirm quality of the E3Ph heuristic, games against another shareware edition of commercial checkers program Mad Checkers (*http://www.sapphiregames.com/madcheckers*) were played. This time alpha-beta's depth limit was set to 8 as usual, but Mad Checkers (MC) engine, being less acclaimed application, was given 0.5s for its move, which was not significantly less than it usually took our program to assess mid-game situations. In order to make the competition even easier for MC, alpha-beta was switched to depth limit of 7 (or in some cases even 6) whenever it happened to perform too slowly during endgames. Nevertheless, our heuristic managed to win 3 out of 4 games with one being a draw because of position repetitions, although MC was left with 3 pieces against our heuristic's 5. In the second 4-game tournament against MC our alpha-beta algorithm was switched to depth limit of 6 plies whilst MC was still admitted 0.5s for a move (i.e. on average more than

two times longer than our program would require for a move). This time our heuristic managed to win 1 game and draw 3 others.

5 Conclusions

The focus of this paper is to test usefulness of pure genetic approach for generating evaluation functions in the game of checkers. The underlying assumption is simplicity and generality of proposed solutions which results in using straightforward genetic operators and shallow game tree search with plain alpha-beta algorithm. In particular, no alpha-beta search enhancements (transposition tables, history heuristics, killer moves, etc.) and no opening books or end-game databases are used.

Not surprisingly, the best of all tested heuristics is E3Ph − a nonlinear 3-phase heuristic. This result confirms the common knowledge in game research that *it is advantageous to divide the entire game into phases* and develop separate heuristics for each part of the game. Specifically it was also observed that game phases need to partition board positions space into *disjoint sets*. Otherwise the genetic process may have difficulties in assigning coefficients for the features shared by two or more game phases.

Another general guideline is *to use differences of respective parameters calculated for both sides* rather than raw numbers separately for both players. This observation was fully confirmed, with the only exception being the case of relatively simple heuristics, where it is recommended that the numbers of pawns and kings be input as raw values (c.f. comparison between 8F and 10F heuristics).

Taking into account the above mentioned assumptions concerning simplicity of proposed approach, the achieved outcomes suggest that usage of genetic algorithms may be a credible way of producing heuristic functions for two-player games. Although heuristics described in this paper did not surpass or even reach the level of play of those created during years of checkers programs development, they are still a challenge for at least some of the commercially available software. This indicates that evolutionary approach, tuned by a careful choice of settings and meta-rules can be successfully applied - particularly to games, for which not enough expert knowledge exists.

References

1. Kusiak, M., Walędzik, K., Mańdziuk, J.: Evolution of heuristics for give-away checkers. In: Proc. ICANN 2005, Part 2, Warszawa, Poland. Volume 3697 of LNCS., Springer-Verlag (2005) 981–987
2. Mańdziuk, J., Osman, D.: Temporal difference approach to playing give-away checkers. In: Proc. ICAISC 2004, Zakopane, Poland. Volume 3070 of LNAI., Springer (2004) 909–914
3. Osman, D., Mańdziuk, J.: Comparison of TDLeaf(λ) and TD(λ) learning in game playing domain. In: Proc. ICONIP 2004, Calcutta, India. Volume 3316 of LNCS., Springer-Verlag (2004) 549–554

4. Sutton, R.: Learning to predict by the method of temporal differences. Machine Learning **3** (1988) 9–44
5. Schaeffer, J.: One Jump Ahead: Challenging Human Supremacy in Checkers. Springer-Verlag (1997)
6. Schaeffer, J., Hlynka, M., Jussila, V.: Temporal difference learning applied to a high-performance game-playing program. In: Proc. IJCAI 2001. (2001) 529–534
7. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229
8. Fogel, D.B.: Blondie24: Playing at the Edge of Artificial Intelligence. Morgan Kaufmann (2001)
9. Borkowski, M.: Analysis of algorithms for two-player games. M.Sc. Thesis, Warsaw University of Technology (in Polish) (2000)