Efficient Parameter Tuning for NSGA-II in Training Multi-Objective Generalist Game-Playing Agents

Evolutionary Computing (X_400111) – Group 17

Evaline Bosch
2753468

Jacco Broere
2749073

Caspar Hentenaar
2778609

Gijs van Meer
2754711

, ,
.

1 INTRODUCTION

Multi objective optimization problems are commonly solved and well-suited for evolutionary algorithms, NSGA-II is a particular evolutionary algorithm designed for solving these Multi-objective problems efficiently and effectively [6]. The use of NSGA-II introduces a set of hyperparameters that could be tuned to improve the model performance. Parameter tuning is an important subject in modern machine learning [2, 14, 17], similarly it has proven valuable in a multitude of evolutionary algorithms as well [13, 15].

Tuning hyperparameters can often be an inefficient process when using naive methods, such as a grid search [9]. To improve on this inefficiency, more informed methods were developed, which incorporate pruning and sampling based on, e.g., successive halving and Bayesian optimization. These approaches have proven effective in hyperparameter search [10] and thus could also prove effective in tuning for hyperparameters of evolutionary algorithms.

Given these findings, this report aims to answer the question: *"Does efficient parameter tuning increase the performance of NSGA-II training a multi-objective, generalist game-playing agent?"* Moreover, the report tests the generalizability of the algorithm using differently-sized objective groups during training phases. To attain this two tuned NSGA-II networks will be trained and tested. They will be compared against the results of untrained NSGA-II networks as benchmark comparisons. Furthermore, the results obtained in [5] will serve as a benchmark paper to compare against.

2 METHODOLOGY

2.1 Evolutionary Algorithms

To investigate the effect of hyperparameter tuning, we set up a baseline evolutionary algorithm (EA) with default parameters and an equal algorithm, however, with the hyperparameters optimized using tuning. Further details on the tuning methodology are discussed in section 2.2.

Both EAs utilize a single-layer perceptron network with 10 hidden units as the GA10 in [5] to translate the observed environment into actions. The 265 weights of this network are trained using the EAs as described here.

Firstly, we outline the components of the baseline EA, as the second EA is a version of the baseline algorithm, but it incorporates (hyper)parameter tuning. This paper makes use of the Non-dominated Sorting Genetic Algorithm II (NSGA-II) based on [6]. NSGA-II attempts to alleviate the three main problems that multi-objective evolutionary algorithms face: their computational complexity, non-elitism approach, and the need of specifying a sharing parameter. NSGA-II solves these concerns in various ways. NSGA-II achieves a lower computational complexity as explained in [6]. Elitism is ensured by constructing a population consisting of both the parents and the offspring. Due to this procedure, the current best solutions are preserved. The sharing parameter is replaced by a comparison operator ensuring diversity. Comparisons in NSGA-II are drawn first by non-domination rank and ties are broken using crowding distance as described in [8]. By incorporating crowding distance in the comparisons between individuals, a diversity of solutions is ensured.

The baseline NSGA-II we employ is implemented in the *pymoo* package for Python [4]. The default setup for NSGA-II in *pymoo* uses Simulated Binary Crossover (SBX), described in [7], as its crossover operator, with $\eta_c = 15$ and $p_c = 0.9$, denoting the scale parameter of the exponential distribution and the probability of a single weight being included in crossover respectively. The default mutation operator is polynomial mutation as described in [7], with $\eta_m = 20$ and $p_m = 0.9$, also denoting the scale parameter of the exponential distribution and the probability of a weight being mutated respectively. Furthermore, parent selection is performed by tournament selection [12], using tournament size $\kappa = 2$.

In the second EA two more operators are included as options to choose from during the tuning of the algorithm. Namely, a Gaussian mutation operator is included which updates the weights of the neural network by adding noise sampled from a normal distribution with a scale parameter, σ. Parent-Centric crossover (PCX) is included as a second crossover option, introducing two scale parameters for sampling from the normal distribution. Moreover, the tournament size, κ, is also optimized during tuning. The choice of operators and their final parameters is made using the procedure outlined in 2.2.

2.2 Tuning

The setup of the NSGA-II as described in section 2.1 introduces a set of hyperparameters that can be tuned in an attempt to increase performance on the task at hand. We propose the use of the open-source hyperparameter optimization framework *Optuna* [1]. *Optuna* addresses problems inherent to common hyperparameter optimization approaches, such as the requirement of a static search space for the parameters, through the employment of dynamically constructed parameter spaces. Moreover, *Optuna* improves on the cost-effectiveness by implementing efficient sampling and pruning mechanisms.

Our setup of *Optuna* employs a Tree-structured Parzen Estimator (TPE) [3], which fits a pair of Gaussian Mixture Models (GMM). For efficient pruning, *Optuna* utilizes Asynchronous Successive Halving (ASHA)[11]. Using this framework requires defining a hyperparameter space to sample from. Using the notation from section 2.1. Probability parameters, p, are sampled from $[0, 1]$, scale parameters η for the exponential distribution from $[0, 50]$, scale parameters, σ, for the normal distribution are sampled from $[0.1, 2]$ and the tournament size κ from $\{2, \ldots, 8\}$. The framework then uses the TPE to continuously sample from this hyperparameter space efficiently through Bayesian optimization principles [1].

2.3 Experimental Setup

Two different groups of enemies are considered for which the multi-objective optimization performance is recorded, namely, the three enemy group $\{2, 5, 7\}$ and the four enemy group $\{2, 6, 7, 8\}$. We found that the behaviour learned on these groups generalized best to all eight enemies through while keeping computing time feasible. The default NSGA-II and NSGA-II incorporating parameter tuning are trained on both enemy groups. Parameter tuning is performed using 50 *trials* per enemy group, i.e., iterations of hyperparameter tuning. In each trial, the NSGA-II is trained for 10 generations with a population size equal to 25, sampling new hyperparameters each trial as outlined in 2.2. The best parameters for both enemy groups are then stored for use in the final algorithm runs. Both

enemy groups and two sets of hyperparameters yield 4 setups to be evaluated. Finally, for each of these setups, the NSGA-II is trained using a population size of 50 and 30 generations. Ten runs are executed for each setup over which results are averaged. For the fitness plots, we use the mean average and maximum average fitness scores. To show the individual gain of the best individuals from these 10 runs per EA, we collect the average individual gain from 5 repetitions of a single test against all 8 enemies. Lastly, we use the package *SciPy* in Python to perform the t-tests [16].

3 RESULTS

3.1 Parameters

Table 1 shows the different parameters that have been found during the tuning. We find that after tuning the parent-centric crossover (PCX) is preferred over the simulated binary crossover (SBX) operator and that polynomial mutation (PM) is preferred over Gaussian mutation which is neither used in the baseline. Furthermore, we see the crossover parameters for both tuned networks being relatively close to each other whereas the mutation parameters diverge more.

EA	*Crossover*		*Mutation*		*Tournament selection*
Benchmark	SBX	$\eta = 15$ $p = 0.9$	PM	$\eta = 20$ $p = 0.9$	$\kappa = 2$
Tuned for 2, 5, 7	PCX	$\sigma_1 = 0.57$ $\sigma_2 = 1.68$	PM	$\eta = 38.12$ $p = 0.83$	$\kappa = 3$
Tuned for 2, 6, 7, 8	PCX	$\sigma_1 = 0.87$ $\sigma_2 = 1.58$	PM	$\eta = 28.60$ $p = 0.24$	$\kappa = 2$

Table 1: Optimized hyperparameters using *Optuna*

3.2 Fitness

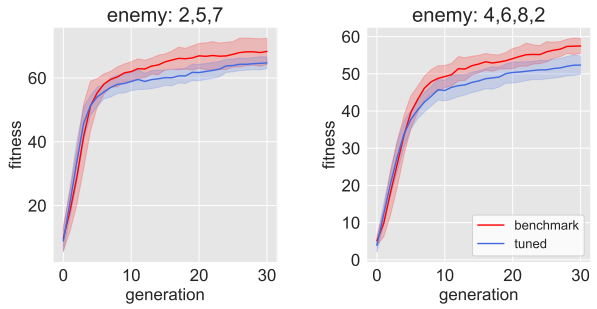

Figure 1: Mean fitness over 30 generations and averaged over 10 runs

The mean fitness averaged over 10 runs is plotted in figure 1. For both enemy groups, the mean fitness is slightly higher in the early generations for the EA incorporating parameter tuning. After 4 to 5 generations, however, the mean fitness for the tuned algorithm develops more slowly and is lower than the benchmark.

Figure 2 displays the maximum fitness averaged over 10 runs. For the EAs trained on the group of 4 enemies, similar patterns as for the mean fitness plots emerges. The EA incorporating parameter tuning achieves a higher maximum fitness early on, falling behind in later generations. For the EA trained on the group of three enemies the performance of both the tuned NSGA-II and the benchmark develop similarly, but the former finishes lower. Note that maximum fitness is the highest mean fitness value over the enemy group present in the population. Table 2 contains the results of the t-test for the difference in mean and maximum fitness between the tuned and non-tuned algorithms. These tests point out that the mean and maximum fitness are significantly higher at a 5% confidence level when using the benchmark NSGA-II.

Statistic	*Enemy group*	*t-score*	*p-value*
Mean fitness	2, 5, 7	2.508	0.027
	2, 6, 7, 8	5.061	<0.01
Maximum fitness	2, 5, 7	4.97	<0.01
	2, 6, 7, 8	2.465	0.024

Table 2: Welch's t-test for the difference in max and mean fitness between tuned and non-tuned NSGA-II results at generation 30

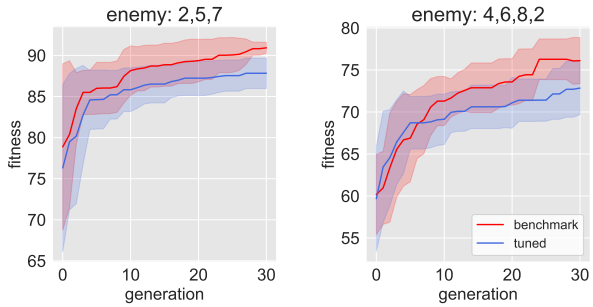

Figure 2: Max fitness over 30 generations and averaged over 10 runs

3.3 Gain

Figure 3 shows the highest individual gain averaged over 5 repetitions and 10 runs, for both the tuned and benchmark EA on all 8 enemies. These results show that the median for both the benchmark and tuned EA trained on enemies 2, 5, and 7 is higher than that of the EAs trained on enemies 2, 6, 7, and 8. Moreover, from the t-tests in table 4 it can be inferred that the mean gain does not differ significantly at a 5% confidence level between the tuned and non-tuned NSGA-II. This applies to both enemy groups. Table 3 shows player remaining energy averaged over 5 repetitions for the best-performing individual in all runs. This is an individual found during the sixth run of the tuned EA whilst training on enemies 2, 5, and 7. It can be seen that this individual not only defeats the enemies that it was trained on but also enemy 8. Furthermore, the behaviour of the player causes enemy 1 to glitch and walk into the wall, resulting in a draw. Table 3 also displays the remaining player energy for the best performing EA's employed by [5] which use the same neural network configuration as utilized here.

4 DISCUSSION

4.1 Results

As the evolutionary algorithm is subject to stochastic behaviour the limited number of trials (50) used for tuning might have led to a

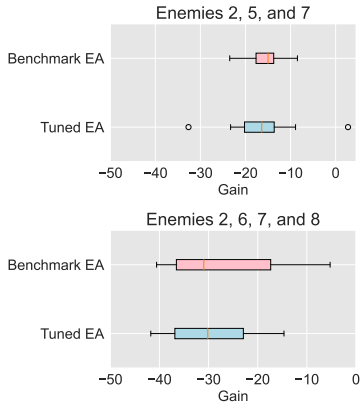

Figure 3: Box plot of highest gain individuals

Enemy number	1	2	3	4	5	6	7	8
NSGA-II (2, 5, 7)	48	82	0	0	57.2	0	74.2	31
GA-10 (7,8)	0	74	0	43	60	0	0	30
LO-10 (1,2,5)	96	74	11	0	71	0	0	44

Table 3: Average player and enemy life after 5 repetitions for the best-performing individual. Results for best performing GA-10 and LO-10 as in [5] are also provided. Training enemy group is supplied in parentheses.

Enemy group	*t-score*	*p-value*
{2, 5, 7}	-0.089	0.931
{2, 6, 7, 8}	-0.562	0.582

Table 4: Welch's t-test for the difference in mean gain between the tuned and non-tuned (benchmark) NSGA-II.

suboptimal parameter setting and operator choice. The parameter tuning resulted in use of a different crossover operator, PCX instead of simulated binary crossover. Another difference was the four enemy variant having a higher tournament pressure of 3 vs 2.

Observing the results in figures 1 and 2 the tuned EA achieves slightly worse results among all metrics at the final generation. The first 5 generations, however, are generally better for the tuned EA. This suggests that the tuning procedure found parameters that are effective for short training periods, which is in conformity with the low number of iterations that the hyperparameter tuning was performed for due to computational limitations. The resulting set of hyperparameters is therefore not necessarily optimal for longer training periods.

Figures 1 and 2 show that the 3 enemy group performs better on average than the 4 enemy group. This might indicate that when a larger group is chosen, a larger number of individuals and generations might be needed to create a generalized model. One would expect however that the 4 enemy variant has a higher ceiling of performance, but due to the time constraints and issues with implementing the model, this could not be verified.

However, we did find that the best individual was produced by the tuned EA trained on the three enemy group. This individual consistently beat enemies 2, 5, 7, and 8. It will also cause enemy 1 to glitch and walk into the wall. This does show that the tuned network is able to generalize to certain enemies it is not trained on. It is also striking that an individual trained on 3 enemies is better than one trained on four enemies. This could be caused by which enemies the EA was trained on since it might be easier to generalize with certain groups of enemies.

Comparing the best results to the baseline paper, which are shown in table 3, we find that performance in comparison to the similar GA-10 achieves comparable results. Disregarding the crashed result on enemy 1, the model is able to beat a similar amount of enemies with similar scores. On average the NSGA-10 performs better against winning enemies. The LO-10 model still slightly outperforms it, however, this could change without the glitch concerning enemy 1.

4.2 Future work

Further investigation of the effects of parameter tuning for NSGA-II could continue by increasing the number of trials for tuning or decreasing the search space for the hyperparameters. Removing the choices for different operators in the tuning procedure could lead to more robust results, along with a higher number of trials during tuning. Another important direction for future research could be increasing the population size and number of generations while tuning, more closely resembling the final training phase. Clearly, increasing these tuning parameters would cause the duration of the tuning procedure to increase. A balance between tuning and training effort should be found as the results found due to better tuning might also be obtained by running a worse-tuned EA for more generations. Lastly, further research into which enemy groups cause the EAs to generalize better might also benefit the EAs.

5 CONCLUSION

In the introduction of this piece the question: *"Does efficient parameter tuning increase the performance of NSGA-II training a multi-objective, generalist game-playing agent?"* was posed. The results don't allow for an affirmative answer to this question. What has been found however is that for parameter tuning in problems similar to the one this paper covered it is important to choose a larger number of generations to assure the model tunes to finding consistent and robust improvements instead of short-term improvements.

Another possible finding is that there might be a limit to the number of enemies one would want to train on when trying to create a generalized strategy, especially when working with a lower amount of generations and population. This was supported by the fact the three enemy variant found a more generalized solution than the four enemy variant, indicating there is some balance needed between enemy count and population/generation size for generalization. Given this observation, it can be stated this is a time versus performance balance to be evaluated.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

[2] Raj Kumar Batchu and Hari Seetha. 2021. A generalized machine learning model for DDoS attacks detection using hybrid feature selection and hyperparameter tuning. *Comput. Networks* 200 (2021), 108498. https://doi.org/10.1016/j.comnet.2021.108498

[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[4] J. Blank and K. Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[5] Karine da Silva Miras de Araujo and Fabrício Olivetti de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. 1303–1310. https://doi.org/10.1109/CEC.2016.7743938

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[7] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. 2007. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 9th annual conference on genetic and evolutionary computation*. 1187–1194.

[8] Félix-Antoine Fortin and Marc Parizeau. 2013. Revisiting the NSGA-II crowding-distance computation. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, Christian Blum and Enrique Alba (Eds.). ACM, 623–630. https://doi.org/10.1145/2463372.2463456

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[10] Manoj Kumar, George E. Dahl, Vijay Vasudevan, and Mohammad Norouzi. 2018. Parallel Architecture and Hyperparameter Search via Successive Halving and Classification. *CoRR* abs/1805.10255 (2018). arXiv:1805.10255 http://arxiv.org/abs/1805.10255

[11] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Ben Recht, and Ameet Talwalkar. 2018. Massively parallel hyperparameter tuning. (2018).

[12] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.

[13] Volker Nannen, Selmar K. Smit, and Ágoston E. Eiben. 2008. Costs and Benefits of Tuning Parameters of Evolutionary Algorithms. In *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings (Lecture Notes in Computer Science)*, Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume (Eds.), Vol. 5199. Springer, 528–538. https://doi.org/10.1007/978-3-540-87700-4_53

[14] Patrick Schratz, Jannes Muenchow, Eugenia Iturritxa, Jakob Richter, and Alexander Brenning. 2018. Performance evaluation and hyperparameter tuning of statistical and machine-learning models using spatial data. *CoRR* abs/1803.11266 (2018). arXiv:1803.11266 http://arxiv.org/abs/1803.11266

[15] Selmar K. Smit and A. E. Eiben. 2010. Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. In *Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I (Lecture Notes in Computer Science)*, Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna Esparcia-Alcázar, Chi Keong Goh, Juan Julián Merelo Guervós, Ferrante Neri, Mike Preuss, Julian Togelius, and Georgios N. Yannakakis (Eds.), Vol. 6024. Springer, 542–551. https://doi.org/10.1007/978-3-642-12239-2_56

[16] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[17] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316. https://doi.org/10.1016/j.neucom.2020.07.061