Speeding up Genetic Algorithm-based Game Balancing using Fitness Predictors*

Mihail Morosan
University of Essex
Colchester
mmoros@essex.ac.uk

Riccardo Poli
University of Essex
Colchester
rpoli@essex.ac.uk

ABSTRACT

Genetic Algorithms (GAs) can find game parameters that fit a designer's requirements. An issue with this is the long time taken to evaluate fitness, as this requires running the game many times. Here we use fitness predictors, currently neural networks, to speed up the process by reducing the number of fitness evaluations. The predictors are trained using data generated by the GA at runtime. After training, the model is invoked to estimate the fitness of newly created individuals. If the estimate is below a threshold, it is accepted. Otherwise, the original fitness function is invoked. We have used this approach on *Ms PacMan* with promising results.

KEYWORDS

Genetic algorithm, neural network, PacMan, prediction, balancing

ACM Reference format:
Mihail Morosan and Riccardo Poli. 2017. Speeding up Genetic Algorithm-based Game Balancing using Fitness Predictors. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3076011

1 INTRODUCTION

Genetic Algorithms (GAs) can find game parameters that fit a designer's requirements [5]. An issue with this is the long time taken to evaluate fitness, as this requires running the game many times. Here we use neural-network (NN) based predictors, trained at runtime, to quickly estimate the fitness of individuals and to decide if they should be evaluated.

Fitness approximation is an actively explored research area [1] because in many problems fitness evaluation is expensive. For instance, [4] used NNs to simulate the actual fitness evaluation in the evolution of music using interactive genetic programming, to reduce human effort. Also, NNs were used as surrogate models for fitness evaluations to optimise radiotherapy treatment [2]. A GA, with the help of the NN, was successful in finding better solutions compared to traditional methods used in practice. The implementation used a pre-trained NN as a surrogate model which required expert knowledge and data on the subject.

*This work is supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1]

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3076011

2 METHODOLOGY

2.1 Ms Pacman

Ms PacMan is a single player game played on a 2–D board, where the player controls the "PacMan" and has the goal of collecting as many points as possible. There are 4 ghosts opposing the player. We altered the game's rules so that players win if they reach 1600 points. So, players can have a win-rate associated to their performance.

To automate the control of the "PacMan", an MCTS agent based on work by Ikehata and Ito [3] was used. It is one of the best performing agents available, but also very expensive computationally.

The metric the game designer can assess is the win-rate, being able to evolve a harder variation of the game by targeting a lower win-rate, or an easier one by aiming for a higher win-rate.

There are 9 parameters evolved: the PacMan's speed, the chasing speeds of each of the 4 ghosts, as well as their fleeing speed.

There are two objectives in the fitness evaluation: the score fitness and the parameter fitness. After running a number of games using the modified parameters, the resulting scores give us a *score fitness value* (W). The *parameter fitness* (Δ_P) is a a sum of the absolute values of the parameters, representing how far the solution is from the default game parameters.

Formally, the fitness function can be written as: $Fitness = W + \Delta_P$, where $W = |WR - DWR| \times C_W$ and $\Delta_P = \sum_{i=0}^{n} |\Delta_i| \times C_\Delta$. In the *score fitness* component W, WR = win-rate (from 0 to 1), DWR = desired win-rate, C_W = win-rate weight. In the *parameter fitness* component Δ_P, Δ_i = difference between the original ith parameter and the evolved ith parameter, C_Δ = parameter difference weight. C_W and C_Δ are the values that the game designer can change to give each fitness component a different importance.

Here, $DWR = 0.5$, representing a win-rate of 50%, $C_W = 5000$ and $C_\Delta = 100$. This gives the two objectives similar relevance, as $|WR - DWR|$ can only be a value between 0 and 1, while the sum in the Δ_P fitness component can be a value between 0 and 45.

The metrics collected are the number of fitness evaluations done during a set number of generations, the best fitness achieved and the actual time it took to complete each run.

2.2 Genetic algorithm

The evolutionary algorithm employed is a variant of a generational GA with two-point crossover (applied with a rate of 35%), a specialised mutation operator (applied with a per-individual rate of 35%) and elitism (applied to the top 15% of the population). The final 15% of the each generation is randomly sampled from the search space through reinitialisation.

The mutation operator was applied with a (per allele) mutation rate of 0.5 (meaning that on average 50% of the elements of an individual would be mutated). At each application of the operator, a

displacement is randomly generated by a random number generator within the range of −0.1 and +0.1 and added to the corresponding parameter value. Should the new value be outside the range of accepted values, it is clamped to be within those values.

The population size was 50. We set the tournament size to 6. Experiments ran until 2000 evaluations of the fitness function were completed or 100 generations passed, whichever happened first.

Each evolved vector was constrained to only contain values between -3 and 5. Values outside of these ranges are not realistic for the tested scenario.

2.3 Neural network

The NN employed for these experiments is a feed-forward NN with a single hidden layer, using the sigmoid activation function. It is initialised with random weights. The training is done via backpropagation.

The number of input neurons is equal to the number of parameters the GA is evolving. The number of output neurons is equal to how many fitness objectives are being tracked (2 in this experiment). The number of hidden layers and hidden layer neurons was chosen arbitrarily to be 1 and 18 respectively.

The network is trained using data generated by the GA. For every evaluation of an individual, the parameters and fitness values are passed to the predictor's data set. These inputs and outputs are then normalized to be within the 0 to 1 range, based on $Range_{Min}$ and $Range_{Max}$ ($Range_{Min} = -3$ and $Range_{Max} = 5$) for inputs, and 9500 (5000 for the *score fitness* component W plus 4500 for the *parameter fitness* component Δ_P) for outputs.

The data set is split into a training set (80% of the data set) and a validation set (the remaining 20% of the data set). When adding a new individual to the set, if there are more individuals than a given maximum (N_{Max}) in it, the oldest one is removed. No training is only done unless there is a minimum number of individuals in the data set (N_{Min}). Both N_{Min} and N_{Max} are set to 100.

2.4 Integrating the predictor within the GA

The predictor is trained at the end of each GA generation on data it has collected until that point. This is done for a number of epochs (N_{Epochs}), which was set to 100. An error value is computed on the validation set. Accuracy of the predictor is then computed based on the median of all fitnesses in the first generation and the error computed after testing on the validation set. $Accuracy = 1 - (VError/(Median_1 * ErrorRange))$ and $ErrorRange = 0.10$.

$VError$ represents the validation error and $Median_1$ is the median of all fitnesses after the first generation, normalized to be between 0 and 1 identically to how the outputs were normalised during data gathering. For the NN we designed, we aimed for the its results on the validation set to be within 10% ($ErrorRange$) of the actual results. Satisfying this condition meant the predictor would receive a positive accuracy rating.

The predictor is then called during the GA's evaluation of each individual. Should the computed accuracy be lower than 0 ($Accuracy <= 0$) the GA carries on with the standard evaluation of the individual.

If the predictor has a positive accuracy, the individual's parameters to it are passed to it. The predictor normalises the input and calculates the NN's output based on it. This output is considered the prediction of the fitness values of that individual. If the prediction represents a fitness that is worse than a given threshold value $Threshold$, that prediction is simply accepted and the standard evaluation of the individual is skipped. Otherwise, the individual is evaluated normally. For this experiments, $Threshold$ is equal to the median of all fitnesses in the previous generation.

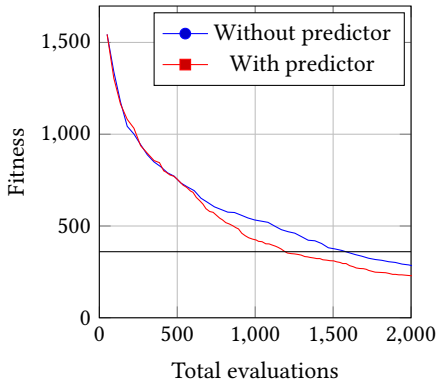

Figure 1: Average best fitness values achieved after a given number of evaluations. Lower values are better.

2.5 Experiments

We ran 20 runs normally, without the predictor, and then the same number of runs with the predictor. To better compare results, we paired each run without a predictor to one with a predictor. Each pair had the same starting populations and random seeds.

3 RESULTS

To achieve an acceptable level of fitness (360) it took the GA without a predictor, on average, 1641 evaluations. The GA with the predictor needed, on average, 1189 evaluations. This represents a speed improvement of 28% and can be observed in Figure 1.

A Wilcoxon signed rank test on the number of evaluations required to achieve the good enough fitness with the null hypothesis that using the predictor is worse, gives us a p value of $p = 0.042$. The difference is thus statistically significant.

Although there is an overhead with the training of the NN, it is minimal compared to the cost of even a single evaluation.

REFERENCES

[1] Maumita Bhattacharya. 2013. Evolutionary Approaches to Expensive Optimisation. *Arxiv - Computers & Society* 2, 3 (2013), 53–59. DOI: http://dx.doi.org/10.14569/IJARAI.2013.020308

[2] Joana Dias, Humberto Rocha, Brgida Ferreira, and Maria do Carmo Lopes. 2014. A genetic algorithm with neural network fitness function evaluation for IMRT beam angle optimization. *Central European Journal of Operations Research* 22, 3 (9 2014), 431–455. DOI: http://dx.doi.org/10.1007/s10100-013-0289-4

[3] Nozomu Ikehata and Takeshi Ito. 2011. Monte-Carlo tree search in Ms. Pac-Man. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE, 39–46. DOI: http://dx.doi.org/10.1109/CIG.2011.6031987

[4] Brad Johanson and Riccardo Poli. 1998. *GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters.* Technical Report.

[5] Mihail Morosan and Riccardo Poli. 2017. Automated Game Balancing in Ms PacMan and StarCraft Using Evolutionary Algorithms. In *EvoApplications 2017*. DOI: http://dx.doi.org/10.1007/978-3-319-55849-325