Evolution of heuristics for give-away checkers

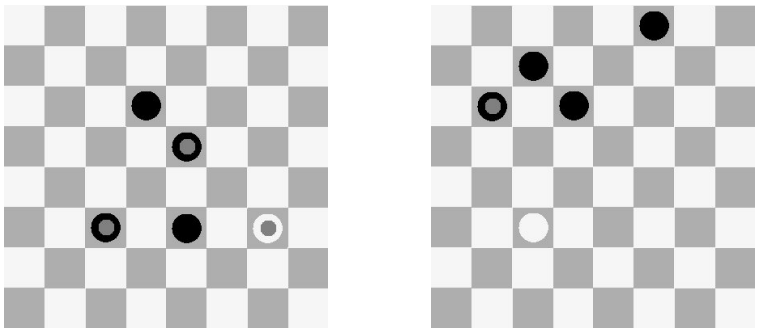
Magdalena Kusiak, Karol Walędzik and Jacek Mańdziuk

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

Abstract. The efficacy of two evolutionary approaches to the problem of generation of heuristical linear and non-linear evaluation functions in the game of give-away checkers is tested in the paper. Experimental results show that both tested methods lead to heuristics of reasonable quality and evolutionary algorithms can be successfully applied to heuristic generation in case not enough expert knowledge is available.

1 The game of Give-Away Checkers (GAC)

The game of US give-away checkers [1] is played according to the same rules as US checkers. The way of determining the winner is the only exception. In order to win in the game of GAC a player has to lose all his/her pieces or be unable to perform a move. The rules of the game are simple and widely known and - at the same time - the results of brute-force algorithm using trivial strategy of losing pieces as quickly as possible are unsatisfactory. One of the reasons for unsuitability of this simple algorithm is the fact that a single piece is barely mobile and has very restricted choice of possible moves. Fig. 1 presents two situations in which white loses despite having only one piece left.



(a) Black to play and win. (b) White to play and lose.

Fig. 1. Examples of inefficiency of greedy heuristics. White move bottom-up. Open circles denote kings.

2 The evaluation function

In each of heuristical evaluation functions discussed in this paper some or all of the following components (factors) were considered (each of them calculated either separately for each player or as a difference of respective values for both players): numbers of (1) pawns (i.e. pieces other than kings), (2) kings and (3) pieces; numbers of (4) safe (i.e. adjacent to the edge of the board) pawns, (5) safe kings and (6) safe pieces; numbers of (7) moveable (i.e. able to make a move other than capturing − this feature was calculated without considering capturing priority) pawns, (8) moveable kings and (9) moveable pieces; (10) aggregated distance of all pawns to promotion line; (11) number of unoccupied fields on the promotion line. Each heuristic consisted of linear combination of some (or none) of the above parameters and arbitrary number of nonlinear components, each of them of the following form:

IF [**NOT**](*param1* **BETWEEN** minVal1 **AND** maxVal1 **AND/OR** *param2* **BETWEEN** minVal2 **AND** maxVal2 **AND/OR** ...) **THEN** $Heuristic\_Result$ $+=LinearCombinationOfParameters$

Only one logical operator (AND or OR) could be used in each nonlinear component and negation could be applied to the whole condition only.

3 Evolutionary algorithms

Genetic algorithms were used to generate weights in each of the above *LinearCombinationOfParameters*. All variable parameters of the heuristics were represented as a vector of real numbers and each number was a single gene. Conditions defined in nonlinear components were not modified by the evolutionary process. Two different approaches, described in the following subsections were implemented.

3.1 Heuristic Generator (HG)

One of the problems encountered while designing a genetic algorithm was defining fitness function for the heuristics. The general idea to solve this problem was based on [2]. The game was divided into several stages and the purpose of the first phase of the algorithm was to obtain a heuristic that would be able to assess correctly situations close to the end of the game. In order to achieve this, a number of situations close to the leaves of the game tree were generated and assessed using alpha-beta algorithm without heuristic. If alpha-beta failed to reach a leaf of the tree, the situation was considered to be a draw. Subsequently, each specimen assessed the same situations and its fitness was defined as $[n/\sum(h_i - a_i)^2]$, where n is the number of test positions, h_i - assessment of the i-th test situation by the heuristic specimen and a_i by the alpha-beta algorithm.

Depending on the algorithm settings the fitness of each specimen could be divided by the sum of similarities of all specimens from the population to it. Similarity of two specimens was defined as $exp(-d^2)$, where d is Euclidean distance between their genotypes. This was done as a mean to encourage speciation [3, 4], which in turn might lead to improved exploration of the problem space. Since GAC has only three possible results and the values of heuristics belong to a continuous interval, the depth of a leaf in the game tree was taken into consideration when assessing it.

Once the initial stage had ended, some new situations were generated, that were closer to the root of the game tree and worst fitted fraction of the population was replaced by random specimens. The fittest specimen of the previous phase was used by alpha-beta to assess these new situations. The process continued until the beginning of the game was reached.

In each phase a constant fraction of all test boards came from the stage of the game nearest to the beginning. Depending on the settings of the algorithm, the situations closer to the leaves of the game tree were either regenerated and reassessed by the newest heuristic or once generated they were used throughout all subsequent phases. The following genetic operators were used:

Selection. Tournament selection was implemented. Several specimens were randomly chosen from the population. The fittest among them was the winner of the tournament. In order to determine a pair of specimens to crossbreed, two such tournaments were held, and their winners were coupled.

Crossover. Each pair of respective linear combinations contained by a heuristic was crossed over independently. The genotype of each linear combination was randomly divided into two parts and values of each part were inherited from one parent. The value of the gene on which the division was placed was taken from the interval defined by the values of this gene in parent specimens. The descendant replaced the weakest specimen in the population.

Mutation. Three kinds of mutations occurred in population: 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. Each gene of a specimen mutated independently[1].

3.2 Simple Heuristic Generator (SHG)

The idea of the algorithm was based on a simplistic assumption that results of games played by pairs of specimens define a relation close to partial order. In order to determine the result, two specimens played one or two (with sides swap) games against each other. By default the search depth during the games was set to 3. Basing on the relation described above, it was relatively easy to compare and sort specimens within a small set. Therefore, no fitness function was necessary to carry out tournament selection.

[1] Instead of multiplication/division of gene's value also addition/subtraction within some range was tested, but results were poorer in that case.

The genetic operators used in this algorithm bear great resemblance to those described above. The only significant difference is the necessity to normalize specimens genotypes. Two specimens to crossbreed were chosen by means of tournaments. Additional tournament was held to determine the weakest specimen to be replaced by the descendant.

4 Results

4.1 Heuristic Types

Based on preliminary tests we have decided to inspect four types of heuristics in more detail. Two of them (8F and 10F) were linear and two other (3Ph and 5Ph) consisted only of nonlinear components. Each heuristic was generated twice: once using HG and once with SHG.

8Factors (8F) heuristic was a linear combination of the differences (between the player and his opponent) in the following parameters described in Sect. 2: (1), (2), (4), (5), (7), (8), (10) and (11). For example (2) in the above denotes the following feature: *the number of kings owned by the player minus the number of opponents kings.*

10Factors (10F) heuristic was a linear combination of the differences in the following six parameters described in Sect. 2: (4), (5), (7), (8), (10), (11) and of the four raw values, namely (1) and (2), each of them calculated for both playing sides.

Basing on the analysis of games played it was decided that it would be advantageous to divide the entire game into several disjoint stages and to use different heuristic for each stage. Two crucial moments requiring change of the heuristic were identified. Firstly, presence of kings certainly indicates that the game has entered an advanced stage. Moreover, due to the mobility issues mentioned earlier, end-game positions might also require defining a new heuristic.

3Phase (3Ph) heuristic assigned each situation on the board to one of three disjoint categories: (a) **ending**: one of the players has at most three pieces left; (b) **kings**: both opponents have more than 3 pieces and at least one player has some kings; (c) **beginning**: both opponents have more than 3 pieces and no kings exist. A linear heuristic respective to 8F was assigned to evaluate situations belonging to each category. For example phase (c) was encoded in the following pseudo-algorithm:

IF ABS(Players_pieces_count - 10.0) < 6.5 AND ABS(Opponent's_pieces_count - 10.0) < 6.5 AND ABS(Total_kings_count) < 0.5 THEN $C_1 * Diff(1) + C_2 * Diff(4) + C_3 * Diff(7) + C_4 * Diff(10) + C_5 * Diff(11)$, where $C_1, \ldots, C_5$ are evolvable coefficients, and $Diff(n)$ denotes the value equal to the difference of feature n (listed in sect. 2) between player and its opponent.

5Phase (5Ph) heuristic was similar to 3Ph with the only exception being that **kings** category (i.e. (b) in the above) was subdivided into three categories

depending on which of the players was in possession of kings. Again, a linear heuristic respective to 8F was assigned to evaluate situations belonging to each of 5 categories.

4.2 Algorithm Settings

In case of HG the depth of the initial situations in the game tree was between 81 and 87. The interval was determined basing on preliminary tests calculating average number of moves necessary to finish a game of GAC performing random moves. The difference in depths between subsequent phases was set to 6 since alpha-beta search with depth limit of 6 was still reasonably fast.

For linear heuristics generation the number of test boards for each phase was 3 000 and reusing test boards was disabled whereas for nonlinear ones the count of the boards was 6 000 and they were reused in different phases. While fewer boards were assessed in each phase during the generation of linear heuristics the total number of situations was greater which resulted in better exploration of the problem space. On the other hand, evaluating as many as 6 000 boards during each phase while generating nonlinear heuristics minimized the chance of considering too few situations belonging to certain categories and propagating the error upwards.

Test populations consisted of 350 specimens. In each phase the weakest 80% of the population were regenerated.

For SHG each test population consisted of $100 - 150$ specimens. Populations had to be smaller because of the way specimens were compared with each other. During all tests alpha-beta search limit in the games played for comparison purposes was set to the depth of 3 and maximum of 150 expanded nodes. Comparisons were symmetrical, i.e. two specimens played two games against each other swapping sides after the first game.

During all tests a newly created specimen replaced the weakest specimen in the population (in SHG the specimen to be replaced was chosen by means of a tournament). However, this only happened if the descendant was fitter than the specimen to be replaced. The potential crossovers to effective crossovers (i.e. the ones in which created specimen was actually added to the new population) ratio was investigated. It turned out that the fraction remained fairly stable throughout the process. About $80\% - 90\%$ of all the crossovers were effective in HG and about 70% in SHG. The stability resulted from the fact that in the initial stages of the algorithm convergence was comparatively quick and therefore descendants tended to be fitter than specimens from previous generations. On the other hand, in the final stages vast majority of the specimens were almost identical and there were virtually no difference in fitness between ancestors and descendants in which case new specimens were preferred and added to the population.

The convergence of the evolution is clearly illustrated by changes in lengths of intervals for different parameters as well as by distinct declines in their variances. For most parameters variances dropped by more than a thousand times in the course of evolution.

It appeared that mutations had no significant influence on the results of evolution. In SHG best specimens were saved every 1 000 crossbreedings and in most runs not a single mutated specimen was logged. In the process of evolving linear heuristics using HG about 1% of the fittest specimens saved turned out to have experienced mutation. The fraction was about 10% for nonlinear heuristics. The significant difference may result from the fact that in initial stages of the algorithm some genes were not applicable to the situations evaluated and therefore their mutation had no influence on the overall fitness of the specimen.

4.3 Heuristics' Performance

In order to find out about the quality of different heuristics generated, tests were run during which each heuristic played 40 games (swapping sides after each game) against TD-GAC program [5–7], which uses temporal difference algorithm and learns from games played. During the tests alpha-beta search depth was set to 6 in evolved heuristics and was set to 4 in TD-GAC (since TD-GAC heuristic makes use of more sophisticated parameters, including indirect exploration of the game tree one ply further). In order to make a fair comparison of heuristics the learning ability of TD-GAC was temporarily disabled. Please note, that due to some randomness in searching the game tree implemented in alpha-beta, for any particular heuristic games played with TD-GAC were not identical.

The results of comparison presented in Fig. 2 show clearly that the heuristics tended to perform well, taking into account simple parameters they considered. As it can be seen in Fig. 2 nonlinear heuristics (particularly 3Ph) generally performed slightly better than linear ones which could be expected. Worse performance of 5Ph heuristic might stem from its greater complexity which could have hindered evolutionary process.

Additional tests were carried out to measure performance of the alpha-beta algorithm. It turned out that the results depended to a great extent on the players strategies. During quick games with a lot of compulsory capture sequences lower average branching factors were reported and fewer nodes had to be analyzed as well. During games with the search depth of 6 linear heuristics needed approximately $54-78$ms to assess a situation. The number of nodes analyzed was between 3 700 and 6 700 (at about $75\,000 - 85\,000$ nodes per second). For nonlinear heuristics assessment lasted on average $120 - 280$ms. Heuristics analyzed $6\,000 - 10\,000$ nodes with the speed of about 40 000 nodes per second.

Estimations were also made as to pruning efficiency of different heuristics, which was defined as $(1 - n/s)$, where n denotes the number of nodes analyzed and s-theoretical size of the game subtree calculated basing on branching factor reported and search depth. Approximated pruning efficiency turned out to be rather stable for all heuristics ranging from 0.75 to 0.9.

5 Conclusions and directions for future research

The main research goal of this paper concerns the possibility of building efficient heuristic evaluation functions based on evolutionary approach. In particular the

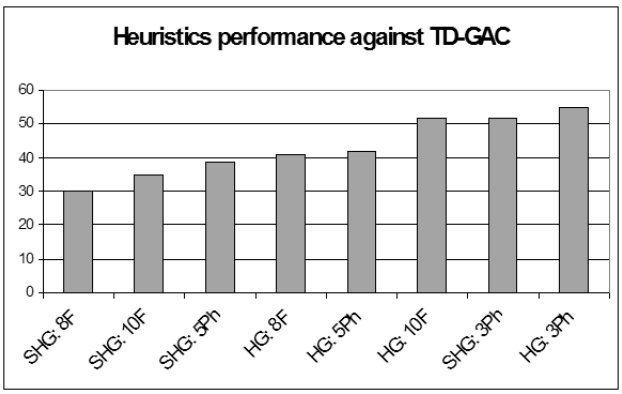


Fig. 2. Performance of the heuristics against TD-GAC.

efficacy of nonlinear vs. linear heuristics is verified along with comparison of HG and SHG.

Results of games played against TD-GAC support the hypothesis that HG generally outperforms SHG, and hence it can be concluded that due to its non-transitivity a direct assessing method of SHG may not be the appropriate evaluation method.

As it can be expected non-linear heuristics dominated over linear ones. However, based on some other tests (not presented) it is strongly recommended that nonlinear components be defined over **disjoint conditions**. Otherwise, having several overlapping conditions makes it possible to achieve very similar results in many ways, each time with very different values of parameters.

In future we plan to verify other schemes of evolving non-linear evaluation functions in GAC as well as apply these methods to other board games.

References

1. Alemanni, J.B. http://perso.wanadoo.fr/alemanni/ give_away.html (1993)
2. Borkowski, M.: Analysis of algorithms for two-player games. M.Sc. Thesis, Warsaw University of Technology (in Polish) (2000)
3. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimisation. In: Proceedings of the 2nd International Conference on Genetic Algorithms. (1993) 41–49
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Pub. Co. (1989)
5. Osman, D., Mańdziuk, J. http://gac-arena.gt.pl/ (2004)
6. Mańdziuk, J., Osman, D.: Temporal difference approach to playing give-away checkers. In Rutkowski, L., et al., eds.: 7th Int. Conf. on Art. Intell. and Soft Comp. (ICAISC 2004), Zakopane, Poland. Volume 3070 of LNAI., Springer (2004) 909–914
7. Osman, D., Mańdziuk, J.: Comparison of tdleaf(λ) and td(λ) learning in game playing domain. In Pal, N.R., et al., eds.: 11th Int. Conf. on Neural Inf. Proc. (ICONIP 2004), Calcutta, India. Volume 3316 of LNCS., Springer (2004) 549–554