Winning Ant Wars: Evolving a Human-Competitive Game Strategy Using Fitnessless Selection

Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch

Poznan University of Technology, Poznań, Poland
Institute of Computing Science

Abstract. We tell the story of *BrilliAnt*, the winner of the Ant Wars contest organized within GECCO'2007, Genetic and Evolutionary Computation Conference. The task for the Ant Wars contestants was to evolve a controller for a virtual ant that collects food in a square toroidal grid environment in the presence of a competing ant. BrilliAnt, submitted to the contest by our team, has been evolved through competitive one-population coevolution using genetic programming and a novel *fitnessless selection* method. In the paper, we detail the evolutionary setup that lead to BrilliAnt's emergence, assess its human-competitiveness, and describe selected behavioral patterns observed in its strategy.

1 Introduction

Ant Wars was one of the competitions organized within GECCO'2007, Genetic and Evolutionary Computation Conference, in London, England, July 7–12, 2007. The goal was to evolve a controller for a virtual ant that collects food in a square toroidal grid environment in the presence of a competing ant. In a sense, this game is an extension of the so-called Santa-Fe trail task, a popular genetic programming benchmark, to two-player environment.

Ant Wars may be classified as a probabilistic, two-person board game of imperfect information. Each game is played on a 11x11 toroidal board. Before the game starts, 15 pieces of food are randomly distributed over the board and two players (ants) are placed at predetermined board locations. The starting coordinates of ant 1 and ant 2 are $(5, 2)$ and $(5, 8)$, respectively. No piece of food can be located in the starting cells. An ant has a limited field of view – a square neighborhood of size 5x5 centered at its current location, and receives complete information about the states (empty, food, enemy) of all cells within it.

The game lasts for 35 turns per player. In each turn ant moves into one of 8 neighboring cells. Ant 1 moves first. If an ant moves into a cell with food, it scores 1 point and the cell is emptied. If it moves into a cell occupied by the opponent, it kills it: no points are scored, but only the survivor can go on collecting food until the end of the game. Moving into an empty cell has no extra effect. A game is won by the ant that attains higher score. In case of tie, Ant 1 is the winner.

As the game outcome strongly depends on food distribution, the games may be grouped into *matches* played on different boards. Each match consists of $2 \times k$

M. O'Neill et al. (Eds.): EuroGP 2008, LNCS 4971, pp. 13–24, 2008.

games played on k random boards generated independently for each match. To provide for fair play, the contestants play two games on the same board, in the first game taking roles of Ant 1 and Ant 2, and then exchanging these roles; we refer to such a pair of games a *double-game*. To win the match, an ant has to win $k+1$ or more games within the match. In the case of tie, the total score determines the match outcome. If there is still a tie, a randomly selected contestant wins.

The Ant War contestants were required to produce an ANSI-C function *Move*(*grid*, *row*, *column*), where *grid* is a two-dimensional array representing board state, and (*row*, *column*) represents ant's position. The function was supposed to indicate ant's next move by returning direction encoded as an integer from interval $[0, 7]$. Function code was limited to 5kB in length.

In this paper, we tell the story of Ant Wars winner, *BrilliAnt*, an ant submitted by our team. BrilliAnt has been evolved through competitive one-population coevolution using genetic programming (GP) and a novel *fitnessless selection* method. Despite being conceptually simpler than fitness-based selection, fitnessless selection produces excellent players without externally provided yardstick, like a human-made strategy. An extensive computational experiment detailed in the paper proves that BrilliAnt and other artificial ants evolved using this approach are highly human-competitive in both direct terms (playing against a human opponent) and indirect terms (playing against a human-devised strategy).

In the following Section 2 we shortly summarize the past game-related research in GP. Section 3 describes the model of board perception and the repertoire of GP functions used for strategy encoding. Section 4 provides details on experimental setup and defines the fitnessless selection method. In Section 5, we assess human-competitiveness of the evolved ants, and in Section 6 we describe the most interesting behavioral patterns observed in BrilliAnt's strategy.

2 Genetic Programming for Evolving Game Players

Achieving human-competitive performance in game playing has been AI's holy grail since its very beginning, when game playing strategies, like the famous Bernstein's chess and Samuel's checker players, were hand-crafted by humans. The most spectacular achievement of AI in the game domain was the grand master Garry Kasparov's defeat in duel with Deep Blue, which implemented a brute force approach supported by human expertise. Through successful, it is dubious whether this kind of approach can be applied to more complicated games, and how much does it help to understand and replicate the human intelligence. The $1.5M Ing Prize for the first computer player to beat a nominated human competitor in the Go game is still untouched, presumably because Go has too many states to be approached by brute force. Hard AI is also often helpless when it comes to real-time (strategy) games [3] or multi-agent games where the number of possible states can be even greater than in Go. Things get more complicated also for hand-designed algorithms when the game state is only partially-observable or the game is probabilistic by nature.

The partial failure of hard AI in devising truly intelligent approach to games clearly indicates that handcrafting a good game-playing strategy for a nontrivial game is a serious challenge. The hope for progress in the field are the methods that automatically construct game playing programs, like genetic programming (GP, [7]) used in our approach.

Koza was the first who used GP to evolve game strategies [6] for a two-person, competitive, simple discreet game. Since then, other researchers have demonstrated that the symbolic nature of GP is suitable for this kind of task. Studies on the topic included both trivial games such as Tic Tac Toe [1] or Spoof [16], as well as more complicated and computationally-demanding games, like poker [14]. Core Wars, a game in which two or more programs compete for the control of the virtual computer, is among the popular benchmark problems for evolutionary computations and one of the best evolved players was created using a μGP [4]. Luke's work [8] on evolving soccer softball team for RoboCup97 competition belongs to the most ambitious applications of GP to game playing, involving complicated environment and teamwork. Recently, Sipper and his coworkers demonstrated [13] human-competitive GP-based solutions in three areas: backgammon [2], RoboCode [12] (tank-fight simulator) and chess endgames [5].

3 Ant's Architecture

In the game of Ant Wars introduced in Section 1, ant's field of view (FOV) contains 25 cells and occupies 20.7% of the board area. The expected number of visible food pieces is 3.02 when the game begins. The probability of having n food pieces within FOV drops quickly as n increases and, for instance, for $n = 8$ amounts to less than 0.5%. This, together with FOV's rotational invariance and symmetry, indicates that the number of unique and realistically possible FOV states is low, and any strategy based on the current (observed) FOV state only cannot be competitive in a long run. More may be gained by virtually extending the FOV, i.e., keeping track of past board states as the ant moves. To enable this, we equip our ants with memory, implemented as three arrays overlaid over the board:

- *Food memory* F, that keeps track of food locations observed in the past,
- *Belief table* B, that describes ant's belief in the current board state,
- *Track table* V, that marks the cells visited by ant.

At each move, we copy food locations from ant's FOV into F. Within FOV, old states of F are overridden by the new ones, while F cells outside the current FOV remain intact. As board states may change subject to opponent's actions and make the memory state obsolete, we simulate memory decay in the belief table B. Initially, the belief for all cells is set to 0. Belief for the cells within FOV is always 1, while outside FOV it decreases exponentially, by 10% with each move. Table V stores ant's 'pheromone track', initially filled with zeros. When ant visits a cell, the corresponding element of V is set to 1.

To evolve our ants, we use tree-based, strongly typed genetic programming. A GP tree is expected to evaluate the utility of the move in a particular direction: the more attractive the move, the greater tree's output. To benefit from rotational invariance, we use one tree to evaluate multiple orientations. However, as ants are allowed to move horizontally, vertically, and diagonally, we evolve two trees in each individual to handle these cases: a *'straight'* tree for handling main directions (N, E, S, W) and a '*diagonal*' tree to handle the diagonal directions (NE, NW, SE, SW)[1]. We present the FOV state to the trees by appropriately rotating the coordinate system by a multiple of 90 degrees; this affects both FOV and the ant's memory. The orientation that maximizes trees' output determines the ant's move; ties are resolved by preferring the earlier maximum.

Our ants use three data types: *float* (F), *boolean* (B), and *area* (A). An area represents a rectangle stored as a quadruple of numbers: midpoint coordinates (relative to ant's current position, modulo board dimensions) and dimensions. In theory, the number of possible values for area type is high, so it would be hard for evolution to find the most useful of them. That it why we allow only for relatively small areas, such that their sum of dimensions does not exceed 6. For instance, the area of dimensions (2, 5) cannot occur in our setup.

The set of GP terminals includes the following operators:

- Const(): Ephemeral random constant (ERC) for type F ($[-1; 1]$),
- ConstInt(): Integer-valued ERC for type F (0..5),
- Rect(): ERC for type A,
- TimeLeft() – the number of moves remaining to the end of the game,
- Points() – the number of food pieces collected so far by the ant,
- PointsLeft() – returns 15−Points().

Functions implementing non-terminal nodes (operators):

- IsFood(A) – returns *true* if the area A contains at least one piece of food,
- IsEnemy(A) – returns *true* if the area A contains the opponent,
- Logic operators: And(B, B), Or(B, B), Not(B),
- Arithmetic comparators: IsSmaller(F, F), IsEqual(F, F),
- Scalar arithmetics: Add(F, F), Sub(F, F), Mul(F, F),
- If(B, F, F) – evaluates and returns second child if first child returns true, otherwise evaluates and returns its third child,
- NFood(A) – the number of food pieces in the area A,
- NEmpty(A) – the number of empty cells in the area A,
- NVisited(A) – the number of cells already visited in the area A,
- FoodHope(A) – returns the estimated number of food pieces that may be reached by the ant within two moves (assuming the first move is made straight ahead, and the next one in arbitrary direction).

[1] We considered using a single tree and mapping diagonal boards into straight ones; however, this leads to significant topological distortions which could possibly significantly deteriorate ant's perception.

Note that functions that take the argument of area type compute their return value basing not only on FOV, but on the food memory table F and the belief table B. For example, NFood(a) returns the scalar product, constrained to area a, of table F (food pieces) and table B (belief).

One should also emphasize that all GP functions mentioned here are straightforward. Even the most complex of them boil down to counting matrix elements in designated rectangular areas. Using more sophisticated functions would be conflicting with contests rules that promoted solutions where the intelligence was evolved rather than designed.

4 How BrilliAnt Evolved

In our evolutionary runs ants undergo competitive evaluation, i.e., face each other rather than an external selection pressure. This is often called one-population coevolution [10] or competitive fitness environment [1,8]. In such environments, the fitness of an individual depends on the results of games played with other individuals from the same population. The most obvious variant of this approach is the *round-robin tournament* that boils down to playing one game between each pair of individuals. The fitness of an individual is defined as the numbers of games won. Since the round-robin tournament needs $n(n-1)/2$ games to be played in each generation for population of size n, some less computationally demanding methods were introduced.

Angeline and Pollack [1] proposed *single-elimination tournament* that requires only $n-1$ games to be played. In each round the players/individuals are paired, play a game, and the winners pass to the next round. At the end, when the last round produces the final winner of the tournament, fitness of each individual is the number of won games. Another method reported in literature, *k-random opponents*, defines individual's fitness as the average result of games with k opponents drawn at random from the current population. The method requires kn games to be played. The special case of this method for $k = 1$ is also known as *random pairing*. An experimental comparison between k-random opponents and single-elimination tournament may be found in [11].

Here we propose a novel selection method called *fitnessless selection.* It does not involve explicit fitness measure and thus renders the evaluation phase of evolutionary algorithm redundant. Fitnessless selection resembles tournament selection, as it also selects the best one from a small set of individuals drawn at random from the population. In the case of tournament selection the best individual is the one with the highest fitness. Since our individuals do not have explicit fitness, in order to select the best, we apply a single-elimination tournament, in which the winner of the last (final) round becomes immediately the result of selection. This feature, called *implicit fitness*, makes our approach significantly different from most of contributions presented in literature. The only related contribution known to us is [15].

Using ECJ [9] as the evolutionary engine, we carried out a series of preliminary experiments with various evolutionary setups, including island model and

different variants of selection procedure. In a typical experiment, we evolved a population of 2000 individuals for 1500 generations, which took approx. 48 hours on a Core Duo 2.0 GHz PC (with two evaluating threads). In all experiments, we used probabilities of crossover, mutation, and ERC mutation, equal to 0.8, 0.1, and 0.1, respectively. GP trees were initialized using ramped half-and-half method, and were not allowed to exceed depth 8. For the remaining parameters, we used ECJ's defaults [9].

We relied on the default implementation of mutation and crossover available in ECJ, while providing specialized ERC mutation operators for particular ERC nodes. For Const() we perturb the ERC with a random, normally distributed value with mean 0.0 and standard deviation 1/3. For ConstInt() we perturb the ERC with a random, uniformly distributed integer value from interval $[-1; 1]$. For Rect() we perturb each rectangle coordinate or dimension with a random, uniformly distributed integer value from interval $[-1; 1]$. In all cases, we trim the perturbed values to domain intervals.

To speed up the selection process and to meet contest rules that required the ant code to be provided in C programming language (ECJ is written in Java), in each generation we serialize the entire population into one large text file, encoding each individual as a separate C function with a unique name. The resulting file is then compiled and linked with the game engine, which subsequently carries out the selection process, returning the identifiers of selected individuals to ECJ. As all individuals are encoded in one C file, the compilation overhead is reasonably small, and it is paid off by the speedup provided by C (compared to Java). This approach allows us also to monitor the actual size of C code, constrained by contest rules to 5kB per individual.

The best evolved ant, called *BrilliAnt* in the following, emerged in an experiment with population of 2250 individuals evolving for 1350 generations, using fitnessless selection with tournament size 5 (thus 4 matches per single-elimination tournament), and with 2×6 games played in each match. BrilliAnt has been submitted to GECCO'07 Ant Wars competition and won it. We would like to point out that BrilliAnt evolved and was selected in completely autonomous way, without support from any human-made opponent. To choose it, we ran a round-robin tournament between all 2250 individuals from the last generation of the evolutionary run. It is worth noticing that this process was computationally demanding: having only one double-game per match, the total number of games needed was more than 5,000,000, i.e., as much as for about 47 generations of evolution.

5 Human Competitiveness

The game-playing task allows for two interpretations of human competitiveness. To assess the *direct competitiveness* we implemented a simulator that allows humans to play games against an evolved ant. Using this tool, an experienced human player played 150 games against BrilliAnt, winning only 64 (43%) of them and losing the remaining 86 (57%). BrilliAnt's total score amounted to 1079, compared to human's 992. Even when we take into account the fact, that playing

Table 1. The results of a round-robin tournament between the evolved ants (in bold) and humants (plain font). Each match consisted of $2 \times 100,000$ games.

Player	Matches won	Games won	Total score
ExpertAnt	6	760,669	10,598,317
HyperHumant	6	754,303	10,390,659
BrilliAnt	6	753,212	10,714,050
EvolAnt3	3	736,862	10,621,773
SuperHumant	3	725,269	10,130,664
EvolAnt2	3	721,856	10,433,165
EvolAnt1	1	699,320	10,355,044
SmartHumant	0	448,509	9,198,296

150 games in a row may be tiring for a human and cause him/her make mistakes, this result can be definitely considered as human competitive. The reader is encouraged to measure swords with BrilliAnt using Web interface provided at http://www.cs.put.poznan.pl/kkrawiec/antwars/.

We analyzed also *indirect competitiveness*, meant as ant's performance when playing against human-designed *programs* (strategies), called *humants* in the following. We manually implemented several humants of increasing sophistication and compared them with the evolved ants using matches of $2 \times 100,000$ games. Let us emphasize that the C programming language used for that purpose offers richer control flow (e.g., loops) and more arbitrary access to game board than the GP encoding, so this gives a significant handicap to humants. Nevertheless, the first of our humants was easily beaten by an ant evolved in a preliminary evolutionary run that lasted 1000 generations with GP tree depth limit set to 7. The next one, *SmartHumant*, seemed more powerful until we increased the depth limit to 8 and equipped ant with memory. That resulted in evolving an ant that beats even SmartHumant. Having learned our lessons, we finally designed *SuperHumant* and *HyperHumant*, the latter being the best humant we could develop. HyperHumant stores states of board cells observed in the past, plans 5 moves ahead, uses a probabilistic memory model and several end-game rules (e.g., *when your score is 7, eat the food piece even if the opponent is next to it*).

To our surprise, by tuning some evolutionary operators we were able to evolve an ant, *ExpertAnt*, that wins 50.12% of games against HyperHumant. The difference in the number of games won between ExpertAnt and HyperHumant is statistically insignificant at the typical 0.01 level, but it is significant at the 0.15 level. As BrilliAnt turned out to be a bit worse than HyperHumant (loosing 52.02% of games), ExpertAnt apparently could be considered a better pick for the Ant Wars contest. However, although ExpertAnt evolved without human intervention, it has been selected by explicitly testing all ants from the last generation against the *manually designed* HyperHumant. As our intention was to evolve contestant fully autonomously, so, notwithstanding ExpertAnt performance, we decided to submit BrilliAnt to the contest as it evolved and has been selected completely autonomously. Quite interestingly, we observed also that the

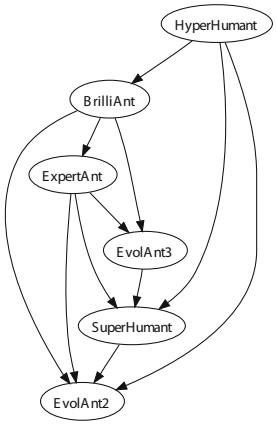


Fig. 1. Graph showing relations between players. An arrow leading from ant a to ant b means that a is statistically better than b ($\alpha = 0.01$). $2 \times 100,000$ games were played between every two ants. EvolAnt1 and SmartHumant were not showed to improve graph's readability. EvolAnt1 wins against SmartHumant only.

method used to select ExpertAnt probably promotes overfitting: despite being slightly better than HyperHumant, ExpertAnt loses against BrilliAnt (in 51.77% of games).

Table 1 presents the results of a round-robin tournament between eight ants, the five mentioned earlier and three other evolved ants (*EvolAnt**). Each participant of this contest played 1,400,000 games against seven competitors and could maximally score 21,000,000. It is hard to say which ant is the ultimate winner of this tournament. Three of them won six matches each. ExpertAnt won the most games, but it is BrilliAnt that got the highest total score.

The results of the same experiment are shown also in the form of graph in Fig. 1. An arrow leading from a to b indicates that a turned out to be statistically better than b (at 0.01 level). No arrows between ants means no statistical advantage. HyperHumant is the only player that never loses significantly and in this respect it can be considered as the winner of the tournament. Interestingly, there are no cycles in this graph and it is weakly transitive.

6 BrilliAnt's Strategy

As BrilliAnt's code is too complex to analyse it within this paper, we describe selected observations concerning its behavior. Let us start from the most obvious strategies. Faced with two corner areas of the field of view (FOV) occupied by food, BrilliAnt always selects the direction that gives chance for more food pieces. It also reasonably handles the trade-off between food amount and food proximity, measured using chessboard (Chebyshev) distance (the number of moves required to reach a board cell). For instance, given a group of two pieces of food at distance

2 ((2, 2) for short), and a group of two pieces of food in distance 1, i.e., (2, 1), BrilliAnt chooses the latter option, a fact that we shortly denote as $(2,2) \prec (2,1)$. Similarly, $(1,1) \prec (2,2)$, $(3,2) \prec (2,1)$, $(3,2) \prec (3,1)$, and $(2,2) \prec (3,2)$. If both groups contain the same number of food pieces but one of them is accompanied by the opponent, BrilliAnt chooses the other group. It also makes reasonable use of memory: after consuming the preferred group of food pieces, it returns to the other group, unless it has spotted some other food in the meantime.

Food pieces sometimes happen to arrange into 'trails', similar to those found in the Artificial Ant benchmarks [7]. BrilliAnt perfectly follows such paths as long as the gaps are no longer than 2 cells (see Fig. 2). However, when faced with a large group of food pieces, it not always consumes them in an optimal order.

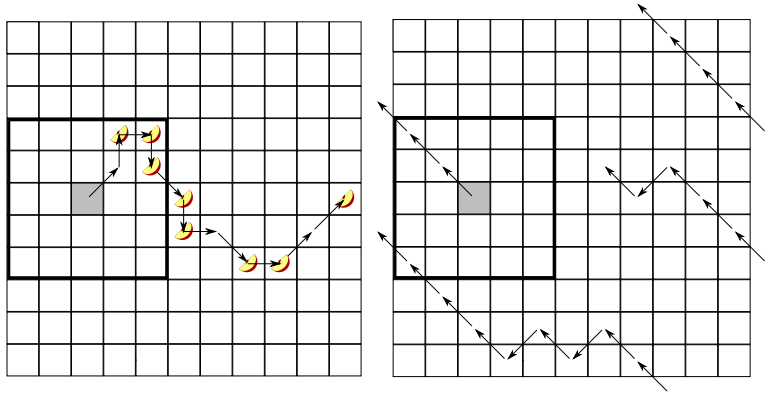
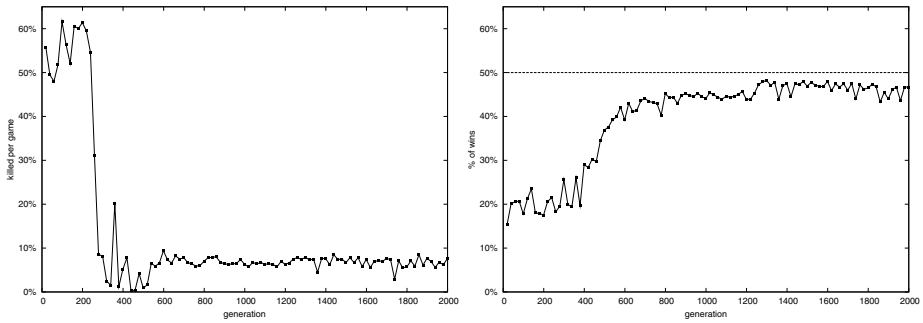


Fig. 2. Brilliant's behaviors when following a trail of food pieces (a), and in absence of food (b). Gray cell and large rectangle mark Brilliant's starting position and initial FOV, respectively.

If the FOV does not contain any food, BrilliAnt proceeds in the NW direction. However, as the board is toroidal, keeping moving in the same direction makes sense only to a certain point, because it brings the player back to the starting point after 11 steps, with a significant part of the board still left unexplored. Apparently, evolution discovered this fact: after 7 steps in the NW direction (i.e., when FOV starts to intersect with the initial FOV), BrilliAnt changes direction to SW, so that the initial sequence of moves is: 7NW, 1SW, 1NW, 1SW, 6NW, 1SW, 1NW. A simple analysis reveals that this sequence of 18 moves, shown in Fig. 2b, provides the complete coverage of the board. This behavior seems quite effective, as the minimal number of moves that scans the entire board is 15. Note also that in this sequence BrilliAnt moves only diagonally. In absence of any other incentives, this is a locally optimal choice, as each diagonal move uncovers 9 board cells, while a non-diagonal one uncovers only 5 of them.

Evolving this full-board scan is quite an achievement, as it manifests in complete absence of food, a situation that is close to impossible in Ant Wars, except for the highly unlikely event of the opponent consuming all the food earlier. BrilliAnt exhibits variants of this behavioral pattern also *after* all some food pieces have been eaten and its FOV is empty.

BrilliAnt usually avoids the opponent, unless it comes together with food and no other food pieces are in view. In such a case, it cannot resist the temptation and approaches the food, maintaining at least distance 2 from the opponent. For one food piece, this often ends in a deadlock: the players hesitatingly walk in the direct neighborhood of the food piece, keeping safe distance from each other. None of them can eat the piece, as the opponent immediately kills such a daredevil. However, there is one exception from this rule: when the end of game comes close and the likelihood of finding more food becomes low, it may pay off to sacrifice one's life in exchange for food. This in particular applies to the scenario when both players scored 7 and the food piece of argument is the only one left.



(a) Percent of deaths per game (b) Percent of wins per game

Fig. 3. Graphs show evolution dynamics for a typical process of evolution. Each point corresponds to an best-of-generation ant chosen on the basis of 2×250 games against HyperHumant. The presented values are averaged over 2×10000 games against HyperHumant. It can be noticed that the evolution process usually converges around 1300 generation when the wining rate against a fixed opponent ceases to improve.

This sophisticated 'kamikaze' behavior evolved as a part of BrilliAnt's strategy and emerged also in other evolutionary runs. Figure 3b illustrates this behavior in terms of death rate statistic for one of the experiments. The ants from several initial generations play poorly and are likely to be killed by the opponent. With time, they learn how to avoid the enemy and, usually at 200-300th generation, the best ants become perfect at escaping that threat (see Fig. 3b). Then, around 400-500th generation, the ants discover the benefit of the 'kamikaze' strategy, which results in a notable increase of death rate, but pays off in terms of winning frequency.

7 Conclusions

This paper presented an evolved game strategy that won the Ant Wars contest and has been produced by means of a novel fitnessless mechanism of selection. This mechanism lets individuals play games against each other and simply propagates the winner to the next generation, allowing us to get rid of the objective fitness. Though unusual from the viewpoint of the core EC research, selection without fitness has some rationale. The traditional fitness function used in EC is essentially a mere technical means to impose the selective pressure on the evolving population. It is often the case that, for a particular problem, the definition of fitness is artificial and usually does not strictly conform its biological counterpart, i.e., the *a posteriori* probability of the genotype survival. By eliminating this need, we avoid subjectivity that the fitness definition is prone to.

Despite its simplicity, the evolution with fitnessless selection produces sophisticated human-competitive strategies. We do not entice the evolution by providing competitive external (e.g., human-made) opponents, so that both evolution as well as selection of the best individual from the last generation are completely autonomous. Improvement of individuals' performance takes place only thanks to competition between them. Let us also emphasize that these encouraging results have been obtained despite the fact that the game itself is not trivial, mainly due to incompleteness of information about the board state available to the players.

Interestingly, in our evolutionary runs we have not observed any of the infamous pathologies common to coevolution, like loss of gradient or cycling. This may be probably attributed to the fact that our setup involves single a population. The detailed comparison of the fitnessless selection and fitness-based selection methods will be subject of a separate study.

So, is it really true that an evolved solution can be better than human's mind? Check at the page http://www.cs.put.poznan.pl/kkrawiec/antwars/ if you can beat BrilliAnt!

Acknowledgment

This research has been supported by the Ministry of Science and Higher Education grant # N N519 3505 33.

References

1. Angeline, P.J., Pollack, J.B.: Competitive environments evolve better solutions for complex tasks. In: Forrest, S. (ed.) Proceedings of the 5th International Conference on Genetic Algorithms, pp. 264–270 (1993)
2. Azaria, Y., Sipper, M.: GP-gammon: Genetically programming backgammon players. Genetic Programming and Evolvable Machines 6(3), 283–300 (2005)
3. Buro, M.: Real-time strategy games: A new AI research challenge. In: Gottlob, G., Walsh, T. (eds.) IJCIA, pp. 1534–1535. Morgan Kaufmann, San Francisco (2003)

4. Corno, F., Sanchez, E., Squillero, G.: On the evolution of corewar warriors. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, June 20-23, 2004, pp. 133–138. IEEE Press, Los Alamitos (2004)
5. Hauptman, A., Sipper, M.: Evolution of an efficient search algorithm for the mate-in-N problem in chess. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 78–89. Springer, Heidelberg (2007)
6. Koza, J.R.: Genetic evolution and co-evolution of game strategies. In: The International Conference on Game Theory and Its Applications, Stony Brook (1992)
7. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
8. Luke, S.: Genetic programming produced competitive soccer softbot teams for robocup97. In: J.R.K., et al., (eds.) Genetic Programming 1998: Proceedings of the 3rd Annual Conference, Madison, Wisconsin, USA, pp. 214–222 (1998)
9. Luke, S.: ECJ evolutionary computation system (2002), `http://cs.gmu.edu/eclab/projects/ecj/`
10. Luke, S., Wiegand, R.: When coevolutionary algorithms exhibit evolutionary dynamics. In: 2002 Genetic and Evolutionary Computation Conference Workshop Program, pp. 236–241 (2002)
11. Panait, L., Luke, S.: A comparison of two competitive fitness functions. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 503–511. Morgan Kaufmann, San Francisco (2002)
12. Shichel, Y., Ziserman, E., Sipper, M.: GP-robocode: Using genetic programming to evolve robocode players. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J.I., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 143–154. Springer, Heidelberg (2005)
13. Sipper, M.: Attaining human-competitive game playing with genetic programming. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, Springer, Heidelberg (2006)
14. Smilak, K.C.: Finding the ultimate video poker player using genetic programming. In: Koza, J.R. (ed.) Genetic Algorithms and Genetic Programming at Stanford 1999, pp. 209–217 (1999)
15. Tettamanzi, A.G.B.: Genetic programming without fitness. In: Koza, J.R. (ed.) Late Breaking Papers at the Genetic Programming 1996 Conference (1996)
16. Wittkamp, M., Barone, L.: Evolving adaptive play for the game of spoof using genetic programming. In: S.J.L., et al. (eds.) Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG 2006), University of Nevada, Reno, USA, pp. 164–172. IEEE, Los Alamitos (2006)