Evolving Defence Strategies by Genetic Programming

David Jackson

Dept. of Computer Science, University of Liverpool,
Liverpool L69 3BX, United Kingdom
d.jackson@csc.liv.ac.uk

Abstract. Computer games and simulations are commonly used as a basis for analysing and developing battlefield strategies. Such strategies are usually programmed explicitly, but it is also possible to generate them automatically via the use of evolutionary programming techniques. We focus in particular on the use of genetic programming to evolve strategies for a single defender facing multiple simultaneous attacks. By expressing the problem domain in the form of a 'Space Invaders' game, we show that it is possible to evolve winning strategies for an increasingly complex sequence of scenarios.

1 Introduction

In military contexts, implementing a defence strategy for an autonomous entity often involves highly complex computer programming. The 'intelligence' that must be built into such systems is often derived from detailed warfare simulation and the application of game theory. Indeed, it has long been recognised that there is an extensive overlap of true battlefield strategy with various forms of game playing.

In many modern computer games, artificial intelligence techniques are extensively used to increase the sophistication of the behaviour of computer opponents, and to heighten the sense of reality of the game-playing experience [1]. Such techniques are usually programmed in by the game's developers, but it is becoming increasingly apparent that evolutionary computation techniques may also have a role to play in this regard. Evolutionary programming has been used to evolve strategy for a large number of games, including chess [2], checkers [3], poker [4], Othello [5], and backgammon [6]. Many of these are board or card games involving the evolution of 'mini-max' strategies [7], but there has also been work done on problem domains with more obvious militaristic connections, including air strike planning [8], pursuer-evader scenarios [9], minesweeping [10], and missile firing [11,12].

In general, however, much less research work has been done on evolving defence strategy for responding to multiple simultaneous enemies launching unpredictable attacks in real time. In our paper, we wish to take a closer look at some of the issues involved in evolving defence strategy via genetic programming. To achieve this we couch the problem in the form of the well-known arcade game Space Invaders. This game, one of the earliest and most successful computer games ever written, is deceptively simple in concept and yet frustratingly difficult to master.

M. Keijzer et al. (Eds.): EuroGP 2005, LNCS 3447, pp. 281–290, 2005.

For those not familiar with the game, the idea is that there is a lone defender at the bottom of the screen, and a large number of aliens who descend from the skies, dropping bombs as they move. The defender's task is to shoot down all the aliens before they land, whilst avoiding being destroyed by the bombs. To achieve this, the defender has only three actions available: move left, move right, and fire a missile. However, protection may be sought beneath a number of fortifications until they too are destroyed.

The simplicity of the game makes it an ideal subject for studying the potential for devising defence strategy via evolutionary programming. More specifically, the research question we wish to address is whether, through the use of genetic programming, we can evolve programs for the defender that will enable it to win sets of increasingly complex games. In the next section we will describe the experimental approach we have used in more detail. This is followed by a description of the experiments themselves and their outcomes, and then some concluding remarks.

2 Experimental Approach

The programs that we shall evolve in the following experiments are viewed as directing the behaviour of a single defender operating within a square 'arena,' as depicted in Fig. 1. The arena takes the form of a grid of cells; for simplicity, Fig. 1 depicts this to be of size 10x10, although the experiments described here actually used an arena size of 20x20.

In evaluating the fitness of an individual, the program code is executed over a set of random tests (games). The number of such games is usually set at 50. For each test of a candidate program, the defender is placed randomly somewhere on the bottom row of the arena. The aliens materialise randomly anywhere in the top 6 rows, and are

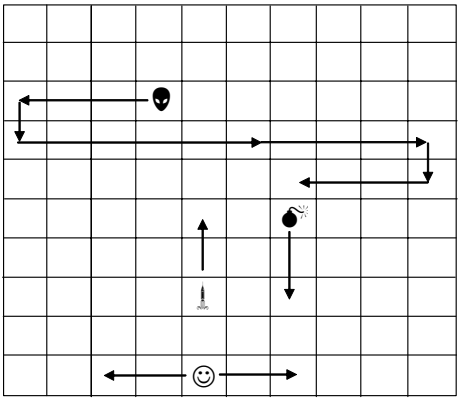

Fig. 1. Movement of entities within the battle arena

initially travelling in a randomly-chosen direction. Each alien descends in a boustrophedon pattern, i.e. across the arena, down a row, across in the opposite direction, and so on. As the aliens travel, they may release bombs. Again for

simplicity, Fig. 1 shows a single alien and a single bomb, and also a missile that the defender has launched to attack the alien. Although there may be many bombs in the air simultaneously, there can be at most only one missile; the defender must wait until an existing missile hits an enemy or leaves the arena before it launches the next.

The program code corresponding to a member of the population being evolved is built from the three primitives representing a move to the cell on the immediate left, a move right, and the firing of a missile, together with other primitives for all the decision-making associated with the strategy. A game advances in 'steps.' Execution of the left, right and fire primitives cause the game to move on by one step. The other nodes of a program tree – those corresponding to the decision making – are viewed as consuming negligible 'thinking' time, and do not cause a game step to be taken.

In each game step, all aliens, bombs and missiles move ahead one cell in their respective directions of travel. As there are no looping constructs available in the programming primitives, each program is evaluated repeatedly until a game ends. To ensure progression of 'think-only' programs, i.e. those which do not employ any of the left, right or fire primitives, a game is also advanced by a step at the end of each program iteration. In general, a game ends when all aliens are destroyed, when an alien succeeds in landing, or when a bomb hits the defender (unlike the original arcade game, this defender gets no second chances !)

In performing the following experiments, we have elected to use steady-state evolution. Candidates for both reproduction and deletion within the population are selected via tournament, acting on a sample size of 5. The population size is 500; this is initialised using the 'ramped half-and-half' method advocated by Koza [13], in which an equal number of program trees is randomly generated for each tree depth up to an initial maximum (6 in our case). For each set of trees of a given depth, half are generated 'fully,' i.e. all branches are of the same complete depth, and half are 'grown,' i.e. branches may end in a terminal node before the set depth is reached. There are no duplicates in this initial population, although no attempt is made to identify or prevent them in subsequent generations.

Evolution proceeds over 50 generations using the standard operators of crossover and fitness proportionate reproduction, selected probabilistically so that 90% of individuals are created via crossover. Crossover points are selected randomly, but with a distribution of 90% applied to function nodes and 10% applied to terminal nodes. Mutation was not used in these experiments.

3 The Experiments

3.1 Experiment 1

The first experiment begins things simply, with the defender opposed by just a single alien. Moreover, the alien has no weapons, so that a game ends when it either manages to land or is hit by a missile. The terminal set for this experiment is as follows:

{LEFT, RIGHT, FIRE, Y_DIST, X_DIST}

The first two terminals in this set, LEFT and RIGHT, simply move the defender one cell in the appropriate direction, unless the defender is against the edge of the

arena that makes such movement impossible. FIRE launches a missile from the defender's current coordinates, unless a missile is already in the air. Execution of LEFT, RIGHT and FIRE nodes during fitness evaluation all cause the game to be advanced one step, and all three nodes return a zero result to the tree evaluation function. Y_DIST returns the current vertical distance of the alien above 'ground level' and X_DIST returns its horizontal distance from the defender. The value of X_DIST is positive if the alien is currently approaching the defender, negative if it is receding.

The function set for the experiment looks like this:

{IF, EQ, PROGN2, PROGN3}

The IF function takes three arguments and is defined as:

if <arg1> then <arg2> else <arg3>

The EQ function evaluates its 2 arguments and tests them for equality, returning 1 if the results are the same and zero otherwise. PROGN2 and PROGN3 are connectives, as used in problems such as the Santa Fe artificial ant trail [13]. They simply cause each of their sub-tree arguments to be evaluated in turn, PROGN2 having two arguments and PROGN3 having three. PROGN2, PROGN3 and the IF function all return a zero result when executed.

The next problem to consider is how to define the fitness metric. Since the primary aim of the defender is to shoot down the alien, we can define fitness in terms of how close the defender's missiles come to hitting the alien. When a missile achieves the same Y-axis value as the alien, the distance considered is the absolute difference in their X-coordinates. The best of these measurements forms the value recorded for that game. This means that, if the defender manages to score a direct hit with any one of its missiles during a game, the score for that game is zero. A non-zero value at the end of the game indicates that none of the missiles hit their target, and the alien managed to land. These values are then summed over the 50 games to give a final fitness score. It follows that this score will be zero only if all games are won by the defender.

In executing the GP system for this initial problem, it was found that it was not difficult to evolve solutions. Almost every run led to a solution, usually within a handful of generations. It was also found that the winning strategies that generally appeared involved firing missiles as often as possible, whether on-target or not. Part of the reason for this is that, as long as one of the missiles hits the alien, it does not matter where the others go. If a kill is achieved, the shots that go wide will not affect the perfect game score of zero.

However, it is not sufficient merely to stand still and fire repeatedly. Programs which did this let too many aliens slip through to ground level. Rather, a successful strategy involves firing coupled with movement. Most of the zero-fitness programs achieved this by gradually migrating to the left or right edge of the arena, and then firing continually from there. The reason for this seems to be that the leftmost and rightmost columns are the only ones in which the alien performs any vertical motion, and the extra time spent lingering in those columns while the descent is achieved seems to improve the defender's chances of a direct hit. The following 41-node solution is one that works in such a way. It moves to the right edge, and then fires repeatedly from there, occasionally jumping one cell to the left and back again:

PROGN3 (IF (IF (LEFT RIGHT X_DIST) PROGN3 (FIRE RIGHT RIGHT) EQ (RIGHT RIGHT)) PROGN2 (IF (FIRE PROGN3 (EQ (X_DIST EQ RIGHT (RIGHT)) IF (FIRE RIGHT LEFT) PROGN3 (LEFT LEFT LEFT)) RIGHT) IF (RIGHT LEFT Y_DIST)) EQ (Y_DIST IF (RIGHT LEFT FIRE)))

In an attempt to encourage solutions in which accurate aiming would become more of a priority, we made a small alteration to the fitness metric. Instead of basing the fitness score on the *best* missile proximity of a game, we tried basing it on the proximities of *all* the shots fired. In this modified version of the experiment, any misses whatsoever lead to a non-zero fitness, so that zero (optimal) fitness is achieved only if each and every missile hits its target. We also alter our termination criterion so that success is defined in terms of the number of kills, rather than zero fitness. This means that a run is judged successful if it evolves a program that wins all 50 games. This 'best' program may or may not have zero fitness. The situation is similar to that used in, for example, symbolic regression problems, where the best program is often regarded as the one which scores the most matches of inputs to outputs, even though the overall error term representing its fitness value may be worse than that of other population members. Finally in this modified version of the experiment, we also introduce an additional penalty (200 points) to be added to the score each time an alien manages to land.

As before, little computational effort was required to evolve solutions, and approximately 90% of these were of the 'rapid-fire' type seen previously. The other 10%, however, were much more considered in their shooting strategies, preferring to wait until the enemy was in range before firing a missile. The following 20-node solution is one such 'aimer':

IF (EQ (Y_DIST X_DIST) PROGN3 (FIRE LEFT X_DIST) IF (PROGN3 (PROGN3 (Y_DIST Y_DIST RIGHT) LEFT PROGN2 (LEFT LEFT)) Y_DIST Y_DIST))

Execution of this program causes the defender to migrate to the left edge of the arena, from where it fires only when it calculates that a direct hit is possible. The decision is based on the X and Y distances of the alien from the defender.

3.2 Experiment 2

In this experiment, the alien fights back! It is now given the ability to drop a bomb when it is directly above the defender. However, only one bomb can be in the air at a time. This bomb moves down one cell each time the game advances, moving in lockstep with the alien, any missile, and possibly the defender. The defender cannot shoot down a bomb; it can only move out of its way. To assist it in identifying when it needs to do this, the terminal set is expanded slightly:

{LEFT, RIGHT, FIRE, Y_DIST, X_DIST, ATTACKED}

The ATTACKED terminal returns 1 if the defender is directly below a bomb, zero otherwise. We also impose a penalty of 200 points if the defender is hit by a bomb.

This experiment was tried with each of the two approaches to fitness evaluation described for the previous experiment. Using the 'best proximity' approach, in which only the closest shot in a game counts towards the final score, it proved very difficult to evolve solutions. In one set of 100 runs, only 5 of those runs resulted in solutions.

The lack of accuracy inherent in the 'rapid-fire' approach coupled with the newly-introduced need to spend time dodging the enemy bombs usually resulted in the defender being killed first or the alien landing.

The 'all-missile' approach, in which the proximity of each and every missile counts towards the fitness score, resulted in a much better performance, with a success rate of 33% in one set of 100 runs. In one run, the following 46-node solution was produced in generation 26:

IF (EQ (IF (IF (EQ (IF (ATTACKED ATTACKED Y_DIST) EQ (X_DIST Y_DIST)) LEFT EQ (EQ (X_DIST Y_DIST) ATTACKED)) RIGHT LEFT) EQ (X_DIST Y_DIST)) PROGN2 (PROGN2 (RIGHT Y_DIST) EQ (IF (ATTACKED IF (LEFT RIGHT LEFT) EQ (X_DIST Y_DIST)) EQ (X_DIST Y_DIST))) EQ (ATTACKED PROGN3 (FIRE ATTACKED RIGHT)))

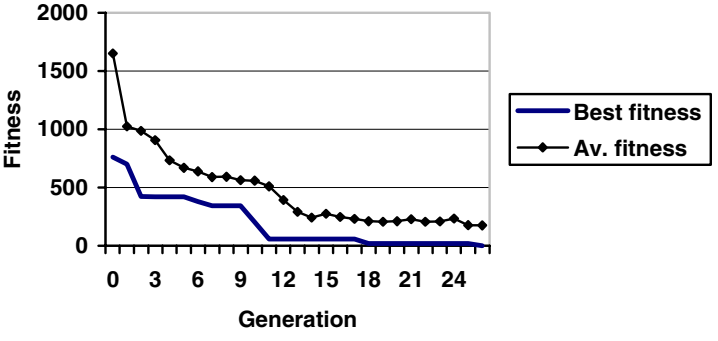

Fig. 2. Fitness graph for Experiment 2

Monitoring of the execution of this program reveals that it avoids bombs by repeatedly performing little 'dances' that involve taking two steps to the left and then one step to the right. The graph of Figure 2 shows how the best fitness and average fitness of the population changed during the evolution of this particular program.

3.3 Experiment 3

This experiment is similar to the previous one, in that there is still just a single alien and defender, but the alien is now capable of dropping a lot more bombs. As before, the alien will tend to drop a bomb when directly above the defender, but it will also drop other bombs at random on each pass across the arena. Within the GP system, the maximum number of bombs in the air at any point is controlled by a parameter MAX_BOMBS. Another restriction is that bombs cannot share the same X coordinate, i.e. only one bomb can be present in each column of the arena.

In performing this experiment we abandoned the 'best proximity' approach to measuring fitness, which performed so poorly in the previous experiment. Using the 'all missiles' approach instead, we were able to evolve programs that could cope with large numbers of bombs. In one set of runs, for example, we found that it was possible to evolve programs that could cope with MAX_BOMBS set to 10, i.e. with up to half

the sky containing bombs. In one such run, the following 19-node program evolved in generation 35:

IF (ATTACKED PROGN2 (ATTACKED LEFT) IF (EQ (X_DIST Y_DIST)
IF (EQ (X_DIST Y_DIST) IF (Y_DIST FIRE X_DIST) Y_DIST) X_DIST))

In executing this strategy, the defender moves as little as possible – enough to evade the bombs. It does this as soon as a bomb appears above it, while the bomb is still high in the air. This gives the defender time to check if it is still in danger after moving, so that it can take further evasive action if required. As soon as the enemy comes into range, the defender launches a missile to bring it down.

3.4 Experiment 4

In this final experiment, we take a step closer to the original game by introducing multiple aliens. A consequence of this is that the function set (F) and terminal set (T) are altered as follows:

T = {LEFT, RIGHT, FIRE, ATTACKED, TARGET_LEFT, TARGET_RIGHT}
F = {IF, PROGN2, PROGN3}

It will be seen that the tokens X_DIST, Y_DIST and EQ have all been removed from these sets. The existence of the first two of these in particular no longer makes any sense, since they refer to a single alien, and we now have many. Moreover, their primary purpose was to enable the evolution of exact targeting, and it has already been demonstrated in the previous experiments that such an ability can be readily evolved.

In their place we now have TARGET_LEFT and TARGET_RIGHT. These terminals return 1 if a missile launched from one place to the defender's left/right at the present time would lead to a better shot (i.e. closer proximity to any alien) than one launched from the current position; otherwise they return zero. These primitives act only as 'hints' as to how to move in order to improve the chances of a missile launch being on target, since they take no account of the time-consuming steps that may be associated with the other nodes of the program tree. By the time the defender has physically moved to a new position that it calculates to be better, the game may have advanced several steps and the aliens will all have changed positions. This makes life more difficult for the defender, but it may also encourage the evolution of strategies that involve more proactive 'pursuit' of the invaders.

The effort involved in evolving solutions to this problem is obviously affected by the value of MAX_BOMBS, and by a new parameter MAX_ALIENS. A useful metric here is provided in Koza's definition of computational effort [13], which calculates the minimum number of individuals that need to be processed in order to attain a probability of 0.99 that a solution will be evolved for a given population size. If MAX_BOMBS and MAX_ALIENS are both set to 3, for example, the success rate over 100 runs is 47%, and the computational effort required turns out to be 58,500 individuals, representing 13 runs to generation 8. If MAX_BOMBS and MAX_ALIENS are each increased to 5, the success rate degrades to 12%, while the computational effort jumps to 675,000 individuals (75 runs to generation 17). Figure 3 depicts the changes in best and average fitness during the evolution of one of these latter programs over 26 generations.

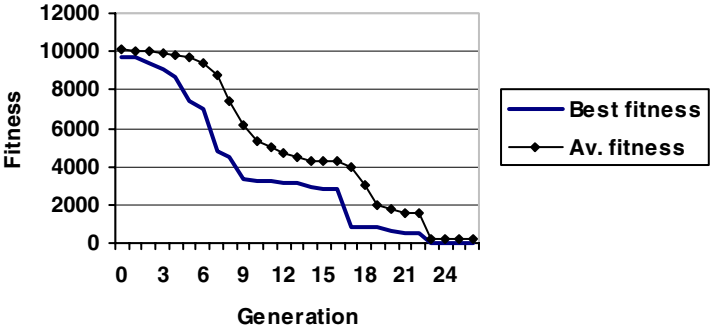

Fig. 3. Fitness graph for Experiment 4

For most of the solutions produced, movement of the defender is usually associated with dodging the bombs that rain down. However, when the bombs are very few in number, these evasion tactics become less important, while the need to destroy the aliens before they land becomes more so. In such scenarios, the pursuit strategy mentioned earlier may become apparent, with the defender attempting to place itself in optimum positions for ensuring the accuracy of its missiles.

4 Conclusions

It has been shown that by using genetic programming it is possible to evolve defence strategies to cope with battle games of significant difficulty. Such strategies incorporate many of the aspects of evasion, pursuit and targeting behaviour found in human game players. It has also been interesting to see the evolution of individuals exhibiting varying forms of game strategy, such as the 'rapid-firers' and the 'aimers.'

Although the individuals that satisfy the termination criteria of a run are often referred to in this paper as 'solutions,' this is in fact a slight exaggeration of the truth. To be accurate, what these programs represent are defence strategies that have demonstrated their ability to win a sequence of 50 consecutive random games. It is sometimes found that such programs can and do lose games when tested further; however, experimentation suggests that the 50 game test sequence is sufficient to promote fairly good generality. For example, the 'solutions' produced in Experiment 4, with MAX_BOMBS and MAX_INVADERS both set to 3, were further tested in another sequence of 50 games that had not been used as a 'training set' during the evolutionary process. It was found that, on average, 45 of these 50 games could still be won, and that 10% of the programs won all of these 50 additional games. It would of course be possible to increase confidence in the robustness of strategies by simply raising the number of tests performed during fitness evaluation; this has to be balanced against the additional computational expense incurred.

As we have moved through the series of experiments described above, we have gradually increased the level of sophistication of the games. Future plans are to see how much further this can be taken. An obvious future addition is the inclusion of buildings behind which the defender can seek refuge. We would also like to

investigate the effects of heightening the sensory and decision-making powers of the defender. One way of achieving this would be to endow it with the ability to sense the distance of bombs, and not just their mere presence, so that it might be able to make more informed decisions about exactly when it should dodge an approaching bomb. Another way might be to allow it somehow to analyse the positions of all the aliens simultaneously and assign priorities to them, so that perhaps the most threatening enemies are eliminated first. Yet another suggestion is to endow the aliens with more sophisticated strategies, perhaps to be co-evolved with those of the defender.

Finally, it should be noted that, since fitness evaluation of the strategies being evolved essentially involves execution of a set of games from start to finish, it is possible to bolt a simple visual interface onto the fitness function to view these games. This was in fact done in the experiments described above, providing a fascinating insight into defence strategies created entirely without explicit human programming. A screen-shot of one of the games being played out for a solution evolved in Experiment 4 is shown in Figure 4.

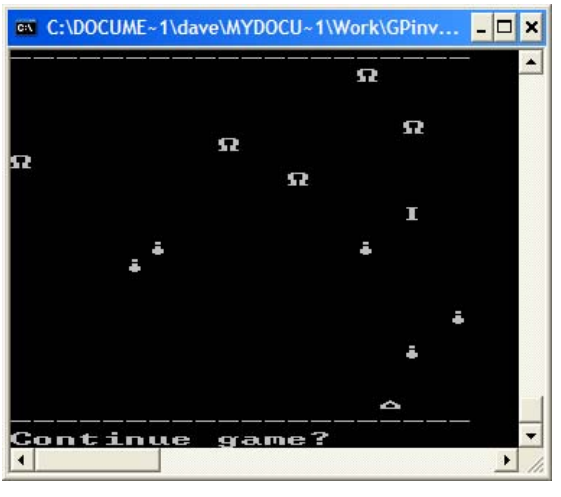

Fig. 4. Visual interface to the GP system during fitness evaluation

References

1. Laird, J.E.: Research in Human-Level AI Using Computer Games. Communications of the ACM 45(1) (2002) 32-35
2. Gross, R., Albrecht, K., Kantschik, W., Banzhaf, W.: Evolving Chess Playing Programs. In: Langdon, W.B. et al (eds.): GECCO 2002. Morgan Kaufmann, San Francisco, CA (2002) 740-747
3. Chellapilla, K., Fogel, D.B.: Evolving an Expert Checkers Playing Program without Using Human Expertise. IEEE Trans. on Evolutionary Computation 5(4) (2001) 422-428
4. Kendall, G., Willdig, M.: An Investigation of an Adaptive Poker Player. In: 14th Australian Joint Conf. on Artificial Intelligence, Lecture Notes in Artificial Intelligence, vol. 2256. Springer-Verlag, Berlin Heidelberg (2001) 189-200

5. Eskin, E., Siegel, E.V.: Genetic Programming Applied to Othello: Introducing Students to Machine Learning Research. In: Proc. 30th Technical Symposium of the ACM Special Interest Group in Computer Science Education (SIGCSE), New Orleans, LA, USA (1999) 242-246
6. Pollack, J., Blair, A.D., Land, M.: Coevolution of a Backgammon Player. In: Langton, C.G. and Shimohara, K. (eds.): Artificial Life V: Proc. 5th Int. Workshop on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA (1996) 92-98
7. Koza, J.R.: Genetic Evolution and Co-Evolution of Game Strategies. In: International Conf. on Game Theory and its Applications, Stony Brook, New York (1992)
8. Miles, C., Louis, S.J., Cole, N.: Learning to Play Like a Human: Case Injected Genetic Algorithms Applied to Strategic Computer Game Playing. In: Congress on Evolutionary Computation (CEC 2004), Portland, Oregon, USA (2004)
9. Moore, F.W., Garcia, O.N.: A Genetic Programming Approach to Strategy Optimization in the Extended Two-Dimensional Pursuer/Evader Problem. In: Koza, J.R. et al (eds.): Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan Kaufmann, San Francisco, CA, USA (1997) 249-254
10. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA, USA (1994)
11. Moore, F.W.: A Methodology for Missile Countermeasures Optimization under Uncertainty. Evolutionary Computation 10(2). MIT Press, Cambridge, MA, USA (2002) 129-149
12. Nyongesa, H.O.: Generation of Time-Delay Algorithms for Anti-air Missiles using Genetic Programming. In: Boers, E.J.W. et al (eds.): EvoWorkshop 2001, Lecture Notes in Computer Science, vol. 2037. Springer-Verlag, Berlin Heidelberg (2001) 243-247
13. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)