Project Report

Deep Artificial Neural Network Optimization

Konrad Piastka M20180038 | Dominika Leszko M20180077 | Joana Lorenz M20180412

Executive summary

This report aims to outline our project approach for optimizing the initial weights of a provided digit-recognition ANN. First, we give a brief overview over the project steps, next we go into more detail on each step - the implemented methods, our hypotheses and the conclusions. Finally, we present our best performing algorithm and give a summary of our project and the most relevant findings.

1. Project Overview

1.1 Problem data

Our task was to optimize the weights for a provided Artificial Neural Network (ANN) with two hidden layers and the neuron structure 64 | 10 | 10 | 10 with the task to recognize the digit on a hand-written picture. The input data is an array of 8 * 8 integer values between 1 and 16 representing the picture's pixels on a grey scale. The output is an integer between 0 and 9, representing the digit that the ANN assigns to a specific image. The weights should be optimized using a separate algorithm with the fitness function being the accuracy of the ANN's digit prediction.

When considering the structure of the underlying ANN in order to find the most promising trigger points for the weight optimization, we noticed that the baseline set-up of the weights (64 000 input weights) seemed odd. Based on literature and lecture material, a weight or node defines a single connection between two neurons in a Neural Network. Meaning that with the given ANN architecture of 64 + 10 + 10 + 10 neurons, we should be optimizing a total of 64 * 10 + 10 * 10 + 10 * 10 = **840 weights.** Thus, all methods used throughout the project and all insights mentioned in this report are based on the optimization of 840 instead of 64 000 weights.

1.2 Work Flow

To facilitate the project overview, we have organized our optimization efforts into four top-level categories as presented in *Figure 1*.

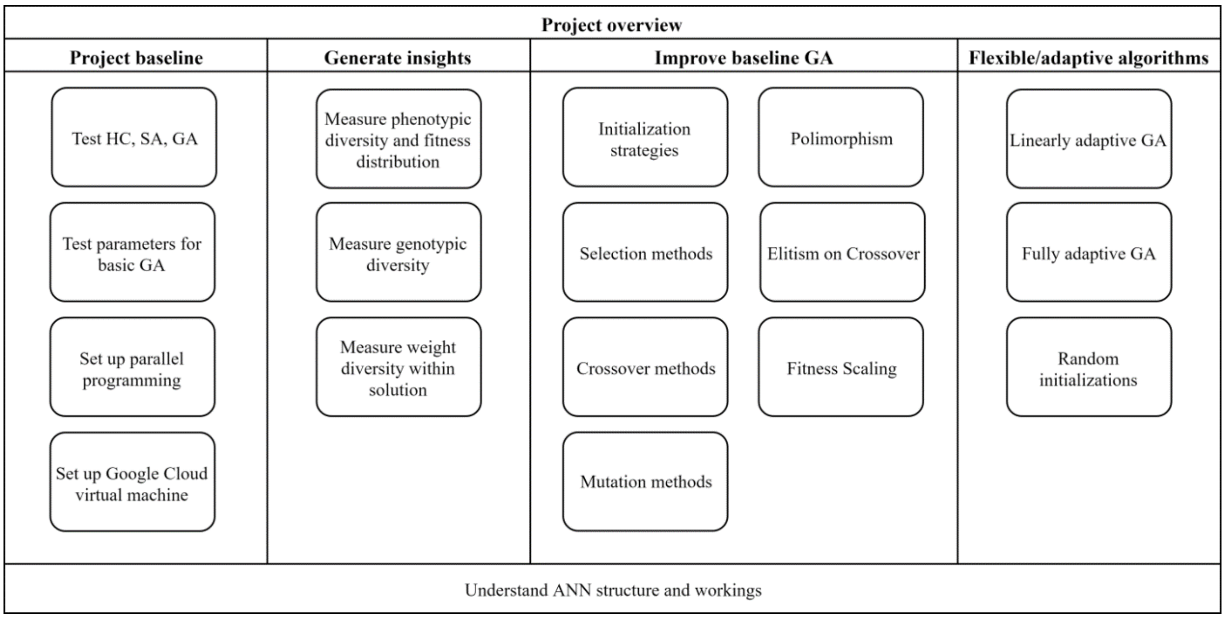
Project overview
Project baseline
Generate insights
Improve baseline GA
Flexible/adaptive algorithms
Test HC, SA, GA
Test parameters for basic GA
Set up parallel programming
Set up Google Cloud virtual machine
Measure phenotypic diversity and fitness distribution
Measure genotypic diversity
Measure weight diversity within solution
Initialization strategies
Selection methods
Crossover methods
Mutation methods
Polimorphism
Elitism on Crossover
Fitness Scaling
Linearly adaptive GA
Fully adaptive GA
Random initializations
Understand ANN structure and workings

Figure 1: Project Overview

First, we focused on the baseline, testing basic set-ups of Hill Climbing (HC), Simulated Annealing (SA), Genetic Algorithm (GA) on the optimization problem. Furthermore, we tested different parameters of the GA in order to find an appropriate range for all parameters on the problem. Also, since a lot of computational effort would be required throughout the project, we set up parallel programming using the Python library job lib and created a virtual machine on Google Cloud in order to run GA tests in the Cloud.

Since the project should be focused on Genetic Algorithm optimization, we then created measures to get additional insights into individual runs of our GA and understand how solutions were found, what they looked like and how the population developed over iterations. Specifically, we measured phenotypic diversity, genotypic diversity and the distribution of weights within individual solutions.

Based on the insights we were constantly gaining from these measures, we then improved the baseline GA through new or adapted initialization strategies, selection, crossover and mutation methods. Furthermore, we implemented multiple versions of polymorphism, elitist crossover as well as fitness scaling.

Last, following our observations from the implemented new methods, we took measures to build a more flexible genetic algorithm by implementing linearly adaptive GA's, a fully adaptive GA, and a GA with random initializations. Furthermore, throughout all project phases and implemented methods, we constantly tried to understand the internal workings of the ANN better in order to adapt our optimization strategy.

2. Baseline performance optimization

2.1 Comparison of baseline algorithms

All algorithms tested with the default settings outperformed Random Search, which is an expected outcome. **Genetic Algorithm** achieved the highest scores, but comparable results could be observed with Simulated Annealing. However, since the project focus should be set on Genetic Algorithms and its variations, we focused on the GA optimization during our project. Additionally, GA offers more flexibility in terms of parameters and methods than the other algorithms, thus offering more potential for further optimization.

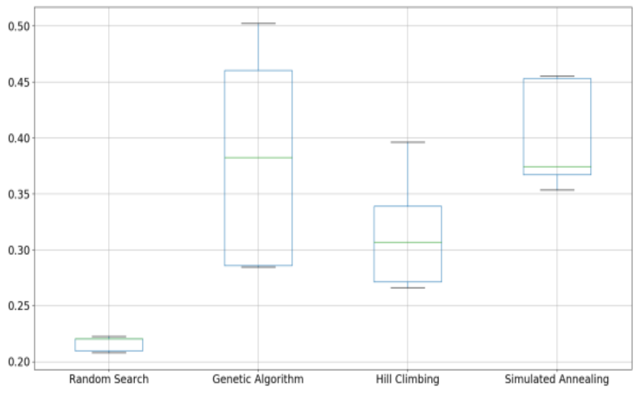

Figure 2: Comparison of baseline algorithms on 5 seeds

2.2 Parameter testing of Genetic Algorithm

Beside the selection, crossover and mutation methods, the basic genetic algorithm is defined by four parameters: population size (ps), probability of mutation (p_m) and probability of crossover (p_c). The optimal settings for these parameters will most likely be different for a given optimization problem. By testing different combinations of these parameters, we wanted to find good parameter ranges for the underlying problem so we could work with them during our further optimization steps.

Literature suggests that ps should be kept as high as possible, p_c rather high and p_m rather low. While the mutation mechanism will allow the GA to explore the search space, the crossover mechanism enables it to truly exploit the good solutions that are already present within the population. Exploration and exploitation need to be balanced in order to achieve good results. Given the project restrictions of 5000 maximum fitness evaluations, we had to compromise population size for the number of generations possible within a single GA.

During the parameter tests, we noticed that for the baseline GA, the algorithm was producing the best results when both p_m and p_c were set very high (> 0.85). We figured that this discrepancy to the theory could be due to the extremely large search space combined with a limited population size. In this case, mutation seems to be crucial in order to explore the immense search space and reduce the chance of getting trapped in local optima.

2. Generating insights

3.1 Genotypic diversity

A genotypic diversity measure was defined in order to control the diversity of weights within the population across iterations. Sustaining population genotypic diversity is a crucial aspect of Genetic Algorithm Optimization, as it helps to prevent the premature convergence of the algorithm to a local optimum by means of more extensive exploration of the search space. For our problem, we have used inertia to see how diverse the weights distribution in a given population is. Inertia is computed as the sum of squared distances of individuals' weights from a mean value of normalized weights. The measure is divided by the population size in order to make it comparable across algorithms of different population sizes. Genotypic diversity score also served as a orientation for the selection of larger radiuses for ball mutation.

3.2 Phenotypic diversity and fitness distribution

Phenotypic diversity per generation, meaning the diversity of the population in terms of fitness, is another important measure we chose to monitor for each GA run. We measured both the standard deviation of the fitnesses in the population, as well as the distribution of the fitnesses, using boxplots. The latter was especially helpful for a visual interpretation of the behaviour of fitnesses, since we could not only see the change of the absolute range of fitnesses, but also identify the amount of invalid solutions (fitness = 0). This gave us a hint of an overfitting model.

3.3 Within solution inertia

Based on the hypothesis that there could potentially be a weight-setting within a solution that is more favourable for the solutions fitness, we measured within-solution inertia. We also plotted the distribution of the weights after each run. However, weights of the final solution, independent of its accuracy, generally seemed to be normally distributed and we could not find a single weight pattern that was superior to others.

3. Genetic Algorithm Optimization

3.1 Summary of used methods

Multiple methods have been implemented with different aims. Generally, we tried to find the adequate balance of exploration and exploitation for the underlying problem by testing methods with different objectives and observing the impact on the fitness. If a method did not succeed, we did not immediately reject the underlying aim of the method but tried different methods that had a similar objective in order to verify the result and see whether the root cause was rather the method than the specific objective we were following.

Mostly, based on the insights created through our diversity measures and the observation of the developments when making adjustments to parameters or methods, we implemented methods that would:
A) Increase population diversity in order to more thoroughly explore the search space
B) More efficiently exploit existing good solutions or schemata to move towards an optimum
C) Adjust the GA behaviour to the underlying problem

The following table summarizes the implemented methods for each operator and indicates the objective.

Operators	Methods used	Objective
Initialization	Random Normal distribution of weights	A
	Random Uniform distribution of weights	–
Selection	Roulette wheel	–
	Tournament Selection	A
	Boltzmann Selection	A, B
	Rank Selection	A
	Steady State Selection	B
Crossover	Elite Crossover	B
	N-point Crossover	A
	Uniform Crossover	A, B
	Geometric Crossover	A
	Weights Crossover	A
	Node Crossover	C
	Cycle Crossover	B
Mutation	Random Point Mutation	A
	Ball Mutation	–
	Ball Mutation: Random point | Percentage	B
	Ball Mutation with various radiuses	A
	Creep Mutation	A
	Shuffle Mutation	A
	Inversion Mutation	A
	Swap Mutation	A

Mutation	Reverse Sign Mutation	A
	Node Mutation	C
Fitness Transformation	Linear Scaling	A
	Top Scaling	A
	Sigma Scaling	A
	Fitness Sharing	A
Flexibility	Polymorphism	A, B
	Linearly adaptive Genetic Algorithm	C
	Adaptive Genetic Algorithm	C
	Random Crossover	C

3.2 Initialization

One of our observations was that the final solution of a GA run is heavily dependent on the random state. In order to resolve this challenge, we were looking for a superior initialization strategy that would allow us to reach similarly high accuracies over multiple seeds.

As theory suggests, one approach for a more consistent initialization could be the use of a quasi-random number generator such as the Niederreiter, Halton or Hammersley generators (M. Rudnicki, S. Wiak (2002), p. 36; H. Maaranen, K. Miettinen, A. Penttinen (2007), p. 413-415). As opposed to the pseudo-random generators, such as sampling from a uniform distribution, these methods distribute the initialized points as evenly as possible over the search space. However, the performance of quasi-random generators degrades with the problem dimension and that these methods seem to show no more advantage in a search space with a larger dimension than 12 (M. Rudnicki, S. Wiak (2002), p. 36-24).

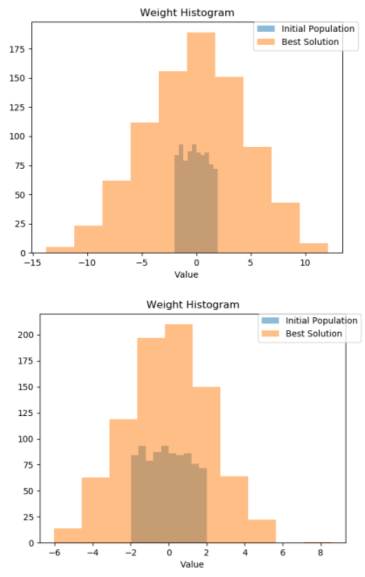

Figure 3: Initial and final weight distribution for a good (top) and bad (bottom) final solution

Another approach would be to observe the final weight structure of good solutions for a pattern that seems to be favourable towards fitness and initialize the first solutions with a similar pattern. However, since our weight distributions did not show any tendency, we could not identify a superior weight structure as seen on an example in *Figure 3*.

Since we noticed that in the final best solution of each run the mean of weights was oscillating around 0 and had a distribution similar to normal, we decided to replace the given initialization method (uniform distribution) with a distribution similar to normal. However, this approach did not have any significant impact on the fitness, so we kept to the initialization from uniform distribution.

4.3 Selection

Roulette Wheel – definition in the class notes.

Tournament Selection – This selection method seemed promising and so proved for our problem because through the selection pressure it allows controlling how strongly focused the algorithm should be on picking the best solutions vs. on exploring the search space more.

Boltzmann Selection (C. Lee, 2003) **–** This selection method leads to adaptive fitness where the fitness function does not stay fixed but varies with the environment. This way, the algorithm can both explore different solutions and exploit better configurations, which should theoretically help to solve the premature convergence problem. To implement the Entropy-Boltzmann selection, we need to appropriately estimate the entropy parameter. In general, the entropy of a system is not known a priori and can be really hard to estimate.

Rank-based Selection – one of the approaches to keep low selection pressure is the use of ranking selection. This approach diminishes the effect of large differences in fitness which leads to a higher probability of being selected for worse individuals. This selection method theoretically overcomes the scaling problem of proportional fitness selection and allows to keeps diversity in the population, since no individuals generate an excessive number of offspring.

Steady State Selection (S. Luke, 2015) represents an element added to the selection process through which we retain only a specific part of the population, by only considering the better-performing individuals in the selection process. By adjusting the percentage of the population that will be kept, the selection pressure can be modified. Having a very limited number of evaluations available per run keeping high selection pressure by deleting significant percentage of population in each iteration allows genetic algorithm to increase values of objective function very quickly. Unfortunately, when the landscape is as strongly rugged as in the case of this problem instance, getting stuck in the local optimum almost never cannot be avoided.

4.4 Crossover

Elitist Crossover (A.J. Umbarkar, P.D. Sheth, 2015) **–** in the basic implementation of the Genetic Algorithm, parents die after crossover and were fully replaced by their offspring. In order not to lose good solutions during crossover in this way, we implemented elitist crossover. Here, right after crossover we choose the two best solutions from the ‘family’ of parents and offspring as survivors. Elitist crossover can be combined with any crossover method. Below visualizations compare the results of a GA run with the exact same parameter settings with and without applied elitist crossover. Elitist crossover proved to drive higher fitness in a more consistent way. However, genotypic diversity drops quicker when elitist crossover is applied, which is an expected behaviour.

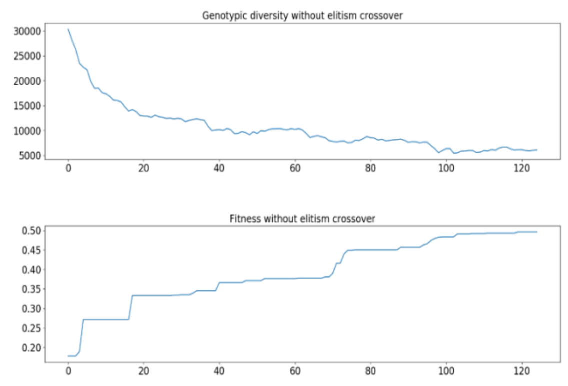

Figure 4: Genotypic diversity and fitness without elitism crossover

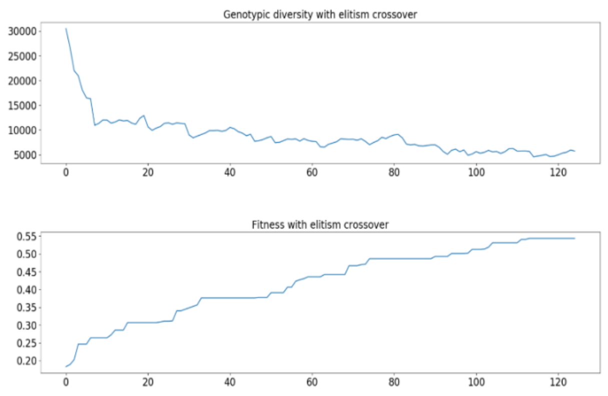

Figure 5: Genotypic diversity and fitness with elitism crossover

N-Point Crossover – being given the implementation
of one-point crossover, we decided to implement two-point and three-point crossover. Our rationale was that approach would be better able to create more mixed solutions and truly interchange the individual’s genes in order to produce more favourable schemata. Since two-point crossover produced much better solutions than one-point crossover, we implemented three-point crossover as well. However, in accordance with what theory suggests, the effect stagnated and three-point crossover failed to produce even better results.

Uniform Crossover (A.J. Umbarkar, P.D. Sheth, 2015) will perform crossover with a given probability at each gene to protect some potential building schemata and at the same time allows us to create diverse combinations, which should slow down the process of losing diversity. The performance of this method strongly depends on the chosen probability. The best performance was observed when p ~0.1, meaning that crossover happened on average 8 times in one iteration and the average length of kept sequences of each parent is about 10 weights.

Geometric Crossover (R. L. Haupt, S.E. Haupt, 2004) is a blending method that is commonly used when the optimization problem involves continuous values. This method is a remedy for point crossover methods’ problem: they do not introduce any new information but only shift values around different positions and as a result, different combinations of the same values are propagated to the next generations, which for a continuum of values merely interchanges two data points. Geometric Crossover combines variable values from both parents into new values for the offspring. We have implemented two versions of geometric crossover – one introduced by Radcliff (1991) and another by Michalewicz (1999). The former method combines the information from the two parents and introduces a new value that falls within the range bracketed by the two parents. The latter method is a variation of traditional geometric crossover and it allows a generation of offspring outside of values of the two parents variables.

Weight crossover is a method taken from a paper about training feedforward neural network using Genetic Algorithms (D.J Montana, L.Davis, 1989). The idea is to put a weight into each position of the child's chromosome by randomly selecting from one of the two parents at the same position in the chromosome.

Node Crossover (D.J Montana, L.Davis, 1989) **–** for each node (neuron) in the neural network child chromosome this method selects corresponding node from one of the parents. It then puts the weight of each ingoing link to the parent's node into the corresponding link of the child's network. The intuition here is that networks succeed because of the synergism between their various weights, and this synergism is greatest among weights from ingoing links to the same node.

Cycle Crossover operator identifies a so-called cycle between two parent chromosomes. For a continuous optimization problem, this method requires to first shuffle at least one of the parents and implement the crossover based on the sequence of shuffled indexes. Cycle crossover creates the opportunity to relocate genes that are part of important schemata (building blocks) and bring them closer together to decrease their chance of being divided. This way, favourable schemata could have a higher chance of surviving crossover.

Testing the implemented crossover methods on several seeds and in combination with different GA settings, usually the two-point and weights-crossover methods outperformed the other crossover methods, which can be observed exemplary in *Figure 6*, which compares the fitness that a GA with identical parameter settings and different crossover methods was able to achieve.

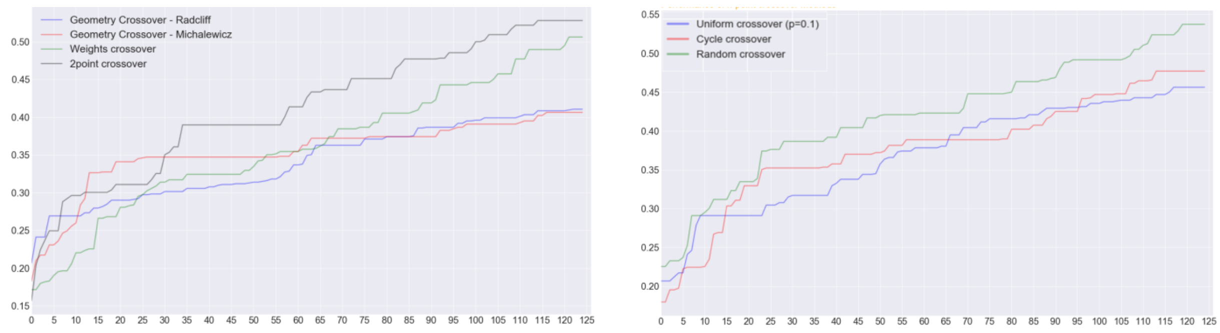

Figure 6: Best solution fitness development on one GA run for different crossover methods

4.5 Mutation

Ball Mutation – Besides the 'classic' ball mutation where the full set of weights within a solution is mutated within the given radius with probability p_m, we implemented alternative ball mutation methods. With these methods we meant to test the lowering of the mutation effect, i.e. keeping the mutated solution closer to the original solution. Both methods are based on classical mutation methods, where instead of a single solution (as is the case in our pre-implemented ball mutation), each gene in each individual is mutated with a given probability. However, each gene chosen for mutation is still only mutated within the given radius. This way, the solutions are still altered but not as strongly as they are with the pre-implemented ball mutation.

- *Percentage ball mutation* mutates each gene in the chromosome with a given probability *p*
- *Random point ball mutation* chooses *n* random points to be mutated

Ball Mutation with various radiuses mutates each gene in the chromosome with a given probability p_m. Additionally, the genes are mutated within the radius1 with a given probability p2 and within the radius2 (radius1*multiplier) with a given probability of 1-p2. The goal of applying different radiuses for ball mutation is to slow down the genotypic diversity loss. As it has been tested, the method proved to have an expected effect on the genotypic score – ball mutation with various radiuses significantly slows down the process of genotypic diversity loss.

Random Point Mutation | Creep Mutation – When having gone through ball mutation, an individual will still be located at a certain closeness, defined by the radius, to its original representation in the search space. We implemented random point mutation and creep mutation, where a number of positions within the individual are mutated randomly within the minimum and maximum weights of the solution. This way, there is much less pressure for the new weight at a certain position to stay close to the original individual. For random point mutation, a given number of positions to be switched is defined. For creep mutation, each position is changed with a given probability *p*.

Shuffle Mutation | Inversion Mutation | Swap Mutation (N. Soni and Dr. T. Kumar) – Shuffle mutation shuffles a random section of the solutions representation randomly, while Inversion Mutation inverts a random section of weights in its order. Swap mutation traditionally interchanges two random genes within the solution. To do our solution length justice, we instead swapped two random subsections of the solution of length between 0 and 100. With all three methods we hoped to break up schemata within the individuals in order to create more favourable strings of genes.

Reverse Sign Mutation reverses the sign of each position of the solution (e.g. -1 to 1 and vice versa) with determined probability p. This method could yield good results when used with geometric crossover, which generates new individuals. Reverse Sign Mutation would not make sense to be used with crossover methods that only shift individuals' positions around, not changing their actual values, as then there is a high risk of premature convergence.

Node Mutation (D.J Montana, L.Davis 1989) – selects n non-input nodes (neurons) of the network which the parent chromosome represents, than for each ingoing link to these n nodes, random value from distribution similar to normal ($\mu = 1$, $\sigma = 0.5$) is added, originally in the paper authors suggest to use the same distribution as in initialization phase but we observe improved performance when distribution close to normal was given.

Having tested different mutation methods, we concluded that ball mutation yielded significantly better results than other methods, which is visualized in *Figure 7* which shows the best solution fitness development of a GA run with the same parameter settings and different mutation methods.

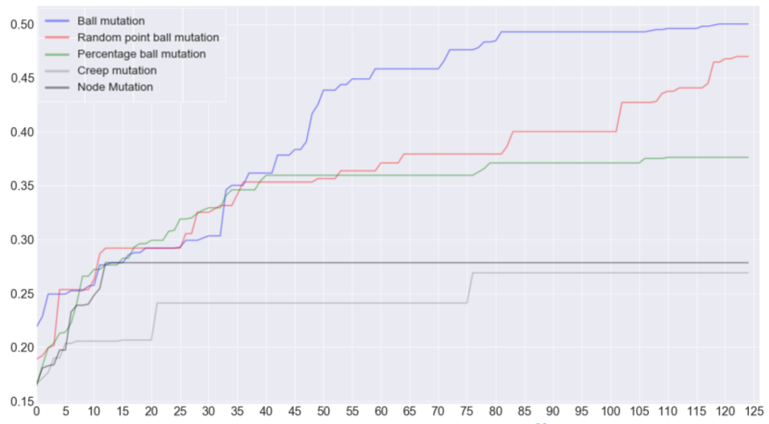

Figure 7: Fitness development over iterations using different mutation methods

4.6 Fitness Transformation

Fitness scaling (F. Sadjadi, 2004) is a step following the computation of individual fitness values by whatever fitness function necessary for a given processing task. During fitness scaling, the fitness values for the whole generation are fit to a predetermined pattern. After the initial population is generated, each individual must be reduced to its phenotype form (one number per chromosome). This form is then fed through a fitness function that assigns a fitness value to each individual based on the properties of its chromosomes' phenotype.

- **Top Scaling** (F. Sadjadi, 2004) sets the fitness of several of the top individuals to the same value (which is proportional to the population size), with all remaining individuals having their fitness values set to zero.
- **Linear Scaling** (F. Sadjadi, 2004) remaps the fitness values of each individual using a = max(raw fitness in population), b = -min(raw fitness in population)/population size.
- **Sigma Scaling** (A.A. Hopgood, A. Mierzejewska, 2009) attempts to moderate selection pressure over time so that it is not too strong in early generations and not too weak once the population has stabilised

and fitness differences are smaller. The standard deviation of the population fitness is used to scale the fitness scores so that selection pressure is relatively constant over the lifetime of the evolutionary program.

Fitness Sharing – Having noticed that genotypic diversity of Genetic Algorithm drops immediately after initialization, we decided to implement fitness sharing in order to, while selecting an individual, not only focus on its fitness, but also on its genotypic diversity. Following this reasoning, individual's raw fitness is divided by sharing coefficient, which results in decreasing fitness of non-diverse solutions, while increasing it for more diverse individuals. Additionally, parameter alpha has been created in order to be able to increase or decrease the effect of genotypic diversity on shared fitness value. (Raw fitness is taken to the power of alpha).

The results clearly indicate that scaling has a positive impact on the genotypic diversity and on the rate of convergence. Depending on the method of scaling, diversity can be increased or suppressed.

4.7 Adaptive/ flexible algorithms

After implementing multiple new and modified methods in our Genetic Algorithm, our fitness had improved but solutions still seemed to be converging too early, the random initialization seemed to have a large effect on the performance of the GA and GA setting with very good individual results were overfitting the algorithm in general.

In order to overcome the described challenges we were still facing with static methods and parameters, we decided to focus our efforts on the development of algorithms that were adaptive to the current state of the GA. The hypothesis was that throughout a single GA run, different parameters were needed in order to find a good solution.

Polymorphism – Multiple algorithms (sub-populations) are initialized parallelly and executed. Then, the main Genetic Algorithm is initialized based on the sub-populations. The hypothesis was that the separate evolution of populations could produce phenotypically diverse solutions that would be good for the main GA's initial diversity. We tested polymorphism with different versions of initialization of the main population: crossover between random members of the sub-populations, mutation of the best individual, mutation of the best individuals and initialization in the neighbourhood of the best individual.

However, while the sub-populations were effectively reaching quite good solutions within a short time, the main GA was not able to further improve on the sub-population outcome. Thus, results stagnated quickly.

Linearly adaptive Genetic Algorithm – For an initial test of our hypothesis that parameters should be adaptive throughout the Genetic Algorithm, we chose to adapt the selection pressure in Tournament Selection linearly over the lifetime of the GA. The hypothesis was that selection pressure should initially be high in order to explore the large search space. At some point, however, pressure might need to increase in order to finally focus on the most promising areas and find the highest possible solution for a given GA run.

Generally, individual seeds scored the highest fitness values we had observed until then. However, when trying to optimize the different parameters for this GA method, we noticed that for every tested seed a different parameter combination worked the best. Looking at the distribution of fitnesses, we also noticed that a large amount of solutions in every generation were invalid. Overall, the method seemed to generally be able to reach good solutions, but overfitting needed to be reduced in order to create a more reliable performance.

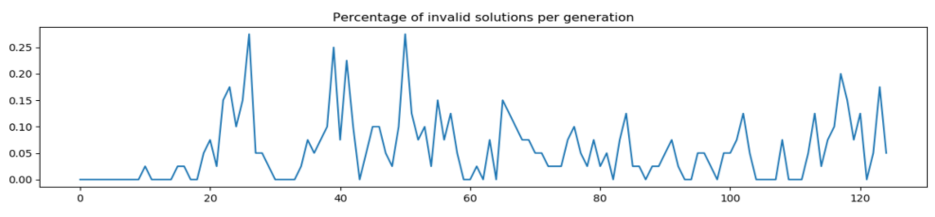

Figure 8: Percentage of invalid solutions on a run of GA using adaptive tournament selection

Random Crossover – in order to introduce more randomness and increase diversity, this method implies different crossover methods with 25% of probability each (one-point crossover, two-point crossover, three-point crossover, uniform crossover).

Tournament Selection with Fitness Sharing based on diversity threshold
This method was implemented in order to use shared fitness when selecting individuals by means of tournament selection. This method takes advantage of fitness sharing, i.e . favours more diverse solutions over the less diverse ones, boosting their shared fitness and hence increasing their probability of being selected. Selection is performed on shared fitness only once the genotypic score (inertia) falls below the pre-defined threshold; otherwise the raw fitness is used for selection.

Adaptive Genetic Algorithm (B. Mc Ginley, F. Morgan, C. O'Riordan, 2008) Genetic algorithm through its run is focusing on two goals, to explore and exploit landscape of objective function. Although both goals are important to come as close as possible to a global optimum, in different stages of the GA run, one is more important than the other. In early iterations it is important to explore the search space, while in the final iterations (especially when the number of iterations is limited) the main focus is on the exploitation of good solutions. This can be achieved, by adjusting all parameters of the GA over the run of genetic algorithm. Adaptation of parameter are based on diversity in population.

4.8 Results

After having tested different selection, crossover and mutation methods with various parameter settings and GA tuning methods described above, we observed the best performance of **~54%** average accuracy over 5 seeds for the GA defined as follows:

- Mutation: Ball Mutation with radius=0.5
- Crossover: Weights Crossover with elitism
- Population size: 40
- Adaptive probability of crossover p_c (from 1 to 0.9) and mutation p_m (from 1) based on genotypic diversity of the current population
- Selection: Tournament Selection with Fitness Sharing
- Selection pressure increasing from 0.15 to 0.3, based on the genotypic diversity of the population and an added factor that increases exponentially over the lifecycle of the GA

We chose this GA not only based on the average fitness on multiple seeds, but also on its ability to appropriately preserve populations diversity and maintain a constant growth of fitness over iterations. Given these facts, it seemed the most promising of the tested algorithms.

4.9 Summary

Overall, the adjustments that made the largest difference for us were:

- Elite crossover
- Using the appropriate amount of weights for the ANN
- Fitness sharing

Otherwise, a lot of methods that were mentioned in literature or seemed rational to us proved inefficient for the underlying optimization problem. Due to time restrictions and limited computational power, we were not able to test each possible combination of the implemented methods extensively and optimize the parameters on all methods. Thus, there might still be a better set-up of the implemented methods.

With our final Genetic Algorithm, we seem to be able to keep diversity throughout the iterations and can avoid early convergence, as can be concluded from the slowly decreasing genotypic diversity and the constantly improving fitness of the best solution per iteration. However, in order to score a higher fitness given the restrictions on number of iterations, the slope of the fitness curve would need to be steeper.

We believe that finding a method for a more directed exploration could improve the Genetic Algorithm. Also, a better understanding of the fitness distribution within the search space would facilitate to adjust parameters for a more targeted exploration as well as choose a more adequate initialization strategy for the Genetic Algorithm.

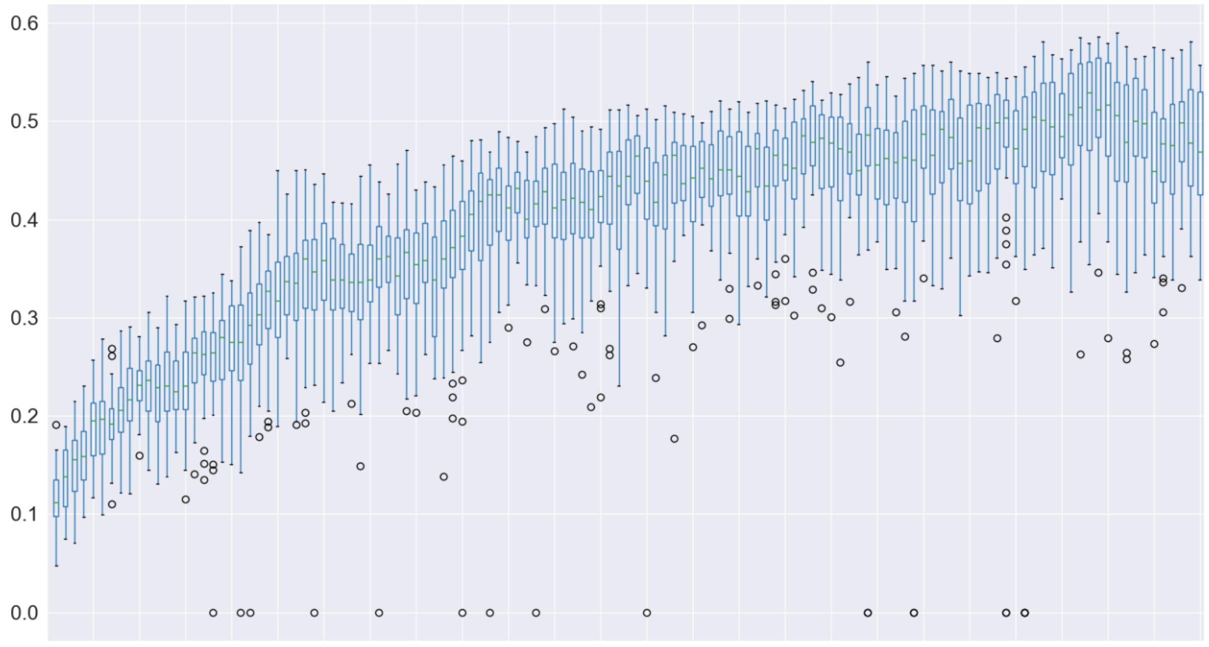

Figure 8: Fitness distribution of best performing GA over iterations

6. Bibliography

I. Lee, Chang-Young, "Entropy-Boltzmann selection in the genetic algorithms", *IEEE Transactions on Systems*, Man and Cybernetics, Part B: Cybernetics 33.01 (2003) : 138-149. IEEE Press Piscataway, NJ, USA.

II. Umbarkar, J and Sheth, P.D., "Crossover Operators in Genetic Algorithms: A Review", *Ictact Journal on Soft Computing 6.1* (2015).

III. Sareni, Bruno and Krahenb, Laurent, "Fitness Sharing and Niching Methods Revisited", *IEEE Transactions on Systems*, Man and Cybernetics, Part B: Cybernetics 2.3 (1998)

IV. Mitchell, Melanie, "An Introduction to Genetic Algorithms" (1996).

V. Koza, John, "Genetic Programming" (1992).

VI. Goldberg, David E., "Genetic Algorithms in Search, Optimization and Machine Learning" (1989).

VII. Pater, Lukasz, "Application of artificial neural networks and genetic algorithms for crude fractional distillation process modelling".

VIII. Farzad, Sadjadi, "Comparison of fitness scaling functions in genetic algorithms with applications to optical processing", Proceedings of SPIE – The International Society of Optical Engineering (2004).

IX. Mc Ginley, Brian and Morgan, Fearghal and O'Riordan, Colm, "Maintaining Diversity through Adaptive Selection, Crossover and Mutation" (2008).

X. Rudnicki, Marek and Wiak, Slawomir, "Optimization and Inverse Problems in Electromagnetism" (2002).

XI. Maaranen, Heikki and Miettinen, Kaisa and Penttinen Antti, "On initial populations of a genetic algorithm for continuous optimization problems", Journal of Global Optimization (2007).

XII. Soni, Nitascha and Dr. Kumar, Tapas, “Study of Various Mutation Operators in Genetic Algorithms” International Journal of Computer Science and Information Technologies 5.3 (2014).

XIII. Morisson, Ronald W. and De Jong, Kenneth, “Measurement of Population Diversity”, Artificial Evolution, 5th International Conference (2001).

XIV. Alabsi, Firas, Naoum, Reydah, “Comparison of Selection Methods and Crossover Operations using Steady State Genetic Based Intrusion Detection System”, Journal of Emerging Trends in Computing and Information Sciences, Vol. 3 No. 7, (2012).

XV. Hopgood, A.A., Mierzejewska, A., “Transform Ranking: A New Method of Fitness Scaling in Genetic Algorithms” (2009).

XVI. Hien, Nguyen Thi and Hoai, Nguyen Xuan, “A Brief Overview of Population Diversity Measures in Genetic Programming” (2006).

XVII. Ochoa, Gabriela and Harvey, Inman and Buxton, Hilary, “Optimal Mutation Rates and Selection Pressure in Genetic Algorithms” (2000)

XVIII. Jebari, Khalid, “Selection Methods for Genetic Algorithms” (2013)

XIX. Luke, Sean, “Essentials of Metaheuristics”, Second Edition, Online Version 2.2 (2015)