Evolutionary Algorithm Projects: The Bias-Variance Trade-off

M Medioli
Delft University of Technology
M.Medioli@student.tudelft.nl

J Feijen
Delft University of Technology
J.Feijen@student.tudelft.nl

M de Groot
Delft University of Technology
-

D Ha
Delft University of Technology
D.M.F.Ha@student.tudelft.nl

1 INTRODUCTION

In the last decade, data analysis has acquired a key role in all the processes that have as their ultimate goal the search for structures and patterns intrinsic to the data in order to exploit them in every field of study. This led to an exponential increase in machine learning techniques, first of all, supervised learning. Thanks to new technologies, collecting data has become faster and more accessible, leading to the dissemination of large datasets containing inputs with associated ground-truth observations on which supervised learning bases. Having access to these observations, supervised learning tries to generate a parametric model that best reflects the behaviour of accessible training data and at the same time, best generalizes the model parameters for unseen data.However, it is not possible to satisfy these two aims at the same time and it is necessary to find a trade-off between the variance and the bias of the model. High-variance learning methods may be able to represent their training set well but are at risk of overfitting to noisy or unrepresentative training data. In contrast, algorithms with high bias typically produce simpler models that don't tend to overfit but may underfit their training data, failing to capture important regularities. The aim of the proposed work is to find this optimal trade-off through the analysis and use of *multi-objectives evolutionary algorithms* (MOEAs) to train a **regression model** in a way that is aimed at optimizing two conflicting objectives: goodness of fit and smoothness.

1.1 Radial Basis Function Regression

The aim of our real-value regression problem is to find a function $f(x) = y$ that matches the data as accurately as possible. **Radial basis function regression** is used to describe approximate the true function using a linear combination of K Gaussians, also referred to as bases. The linear combination is described in equation]1:

$$r(x \mid \theta) = \sum_{k=1}^{K} w_k \cdot \exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right) \tag{1}$$

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

where the regression model parameters $\theta = \{w_k, \mu_k, \sigma_k\}$ consist in the parameters (weights, means and standard deviations) of the K Gaussians. In order to find the best parameters θ, it is necessary to minimize a loss function that reflects the goodness of a model in fitting the training data. The most commonly used loss function for regression problems is the **Mean Squared Error**:

$$f_0 = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i)|^2 \tag{2}$$

where y_i correspond to the predicted real value by the regression function $r(x \mid \theta)$ and $\hat{y}_i$ is the relative function value for input x_i in the training set. The more complex the model that determines y_i is, the more it presents a high variance and consequently a high number of parameters. To reduce the impact of complexity and avoid overfitting, it is possible to introduce a **regularization term** (Tikhonov regularization, also know as ridge regression) in the function to be minimized:

$$f_1 = \frac{1}{K}\sum_{i=1}^{K} |w_k|^2 \tag{3}$$

The common approach to combining the fitness function and the regularization term is to add a penalty term λ to weigh the importance of regularization, i.e. smoothing. Substituting 1 in 2, a single objective function to be minimized is obtained:

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - r\,(x_i \mid \theta)|^2 - \lambda\frac{1}{K}\sum_{i=1}^{K} |w_k|^2 \tag{4}$$

1.2 Multi-objectives optimization

The term penalty λ is user-defined and obtaining it optimally requires an expensive computational tuning phase and a series of time-consuming tests. To avoid this, it is possible to conceptualize the problem of single-objective function minimization as a multi-objective optimization. We analyze and perform 4 different evolutionary algorithm for multi-objectives optimization in order to minimize both objective in 2 and 4. The result of this optimization consists in a set of optimal solutions that constitutes the Pareto set. Each solution in the Pareto set is composed of $3K$ real-values corresponding to the optimal parameters for Gaussians bases. In the following sections the 4 algorithms are described and analyzed in black-box setting, the comparison method between them is defined and finally a whitebox approach is proposed for each algorithm to improve its performance relative to the specific radial basis function regression problem. The algorithms analyzed are the following:

- Multi Objective Real-Valued Gene-PoolOptimal Mixing Evolutionary Algorithm (**MO-RV-GOMEA**)[**?**]

- Non dominated Sorting Genetic Algorithm (**NSGA2**)[**?**]
- Uncrowded Hypervolume-based Gene-pool Optimal Mixing Algorithm (**UHV-GOMEA**)[**?**]
- Multi Objective Adapted Maximum-Likelihood Gaussian Model (**MAMaLGaM**)[**?**]

2 BASIS OF COMPARISONS

In order to compare the to be investigated algorithms, several parameters are kept constant. The problem size (K) is set to 10. Early experiments showed that $K = 10$ is relatively easy to solve compared to $K = 20$ and not too easy compared to $K = 3$. This gives the different algorithms room to show their performance.

The hypervolume measure is the method used to evaluate the quality of a set of solutions. Other methods such as generation distance have been looked into, but the main problem is that there's no exact way to determine the optimal solution set. The drawbacks of hypervolume is that it does not indicate the quality of the entire solution set including dominated solutions. Since we're only interested in the front this does not matter.

The reference point is set to $(400, 400)$. Early on black-box experiments never crossed 200 for either objective value. The reference point was thus set to twice the maximum observed value. The maximum value of each objective function could have been used, but this raised issues with output files of the original algorithms that could only represent the hypervolume with a few decimals.

The measure of resources used is in function evaluations. All writers of this report have different computers so time is definitely not a fair method to compare the resource consumption. Doing all computation on a single computer was also not possible since four different algorithms had to be experimented on with only limited time. The drawback of function evaluations is that overhead is not fairly assessed.

The maximum function evaluations is set to 200.000. The main reason for choosing this value is that every algorithm could finish in decent time.

If an elitist archive is used, the archive is set to a size of 500. The authors of the algorithms usually use archives of 10.000. This value took too long for most algorithms to finish in decent time. 500 was experimentally determined and agreed upon by all parties.

In order to speed up the experiments of the black-box analysis, all algorithms initialized their parameters according to $w_i \in [-100, 100]$, $\mu \in [-100, 100]$ and $\sigma \in [0.1, 100]$. The ranges should not interfere too much as the data lies within $[-3, 2]$ and most parameters converge towards values smaller than 10. σ does not initialize in the same range as the other variables as small values of σ will produce overflow. Negative σ's could have been allowed, but in most cases would require rewriting large parts of the source code. If the boundary repair option is available it should be turned on in case an overflow occurs.

The time limit of the experiments is set to 10 minutes, but this value is never reached.

All algorithm independent parameters were fixed in order to obtain a fair comparison. The stopping criterion is based on function evaluations so that we can measure how well an algorithm uses the function evaluations. However, all algorithms in question have more parameters than those described above. Setting those remaining parameters to the recommended or default values would not be fair because one researcher might have put more effort in the parameter search than the other. Moreover, the default hyperparameter values for one algorithm might work better on the problem at hand than the other. Thus to be able to make a fair comparison, a universal parameter search method was devised. It is described in the next section for the population size.

2.1 Hyperparameter search

An important parameter to optimize for any algorithm is the population size. A large population will take more evaluations to compute per generation. A small population size influences diversity and thus the optimum that can be found.

Each algorithm uses the same population search method. An algorithm starts with a minimum population size and increases the population size exponentially (base 2) until a problem is reliably solved. The algorithm resets the population size to half of what it found and retries the experiment at least 10 times. Each experiment is deemed successful if the algorithm is able to find 90% of the best hypervolume found for that problem size 10 consecutive times. The best hypervolume for a given problem size is found by letting all the algorithms find the best hypervolume using significantly more resources and to pick the best of those hypervolume found. The population size is slowly increased with factors of 1.1 until the population size reaches the population when it first found a successfull run. This last found value will be the one that is used.

Besides population size, all other algorithm specific parameters were optimized individually using the same method. This might not find the optimal combination of hyperparameters, but it is a fair attempt to find a well enough combination and most importantly, as all algorithms are tuned in the same manner, the results can be compared fairly.

3 INDIVIDUAL CONTRIBUTIONS

3.1 Matteo Medioli

The following section analyzes the multi-objective version of the Estimation-Distribution algorithm *AMaLGaM*, in particular the version based on the incremental estimation of the covariance matrix *iAMaLGaM* described in [**?**] which allows a significant reduction in the population size. The multi-objectives incremental version is referred to as *iMAMaLGaM* (hereafter simply called MAMaLGaM).

3.1.1 Key Features

. **Overview**

The *Adapted Maximum-Likelihood Gaussian Model (AMaLGaM)* is a Gaussian EDA that uses a Gaussian distribution related to the current generation to generate a new population of individuals. It is possible to significantly reduce the required population size based on the assumption that subsequent generations have much in common. With an incremental learning approach, it is possible to combine the probability distribution that is estimated from the selected solutions in the current generation t with the distribution that was used in the previous generation $t-1$. Obviously, it is necessary to keep track in memory of the parameters of generation $t-1$. The Gaussian Σ describes the dependencies between the random variables, that is the real-values that make up the individuals of the population. The ML-based updating rules in generation t are the following:

$$\begin{aligned}\hat{\boldsymbol{\mu}}(t) =& \frac{1}{|S|}\sum_{i=0}^{|S|-1} \boldsymbol{S}_i \\ \hat{\Sigma}(t) =& \left(1-\eta^{\Sigma}\right)\hat{\Sigma}(t-1)+ \\ & \eta^{\Sigma}\frac{1}{|S|}\sum_{i=0}^{|S|-1}\left(\boldsymbol{\mathcal{S}}_i-\hat{\boldsymbol{\mu}}(t)\right)\left(\boldsymbol{\mathcal{S}}_i-\hat{\boldsymbol{\mu}}(t)\right)^T\end{aligned} \tag{5}$$

where $\boldsymbol{S}$ denote a vector of data (individuals) and η^{Σ} is the covariance learning rate, determined empirically [**?**]. Taking into account the old dependencies, the new "weak" dependencies within the current generation composed of few individuals are influenced on the old ones, forwarding dependencies during the iterative process. At each iteration, ML estimation increases the strength of the dependence between the random variables. Given these premises, Given these premises, the required population size is expected to be smaller.

Memory-decay and Anticipated Mean Shift

To prevent premature convergence, *AMaLGaM* uses a memory-decay multiplier $\boldsymbol{c}$ in order to reduce the variance of the individuals if the taken direction improves the optimization. Additionally, Anticipated Mean Shift (AMS) allows a look-ahead based on the difference between the means of two subsequent generations, defining a possible correct direction on which to move some of the generated solutions:

$$\hat{\boldsymbol{\mu}}^{\text{Shift}}(t) = \left(1-\eta^{\text{Shift}}\right)\hat{\boldsymbol{\mu}}^{\text{Shift}}(t-1) + \eta^{\text{Shift}}(\hat{\boldsymbol{\mu}}(t)-\hat{\boldsymbol{\mu}}(t-1)) \tag{6}$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + 2\hat{\boldsymbol{\mu}}^{\text{Shift}}(t) \tag{7}$$

MAMaLGaM and cluster registration

To minimize the two objective functions related to the bias-variance trade off problem, it is possible to use Amalgam in its multi-objective version, Mamalgam. Each objective function is minimized through the application of the single-objective algorithm on both functions, thus generating an optimal set of non-dominated solutions, called Pareto-optimal. The Pareto set $\mathcal{P}_S$ is the approximation of the set of all Pareto-optimal solutions to the real-values problem and the Pareto front $\mathcal{P}_F$ is the set off all m-dimensional objectives values corresponding to the solutions in the Pareto set. The algorithm maintains optimal solutions through the use of an elitist archive, in order to have a monotonous convergence during optimization. Looking for improvements on the whole Pareto front can however lead to an improvement in some regions of the Pareto front (*front improvement*) but at the same time worsen other regions (*front degradation*).

The most common technique to avoid worsening of the Pareto front during the optimization is the clustering of the front to locally optimize the solutions in the same region. In the specific case of the estimate-distribution algorithm, it is necessary to associate each cluster of generation t-1 with the cluster of generation t so as not to lose the internal dependencies of the individuals of the sub-population in each cluster. This technique, also called mixture-based EDA, is amply described in [**?**] and uses an algorithm that calculates the distance between clusters of subsequent generations. By applying the ML approach of AMaLGAM within the clusters of the front, it is possible to exploit the potential of AMALGAM in its multi-objective version, MAMaLGaM, to solve the bias-variance dilemma.

3.1.2 Analysis black-box optimization.

An analysis of the MAMaLGaM algorithm is performed below with the aim of verifying expectations and defining the optimal hyperparameter set.

Approximation set size analysis

The starting point of the black-box analysis of the algorithm is the size of the approximation set. Since it was fixed a priori among the members of the group, it was empirically verified that the $approximation\_size = 500$ did not negatively influence the optimization algorithm. The analysis was carried out on

$$approximation\_size \in \{10, 250, 500\}$$

with different population sizes

$$n \in \{64, 128, 256, 512, 1024, 2048\}$$

and different number of bases K, consequently different problem dimensionalities since $|\theta| = K * 3$:

$$K \in \{2, 4, 8, 10, 16\}$$

After performing 10 runs for each population size and problem dimensionalities, it was possible to obtain how too small an approximation size reduced the average hypervolume value (except for $n = 2048$, always reporting the same low values for each approximation size due to the low number of maximum evaluations available). These first results are reported in Figure 24 for $approximation\_size = 10$ and Figure 25 for $approximation\_size =$

500. It is possible to notice lower average hypervolume values for $approximation\_size = 10$. The relative plot with $approximation\_size = 250$ has been omitted without showing particular differences with $approximation\_size = 500$. To maintain consistency for comparison with the other algorithms, the size of 500 was kept.

Population size analysis
As previously described, the expectation is that of not having a particularly large population size thanks to the incremental approach during the estimate of the covariance. First, the research of the minimum population required was carried out. Using the common approach chosen, a starting population was defined n = 16. Through the analysis it was possible to obtain the optimal minimal population $n\_opt = 35$ with *success rate = 100%*. The results of the analysis are shown in Figure 23. Having to compare the different algorithms, the values for K (number of bases), elitist archive size and approximation set defined previously in agreement with the other members of the group were chosen. The number of clusters, or mixture components, for this initial analysis has been kept as the default of the framework. The hyper-parameters for the analysis of populations relative to the graph in 23 are therefore $K = 10$, $elitist\_size = 500$, $approximation\_size = 500$ $max\_evaluations = 200000$ and $n\_clusters = 5$ (default).

After finding an optimal population size $n\_opt = 35$, the convergence performances were compared with the dimensionalities of the problem. In the graphs in Figures 26, 27, 28, 29, 30 the average hypervolume value and the standard deviation are reported as evaluations increase until reach the maximum (200000). Each chart compares all population sizes considered in the previous analysis against a fixed value of K.
It is interesting to note that as the size of the problem increases, the performance of MAMALGAM with the optimal population size $n\_opt = 35$ remains constant, as opposed to larger populations which, as expected, have worse performances due to the set evaluation limit. These graphs also confirm that the choice of n_opt is correct and that a small population has excellent performances thanks to the incremental approach of MAMaLGaM.

Number of mixture components analysis
The last analysis is related to the number of clusters on which the algorithm optimizes locally the solutions in the Pareto front. Taking into account the final comparison goal, 10 runs were carried out on the optimal population by varying the number of clusters, i.e. mixing components that generate the distributions from which to sample during the variation. Several numbers of $n\_cluster$ were also considered on which a convergence analysis was carried out:

$$n\_cluster \in \{3, 5, 6, 8, 10\}$$

In Figure 31 the results of the convergence analysis are reported. It can be seen that the variation on the number of clusters does not particularly affect the convergence of hypervolume. At the same time it must be considered that the number of clusters is closely related to the size of the population. With such a small population size, convergence is not highly affected by the different numbers of mixing component taken into account but using too many would result in a loss of dependencies between the solutions of each generation.

3.1.3 Proposed improvements. To improve the performance of MAmalgam with respect to the bias-variance trade off problem, two methods based on the addition of noise are proposed below. Adding noise that leads to an improvement on an objective function generates a mutation on individuals within the clusters. The improvement generated by the addition of noise is kept in memory by estimating the maximum likelihood.

Worst Basis Identification and Score Function
The assumption that led to the development of the two WBO proposals is based on the research of the 3 worst parameters internal to a single individual. Having to minimize the MSE and at the same time regularize the number of parameters of the model, we tried to search among the Gaussian K that make up an individual the worst. Let us denote the set of $K * 3$ random variables contained in a solution such as $\mathbf{x_{3K}}$. The WBO iteratively eliminates a block of 3 parameters representing an Gaussian internal to an individual by generating a new temporary solution $\mathbf{x_{3K-1}}$. For each individual, a set of K temporary variables is generated $\{\mathbf{x_{3K-1i}}\}^K$. For each temporary variable, an evaluation of the fitness function $f0$ is performed, therefore for each individual, K 1-dimensional objective values are computed. These values are denoted by $\{f0(\mathbf{x_{3K-1}})_i\}^K$. Then, the differences between the fitness function over original solution with all $K * 3$ gaussian parameters $\mathbf{x_{3K}}$ and all the K temporary solutions are computed:

$$\forall i \; D_i = f0(\mathbf{x_{3K}}) - f0(\mathbf{x_{3Ki}}) \tag{8}$$

Now, two scenarios can arise:

- $D_i > 0$: means that removing the $basis_i$, $f0(\mathbf{x_{3K-1i}}) > f0(\mathbf{x_{3K-1}})$ and consequently, the regression model improves of a factor D_i.
- $D_i \leq 0$: means that removing the $basis_i$, $f0(\mathbf{x_{3K-1i}}) <= f0(\mathbf{x_{3K-1}})$ and consequently, the regression model is worst without the selected basis.

The algorithm selects the basis index which, if removed, improves the model of the factor D_i greater. The algorithm chooses the basis for which the difference in the calculated fitness is maximum. The index of the worst basis is denoted by i_{worst} and its relative parameters will be $w_{i_{worst}}$, $\mu_{i_{worst}}$ and $\sigma_{i_{worst}}$. The WBO add a noise keeping the dependecies generating the new real-values from a Normal distribution and applying the Cholesky matrix retrieved from covariance matrix Σ. The added noise is a function of the factor D_i: if eliminating a base, D_i is large, it means that that base has a high negative impact on fitness and consequently it is necessary to apply a high variance in the new dependent value. Instead, slight noise will be applied to the parameter to be updated.The two methods of adding noise are analyzed below. The first generates correlated noise starting from the current value of the parameters $w_{i_{worst}}$, $\mu_{i_{worst}}$ and $\sigma_{i_{worst}}$, the second generates noise in the normal Gaussian which is multiplied by the Cholesky matrix.

Noising parameters with weighted Normal distribution
The first nosing approach is the following:

(1) compute a score function to introduce the previous real-value of a parameter p_i. This function is defined as follow:

$$f(p_i, D_i) = \frac{\sqrt{|p_i|} * D_i}{varm} \tag{9}$$

where $varm$ its a constant variation multiplier.

(2) Update the old parameters p_i as follow:

$$p_i += \mathcal{N}(0, 1) * Cholesky * f(p_i, D_i) \tag{10}$$

3.1.4 Analysis white-box optimization. The results of the WBO are shown in the Figure 32 and 33, it is possible to note that if the noise takes the correct direction, the improvement on the front is significant. On the other hand, noise is not always added in order to apply optimum pressure on the front wall. Thanks to the local ratings for each K basis, the algorithm stops long before the BBO. If the noising does not generate improvements, WBO related or similar fronts can be found in less time (times improve by a factor of K). Thanks to the cholesky matrix, WBO parameters are generated that carry on the dependencies learned during the estimation of covariance through ML.

3.2 Damy Ha

In this section the `Multi Objective Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm` (`MO-RV-GOMEA`) is investigated as described in [**?**].

3.2.1 Key features `MO-RV-GOMEA`. The key features of `MO-RV-GOMEA` are briefly outlined. `MO-RV-GOMEA` maintains a population P of `n` solutions. Non dominated solutions are stored in an elitist archive A. The population is updated iteratively. `MO-RV-GOMEA` does truncation selection on the best non-dominated solutions of the population. To find a good spread of solutions, the selected solutions S are then clustered into a predefined amount of clusters C.

`MO-RV-GOMEA` makes use of the `Family of Subset` (`FOS`) linkage model to describe linkage structures. This linkage model F is learned per cluster C_k. Parameters which are used for variation are deducted for each linkage set in each cluster.

`MO-RV-GOMEA` does variation on a solution by iteratively selecting a linkage set in a random order and changing parts of the solution defined by the selected linkage set. The parameters are changed by sampling from a multivariate Gaussian distribution according to the previously determined parameters. Changes are accepted if the new solution dominates the older solution or when it's not dominated by the solutions in the elitist archive. Otherwise the solution is returned to it's previous state. The process continues until all linkage sets have been examined.

`MO-RV-GOMEA` further has `Adapting Distribution multipliers`, `Anticipated Mean Shift` (`AMS`) and `Forced improvements` (`FI`). `Adapting Distribution multipliers` scale the parameters of the Gaussian distribution if better solutions are found relatively far away from the mean. `AMS` takes a few offspring per cluster and based on data from previous generation tries to project these offspring into the region where improvement is expected. `FI` forces a solution to move towards an elitist solution if it hasn't improved for several generations.

3.2.2 Analysis black-box optimization. One of the interesting features of (`MO-RV-`) `GOMEA` is that it allows the modeling of linkage between parameters. The way the radial function is modeled into `MO-RV-GOMEA` is that every consecutive tuple of 3 parameters represent w_j, μ_j, σ_j of $r(x|\theta_j)$. This makes the parameters of such tuple correlated. These tuples also correlated between them selves as they are summed to represent the radial function.

Figure 1 shows the hypervolume and rate of convergence vs evaluations to the best found hypervolume when all parameters are considered to be related (full), a linkage tree is used (LT), independent of each other (F1) and when consecutive tuples of 3 are formed (F3) for a problem size of $K = 10$, $|P| = 150$ and $A = 500$. All other parameters were set to the recommended value. The latter `FOS` model is not fair to consider when black box analysis is done, but does reveal some insights. The population is much smaller than the recommended population otherwise the algorithm has a limited number of generations.

Figure 1 shows that none of the models are able to reach the best found hypervolume. This hypervolume was found using significantly more resources in a whitebox scenario. Even when significantly more resources were allotted to the black-box scenario it never reaches the best found hypervolume.

What might be confusing of the full model is that it appears to be significantly better when few evaluations (say less than 20.000) have passed. The fundamental difference between the full model and the other models is that the full model does fewer evaluations per generation and thus outputs data relatively early. The other models are using a FOS to do variation which does more evaluations per generation.

The LT model has the largest FOS set and appears not to be done yet when the maximum evaluations was reached. Since we've constrained the evaluations to 200.00 it's not viable to use the LT for larger problems. F3 has roughly the same average performance as the full model, but also has a larger standard deviation which means it's able to find better (or worse) solutions. It's clear that a fitting model is important to find better solutions but also runs into risk of not converging before the maximum evaluations is reached.

Figure 2 shows the front of obtained by a full model when $K = 10$, $|P| = 150$, $A = 500$ and all other parameters are set to the recommended value. The gaps of f_0 shows that MO-RV-GOMEA has difficulty finding radial function with small training error. This is due to the fact that f_0 is much harder to optimize compared to f_1.

Since the allocated evaluations is rather limited it's important to tune the parameters such that the algorithm is able to solve the problem reliably with as few as possible evaluations. An important parameter that influences reliability and function evaluations is the population size. Small populations might not find the optimal hypervolume and big populations will take long to compute.

Problem sizes $K = [1, 2, 4, 10, 16]$ were analyzed according to the method described in section **??**. Each respective problem size was not able to find their respective best found hypervolume in black box setting. When the threshold is set to 90% figure 3 is obtained. The minimum starting population of MO-RV-GOMEA is 60.

Figure 3 shows the number of evaluations needed to get within 90% of the best found hypervolume and the success rate. $K = 1$ is reliably solvable for any population of the given population sizes. $K = 2$ would be considered reliably solvable when $N = 120$. $K = 4$ is reliably solvable when $N = 360$. All bigger problems are not really reliably solvable within the limited amount of function evaluations. $K = 10$ does not seem to remain stably solvable for growing population sizes. If a population had to be picked for future analysis $N = 480$ will be used. $K = 16$ is solvable for smaller population sizes but fluctuates when N grows. This indicates that the best found hypervolume is too optimistic.

When the function evaluations are examined it appears as if every line has a dent in it where function evaluations start to rise significantly when population sizes get bigger. The population sizes at these dents however are misaligned with what is described as stably solvable above and thus picking these populations will take fewer function evaluations at the cost of decrease in reliability.

3.2.3 Proposed improvements. Two improvements are proposed in this report for MO-RV-GOMEA in a white-box optimization scenario. The first improvement is the implementation of partial evaluations. When a few parameters of a solution are changed, currently the fitness functions must be reevaluated entirely. Partial evaluations allows the evaluation of the fitness functions without assessing all the parameters.

Concretely, let $f_1(\theta) = \frac{1}{K}(|w_0|^2 + ... + |w_K|^2)$ be the second objective function and w_j is the only parameter that is changed ($j \in \{0, ..., K\}$). It's easy to see that if w_j is changed, the new objective function can be calculated as $f_1^{\text{new}}(\theta) = f_1^{\text{old}}(\theta) + \frac{1}{K}((w_j^{\text{new}})^2 - (w_j^{\text{old}})^2)$. No additional storage cost is required assuming that $f_1^{\text{old}}(\theta)$ is already stored for logging purposes.

For $f_1(\theta)$, you can to store $Q_i^{old} = \widehat{y}_i - \sum_{k=1}^{K} r(x_i|\theta_k^{\text{old}})$ for every data point such that $f_0^{\text{new}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(Q_i^{\text{new}})^2$, where $Q_i^{\text{new}} = Q_i^{\text{old}} + r(x_i|\theta_j^{old}) - r(x_i|\theta_j^{new})$, where N the amount of data points (not population size).

Analysis shows that if a single θ is changed, the number of operations stays the same except for the number of operation that calculate $r(x_i|\theta)$. Those drop to $2n$ compared to $K \cdot n$ for a full evaluation.

The analysis also shows that partial evaluations is only efficient when less than $\frac{K}{2}$ θ's are changed. Otherwise full evaluations are faster (overhead has been implemented to check this). The memory usage grows significantly as each solution has to store an additional N variables. Bigger population sizes further increase the risk of cache misses.

The second improvement tries to obtain a fitter initial population via an adaptation of the naive k-means (or k-means) clustering algorithm. Section 3.2.2 showed that it's relatively difficult to find solutions that optimize f_0, while optimizations of f_1 are easily found. Using clustering, a few solutions are put into the initial population to kick start the optimization of f_0. The intuition behind the clustering algorithm is that if f_0 needs to be optimized, then each radial function should have some "responsibility" for a select number of data points.

The algorithm starts by initializing K number of clusters. Each cluster represents the points associated to a radial function. The clusters centers start at a unique random data point (instead of randomly in the solution space) to ensure that clusters are not empty. Each data point is then assigned to the nearest cluster. The center of the cluster is recomputed and this process continues until the centers do not move or until a certain amount of iterations have passed.

Depending on the initialization location different clusters may be obtained. Techniques exists to pick "the optimal" clusters but that's not the goal of this report. The first found clusters are used to kick start the process. MO-RV-GOMEA is given the task to undo the errors made by the clustering. The clustering algorithm works for any problem size or set of data points, so it's well generalizable.

Each kick cluster solution is is generated by letting the j-th cluster generate the parameters of θ_j. w_j influences the maximum height of $r(x|\theta_j)$ so it's associated to the y-value of the cluster. μ_j and σ_j influence the position and the width of $r(x|\theta_j)$

respectively so it's associated to the x-value of the cluster. The standard deviation of the clusters are computed and w_j is uniformly drawn from $\left[c^{\text{center}}_{y,j} - c^{\text{std}}_{y,j}, c^{\text{center}}_{y,j} + c^{\text{std}}_{y,j}\right]$, μ_j is drawn from $\left[c^{\text{center}}_{x,j} - c^{\text{std}}_{x,j}, c^{\text{center}}_{x,j} + c^{\text{std}}_{x,j}\right]$ and σ_j is drawn from $\left[0.1, c^{\text{center}}_{x,j} + c^{\text{std}}_{x,j}\right]$.

The other solutions can be initialized in the original range, but since $[-100, 100]$ is quite far away one might as well initialize closer. The decision was made to initialize the other solutions the same way as described above, except that the standard deviation is multiplied by a factor, e.g. 10.

Doing this process brings the downside of optimizing two additional hyper parameters. The multiplication factor is in my opinion a hyper parameter that's not interesting as one can just choose to initialize in the old range. The interesting parameter is how many kick start solution one should put into it's initial population.

3.2.4 Analysis white-box optimization. The thought behind the implementation of partial evaluations is that algorithms are faster in the sense that in the same amount of time more evaluations can be done. That means that if a full evaluation counts as 1 evaluation then a partial evaluation should count as something less. Section 3.2.3 showed how much a partial evaluation should theoretically be worth with respect to a full evaluation. It did this based on number of operations and execution time. This method however hard to calculate especially because of the overhead of the algorithm.

Balancing the partial evaluation count is important. An effect of not having a fairly balanced partial evaluation count is that partially evaluated algorithms will evaluation-time wise be more efficient (as expected), but terminate at a much later time than their fully evaluated algorithm counter part.

To show an example, figure 4 shows `MO-RV-GOMEA` partially and fully evaluated when a `FOS` subset of 3 consecutive parameters is used for various problem sizes. The y-axis shows the computation time in seconds, the x-axis the number of real evaluations. Generation wise, both evaluation types should have the same number of real evaluations, but partial evaluations take less computation time.

When $K = 1$, the required computation time of a partially evaluated algorithm is roughly the same as a fully evaluated algorithm. The reason for this is that generations pass quickly and thus the algorithm is mostly spending time doing overhead, e.g. writing to a file. It's clear that there's almost no improvement time wise. If partial evaluations are discounted the problem described above would occur. A gap starts to form when $K = 4$. Since there are more `FOS` elements to analyze the algorithm spends more time doing evaluation and thus completes faster compare to $K = 1$. When $K = 16$ a clear difference between partial and full evaluations appears.

The question remains what a fair method would be to compare full and partial evaluations. What I chose to do is for every experiment determine the average ratio between fully and partial evaluations at 200.000 evaluations of 30 runs and use that. This ratio should give a decent lower estimate as the averages seem to converge.

Figure 5 shows the convergence of partial evaluations for $K = 10$, $|P| = 480$ (the optimum of blackbox evaluation) and $|A| = 500$. Each experiment was run 60 times. The usage of linkage models is more feasible evaluation count wise compared to the black-box case without partial evaluations. The linkage models remain pretty far from the best found solution. There's no significant difference between the linkage models and the black-box implementation hypervolume wise. The linkage model does seem to be able to reach better solutions compared to the blackbox implementation.

For the second improvement it was argued that it wasn't important how to initialize the remaining non cluster solutions. It is however interesting to ask how many solutions should be added to the initial population.

Figure 6 shows the hypervolume vs the proportion of cluster solutions in the initial population. The experiments were done with a fully evaluated algorithm where each parameter is considered dependent on each other. Furthermore the settings were left to the recommended value, except for $K = 10$, $|P| = 100$ and $|A| = 500$. The blue line represents the mean of 60 runs, the green line the best found solution and the red line the standard deviation with respect to the mean.

At -1% no cluster solutions are used and the algorithm is initialized in the default range ($[100, 100]$). At 0% the algorithm is initialized using the cluster's standard deviation times 10, but no cluster solutions are added. The maximum hyper volume that can be found improves at the cost of significant increase in variance. As cluster solutions are added to the initial population, the maximum hypervolume remain the same, but the variance takes different values.

In the current set-up `MO-RV-GOMEA` contains 20 clusters. 2 clusters are used to optimize a single objective [**?**] each having 10 solutions. It was expected that having 10 or slightly more solutions would be optimal since it would fill an entire/two cluster(s), but this does not truly appear to be the case. The optimum appears to be around 8% (which isn't 10%) and a local optimum appears at 12%.

Figure 7 shows two above average fronts obtained with cluster solutions and without for $K = 10$, $|P| = 100$ and $|A| = 500$. The front without cluster solutions does not show big holes but has issues maximizing f_0. The front with cluster solutions shows a better front but also has issues maximizing f_0. The reason for this will be later investigated.

Figure 8 shows the hypervolume and hypervolume improvement against the number of evaluations for various implementations where $K = 10$, $|A| = 500$. The implementations shown use partial evaluations and cluster solutions and is shown with different linkage models using the optimal parameters found above. Each experiment was run 60 times.

The implementations that use cluster solutions to initialize on average start off from roughly the same hypervolume. This starting hypervolume is far superior than the blackbox case. These implementations also are also able to obtain better hypervolume. The drawback is that these implementations have a lot more variance.

The high variance observed in figure 8 and the fact that f_0 does not seem to be optimized at all can be mainly attributed to the (poor) choice of clustering. As stated in section 3.2.3 there's no optimization process to pick the "best" cluster. This means that clusters can be unbalance where too many data points are assigned to a single cluster or that clusters are too concentrated in a region. This causes `MO-RV-GOMEA` to get stuck into a local optimum it can't get out of within the limited number of function evaluations.

Another issue is that when the perfect clusters are chosen and f_0 is optimized properly it finds objective functions with values above the reference point. These points do not contribute anything to the hypervolume. The algorithm continues to optimize the single objective while leaving big gaps that algorithm can't fill fast enough.

In both cases MO-RV-GOMEA presents a sub-optimal hypervolume. In order to solve the increase in variance, effort should be put into analysis of the clusters. There are plenty of indicator functions that measure the quality of the clusters. The drawback is that these methods require more computation and often multiple iteration which will increase overhead. Another method might be to exploit the different sub-optimal clusters as these solutions might not yield solutions that perfectly fit in between the gaps. The reference point should also be set at the theoretical maximum to to capture the full hypervolume. The output file should then of course show more precision.

At last the optimal population size is analyzed on the implementation using both improvements and a linkage set size of 3. The same methods and threshold as done in section 3.2.2 apply.

Figure 9 shows the results of the experiment. Problems bigger than $K = 4$ remain unreliable, but even worse the reliability dropped significantly. This has to do with the large increase of variance deducted in previous experiments.

As for the improvements on the smaller problems, all problems are solvable with the smallest population. Moreover, fewer evaluations are needed due to partial evaluations. The number of evaluations seems to drop down after $N = 960$ for $K = 1$ and $K = 2$. Analysis shows that population sizes smaller than $N - 960$ usually finish in 1 generation. Larger population sizes sometimes find hypervolumes in the initial population large enough to pass the termination threshold.

3.3 Joris Feijen

In this section the Non dominated Sorting Genetic Algorithm (NSGA2) is investigated. NSGA2 is an algorithm for multiobjective optimization that incorporates elitism using a fast nondominated sorting approach. It is an improvement on the earlier version NSGA.

3.3.1 Key Features. The key concept of NSGA2 is the nondominated sorting algorithm, which sorts the pool of individuals into multiple fronts according to their dominance. Firstly, for each individual p two entities are calculated: the domination count n_p, i.e. the number of individuals that dominate p, and S_p, the set of individuals that are dominated by p. Nondominated solutions will, by definition, have $n_p = 0$. They are put into the first front and, subsequently, the members q of their set S_p are visited and its domination counts are reduced by 1 ($n_q = n_q - 1$). If doing so makes their domination count zero, then they are added to the next front. This procedure is repeated until all fronts are found. This algorithm reduces the time complexity from $O(MN^3)$ to $O(MN^2)$ compared to a naive approach which was used in the earlier version NSGA.

Another improvement is the use of the crowding distance as a means to preserve diversity in the front during the selection process. The crowding distance of an individual is defined as the sum of the difference in normalized objective values between its two neighbours, for all objective values. The benefit of this approach is that it does not have a parameter and thus the diversity of the solution is independent of parameters of the algorithm. Now, a new comparator is defined as in equation 11.

$$\begin{aligned} & i \succ_n j : \\ & \text{if} \quad (i_{rank} < j_{rank}) \\ & \text{or} \quad (i_{rank} = j_{rank} \text{and} \quad i_{distance} > j_{distance}) \end{aligned} \tag{11}$$

The main algorithm can now be described as follows: At the start of generation t the parents and its offspring are put together in a set $\mathcal{R}_t$. These are sorted into a set of fronts $\mathcal{F}$ using fast nondominated sorting. Then the new population is filled with individuals from the first fronts until adding another front would exceed the population size. That front is then sorted using the comparator of equation 11 and the remainder of the new population is filled from this sorted front in increasing order. The new offspring is generated from this new population using crossover and mutation. The operators used are SBX crossover and polynomial mutation[**?**] with their hyperparameters set to the recommended value of 20. Because the new mating pool is selected from both parents and offspring, elitism is ensured.[**?**]

3.3.2 Analysis black-box optimization. Since the algorithm performs real-valued optimization, the Pareto front might contain an infinite number of solutions, which means the non-dominated front can quickly grow large enough to fill up the entire population. Because of this, non-dominated solutions can be left out of the population and thus not added to the approximation front. Therefore an improvement is made upon the original algorithm by additionally implementing an elitist archive (details can be found in section 3.3.3). Apart from that, the other algorithms considered

in the report also have an elitist archive, so NSGA2 must have an elitist archive as well in order to obtain a fair comparison. In the BBO analysis, firstly, the influence of the added elitist archive is investigated. Then, an attempt will be made to find the optimal population size for this specific algorithm, using the method described in section 2. Having found the optimal population size, it is interesting to look at the difference in performance of the algorithm using different numbers of kernels. Using more kernels increases the search space of the algorithm, from which it could be concluded that better and more diverse solutions can be found. All described experiments, including the ones in section 3.3.4 are performed using the following parameters unless specified otherwise: population size is 6, evaluation budget is 200000, $K = 10$ and elitist archive size is 500. Finally, all settings that are tested are run 10 times to obtain the statistics of the outcome, unless specified otherwise.

Influence of elitist archive
To test the effectiveness of the elitist archive the algorithm is given a relatively small budget 50000 function evaluations, since we do not need the algorithm to be fully converged, and a population size of 10. The hypervolumes per generation are stored for the normal population and for the elitist archive. The experiment is then repeated 10 times and its mean and standard deviation are reported. The experiment is run in two settings: $K = 3$ with an elitist archive size of 500 and $K = 10$ with eltist archive size of 100.

The results are shown in the top and bottom part of figure 10. From this figure, two conclusions can be made. The first is that, even though they remain close, the algorithm performs consistently better when making use of an elitist archive. The second is that the claim of elitism without explicit use of an elitist archive is false according to this graph. It can clearly be seen that the hypervolume is not monotonically increasing, which somewhat counters the claimed elitism. A good solution can still leave the population when the first front has more solutions than the population size and this good solution has a lower crowding distance. That is why the hypervolume can still decrease. Using an elitist archive would solve this problem, given that the archive has no bounds. In the case of a low value for K ($K = 3$), a size of 500 might be enough to store all nondominated solutions. However, when $K = 10$ a size of 100 is not enough as can be seen in figure 11, the hypervolume is still nondecreasing. The reason for this remains to be investigated.

Optimal hyperparameters
The optimal population size is, strongly related to the number of variables used. The crossover and mutation operators of NSGA2 are based on their binary counterparts and hence behave similarly. Therefore, when many variables are used the population must be larger in order to represent the whole search space. This can also be seen in figures 12. In the comparison between all algorithms the report considers, only $K = 10$ is used and thus, this value is used in the search experiments. During the search for the optimal population size, it was found that very small population sizes still yield large hypervolumes reliably. However, due to the implementation of the selection operator the population size cannot go lower than 6. The complete search was run and all 10 times the problem could be solved reliably with a population size of 6. The mean and standard deviation of the hypervolume that were found during the search are 143004.67 and 1587.82.

The other two hyperparameters of the algorithm are the SBX parameter η_c and the mutation parameter η_m. The first parameter controls how close to its parents the variable of the offspring is generated during crossover, with a larger η_c corresponding to a bigger range. The second parameter has the same interpretation but is used for the mutation operator. The same experiment as for the population size is run for both hyperparameters individually. When searching for the value of η_c, η_m is kept at the recommended value of 20 and vice versa. To confirm that both values together also work well, an extra test was run using both values that were found. The results are shown in table 1. It was found that with a $\eta_c = 1$ and $\eta_m = 2$, much lower than the recommended values, the problem could already be solved reliably. A possible explanation for this is the large function evaluation budget. Having these parameters low means that the solutions evolve slowly but because the algorithm is allowed to run for a long time, it can still get close to the optimum.

Influence of K
The hypothesis that a larger K leads to a higher hypervolume is tested by running the algorithm for different values of K ($K = [2, 4, 8, 16, 32]$) with a population size of 6 and a budget of 50000. As was stated before, the population size might be a limiting factor when increasing K. Therefore, a "sufficiently large" population size of 60 is also tested for each K. The results of the analyses with population size 6 and 50 are shown in figures 12 and 13. Comparing figure 12 and 13, we observe that the increase in the standard deviation with increasing K diminishes when the population size is increased to 50. This confirms the earlier stated fact that using more variables requires a larger population size. Moreover, the bottom graph of figure 13 shows a strange superiority of $K = 8$. Seeing no reason for $K = 8$ to perform better, we hypothesise that the reason for this is because the hypervolumes have not converged yet. That is why yet another analysis is performed with a "sufficient amount" of function evaluations which is chosen to be 200000 of which the results are shown in figure 14. Now it is seen that, except for $K = 32$, increasing K gives a better hypervolume. The fact that $K = 32$ is still below the others likely is because of two reasons: 1)The standard deviation of $K = 32$ has not changed, while all others have converged, meaning that a population size of 50 is still not sufficient for the algorithm to reliably solve the problem with $K = 32$. 200000 function evaluations might not be enough for the hypervolume to converge. However, because increasing the population size and maximum number of function evaluations even further would tremendously slow down the analysis, this is not investigated any further. Nevertheless, our hypothesis for increasing K is confirmed for three values of K.

3.3.3 Proposed improvements. As mentioned in the previous section, an additional elitist archive was added. One of the key features of the algorithm was the use of crowding distance in the selection and hence this feature will also be incorporated in the algorithm for the elitist archive as to preserve the properties of NSGA2.
The algorithm works as follows: Each candidate is compared against

all members of the archive. As long as the maximum size of the archive is not yet reached, the candidate is added to the archive if it dominates at least one solution. The dominated solutions in the archive are then removed. If the maximum size of the archive is reached and a solution is found that is not dominated by any member of the archive, then the crowding distance is compared. When the crowding distance of the candidate is higher than the lowest crowding distance within the archive, the candidate is added and the solution with the lowest crowding distance is removed from the archive. The effect of adding the elite archive was shown in section 3.3.2.

Besides the additional elitist archive, another method is proposed to improve the performance of the algorithm. From figure 15 it is seen that solutions are more numerous with a low value for the second objective function (f_1) than solutions with a low value for the first objective function (f_0). Hence, it would be desirable to make the algorithm produce more solutions with a low value for f_0.

The proposed method generates new data similar to the original dataset and trains the algorithm on this enlarged dataset. Consider an arbitrary solution x with fitness values f_{x0} and f_{x1}. When the dataset is expanded, the value for f_{x0} will increases, because it is the sum of the squared errors, while the value for f_{x1} will remain the same because x has not changed. In short, this method implicitly puts extra weight on f_0 similar to an "inverse" regularizing parameter. Where a regularizing parameter forces the solution to become smooth, this method forces the solution more towards overfitting. In this way the selection pressure is moved more towards optimizing f_0. The extra data is generated by sampling from a probability distribution that is estimated from the original dataset, by means of Gaussian kernels. The Gaussian kernels are assumed to have no covariance.

3.3.4 Analysis white-box optimization. To analyse the performance of the proposed improvement, a comparison must be made between a baseline without the improvement and one with improvement. The hypervolumes are calculated based on the fitness of all solutions. Adding more data inherently increases f_0 and thus also alters the hypervolumes. That is why the extended dataset will only be used for evolving the population. The resulting parameters will be evaluated on the original dataset so that they may be used to compare against the black box case.

In order to get the best out of the improvement, two parameters are tuned: the fraction with which to increase the dataset f and the variance of the Gaussian kernels used for density estimation σ. A simple search is search is performed, while keeping all other parameters at the recommended values, over a range of values for these two parameters: $\sigma = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]$ and $f = [1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]$. Since the solutions will eventually be evaluated with the original dataset, it is not desirable to generate data far away from the original datapoints, leading to the hypothesis that lower values for σ will be better. Next in order to make a difference, enough extra data points need to be generated, but not too many because that would change the original distribution and perform bad when evaluated on the original dataset. The bounds of the range to search in were determined through examining the generated dataset for different values. Figure 16 shows the lower extremes. The data points are so few that they do not contribute enough, and too close to the original data points to make a difference. Figure 17 shows the upper extremes. Here the generated data goes too far beyond the original data. Additionally, too many points are generated, completely changing the original distribution. A balance is expected to be found somewhere in between these extremes like in figure 18.

The results of the parameter search for the factor f are presented in table 19. It is seen that the best value lies around 1.6 and 1.7. The value of 1.6 is chosen. The results for the search of σ are shown in figure 20. As predicted high values of σ perform worse. The best value appears to lie at the first value that was tested, 0.1. A second smaller search is performed around this value using a smaller evaluation budget of 50000. In the second search each parameter setting is run 20 times in stead of 10 to obtain more accurate statistics about the solution. The results of the second search are shown in figure 21. It is seen that around $\sigma = 0.1$ the values do not differ that much, therefore this value is chosen as the optimum.

A final run with the obtained parameters, $\sigma = 0.1$ and $f = 1.7$, and the "default" settings for the remaining hyper parameters. The results are put table 2 together with the final results of the BBO. Moreover, figure 22 compares the best fronts obtained by the WBO and BBO. As can be seen from both the table and the figure, generating extra data does not improve the algorithm. The original idea was to get the algorithm to produce more solutions in the left area of the objective space. Looking at figure 22, it is seen that there are relatively more solutions in this area. However, because the solution had to be evaluated with the original dataset, the solution could not get a low enough f_0 to compete with the BBO.

In short, the goal of getting more solutions with a low f_0 was achieved, albeit marginally, but it did not make the algorithm perform better. Perhaps, directly increasing the weight of f_0 would be more effective as it does not involve evaluating on a different training set than it has seen during its evolution.

3.4 Matthijs de Groot

In this section the UHV-GOMEA algorithm as described in [**?**] is investigated.

3.4.1 Key Features. The UHV-GOMEA algorithm is an adaptation of the GOMEA algorithm which uses the uncrowded hyper volume as an indicator.

Uncrowded Hypervolume The uncrowded hypervolume metric as used by this algorithm is based on the uncrowded hypervolume improvement as introduced in [**?**]. The uncrowded distance $ud(x, \mathcal{S})$ is defined as the shortest Euclidean distance between x and a point on the *approximate boundary* between the dominated and non-dominated solutions.

The uncrowded hypervolume improvement is a quality measure with respect to $\mathcal{S}$ but not for $\mathcal{S}$ itself. The algorithm therefore uses the uncrowded hypervolume (UHV), defined as the hypervolume of $\mathcal{S}$ penalized by all uncrowded distances.

$$UHV(\mathcal{S}) = HV(\mathcal{S}) - \frac{1}{|\mathcal{S}|}\Sigma_{x\in\mathcal{S}} ud(x, \mathcal{S})^m$$

Indicator-based multi-objective optimization problem (IBMOP) In an IBMOP a quality indicator is used to assign a quality to a solution set. A single objective optimizer can be used to optimize the multi-objective problem used by optimizing entire solution sets based on a quality indicator. In this algorithm the UHV is used as the quality indicator.

UHV-GOMEA The GOMEA algorithm is used to optimize IBMOPs with the UHV as the quality indicator. Because the UHV indicator maps an entire solution set to a single value the underlying problem can be multi-objective while the single-objective GOMEA optimizer is used.

3.4.2 Analysis black-box optimization. The UHV-GOMEA implementation that was used for this analysis, retrieved from [**?**]. The algorithm uses UHV as the quality indicator, this means that the algorithm will perform its optimization over complete solution sets. The size of the solution sets have a large impact on the efficiency of the algorithm. Another parameter that has a large impact on the algorithm is the population size. The third parameter that was found to have a large impact is the elitist archive size. As mentioned in [**?**], the elitist archive does not play a significant role in increasing the performance of algorithm but it does have a large impact on the efficiency. The last factor that was studied during this analysis was the impact of the value of K.

When tuning and analysis of the algorithm limits were set on the maximum amount of evaluations and the maximum time duration. The maximum amount of evaluations was set to 200000 and the time limit was set to 300 seconds. The amount of evaluations was limited to create fair comparison between the amount of evaluations required for convergence to a certain point. The time limit was set to ensure feasible test runs. Every hyperparameter set was run for 30 times to obtain a fair mean, min and max value for the hyperparameter and reduce the impact of randomness.

Population size The population size was tuned using the bisection method. The algorithm was run using exponentially increasing population sizes until a population size was found that reached $0.95 \cdot \text{opt}$ for at least 9 out of 10 runs. After which the found population size, divided by 2 was exponentially increased with base 1.1 until a population size was found that succeeded in at least 9 out of 10 runs. This was the final population size that was used in the rest of the experiments. The progression of the population size is shown in Figure 36. The result of the tuning was that a population size of 32 is optimal and will be used for further tests and tuning.

Solution Set Size The solution set greatly influences the efficiency of the algorithm because the solution set size determines how many solutions are evaluated. A large solution size requires more computations per generation and it therefore decreases the amount of evaluations that can be done within the 300 second time limit. It turned out that a relatively small solution set size of 8 worked well for the algorithm. Since the solution set size also determines how many points are on the Pareto front it should be noted that the resulting Pareto front is very sparse. In Figure 35 it is shown that due to the increase in computational complexity the tests with higher solution set sizes actually perform worse than the smaller solution set sizes. The solution size used for the rest of the comparison is 8.

Elitist Archive For UHV-GOMEA there are two major problems with using the elitist archive. The first problem is that, due to the nature of the algorithm, the elitist archive has a very large impact on the efficiency of the algorithm because the elitist archive holds complete solution sets and the comparison is therefore computationally expensive. The elitist archive did not achieve any significant performance gain, instead it performed worse in most cases, as seen in Figure 38. Because time complexity increases but performance does not I decided not to use the elitist archive implementation.

The impact of K Even though the value of K is more a problem related parameter than a tunable parameter of the algorithm itself it is still useful to determine what the impact of a higher value of K is on the both the algorithms performance as well as its efficiency. Using the tuned population size, solution set size and elitist archive several tests were run to obtain the plot in Figure 34. It is shown that for all the values of K the minimal hypervolume and the maximum hypervolume achieved in 10 runs is similar, but that the median hypervolume does increase slightly at higher values of K. The required time for the 200000 evaluations increases at a much higher rate than the median hypervolume.

Variance As seen in Figure 44 and Figure 43 the variance of UHVGOMEA was very high initially. After further tuning the parameters the variance was much lower as seen in 38. It does show that UHVGOMEA is vulnerable for a high variance but is also capable of high performance. This is something that was experienced during many of the tests and is the reason why some of the tests were repeated with 30 instead 10 runs. When a larger run size was used the median got closer to the maximum so the potential for good performance is definitely present.

3.4.3 Proposed improvements. Partial evaluations can be used in the GOMEA algorithm because the next solution shares its DNA with the previous solution, and therefore only the parameters that are changed have to be recomputed. This was implemented using a method that first scans if any of the three dependent parameters that make up θ is changed, if so it will recompute that subsolutions of only that radial regression function. The expectation is that the algorithm will work better because partial evaluations are only counted for the percentage of parameters they change. Thus with

a smaller amount of evaluations the same amount of solutions can be evaluated.

3.4.4 Analysis white-box optimization. We performed the same analysis as we did with the black box optimization. Comparing the influence of the different hyper parameters on the amount of function evaluations and the hypervolume convergence.

Population size The effect of population size on the white box optimization is shown in Figure 41. It is again shown that a to high population size actually hurts the performance. The optimal population size is somewhere between 16 and 32.

Solution Set Size The effect of solution size is shown in Figure 40. The reason that the results are spiking in the first few generations is due to the partial evaluations. At further generations the partial evaluations behave more like full evaluations. The figure also shows that solution set size that is too high negatively influences the performance, even more so than with the black-box optimization.

Elitist Archive In the white box optimization with partial evaluations the elitist archive has a similarly small impact, as seen in Figure 42. The UHVGOMEA algorithm in general has little value for the elitist archive, as noted in [**?**].

The impact of K In Figure 39 it is shown that a lower value of K is required with partial evaluations than with full evaluations. The figures can also be explained by a wrong or bad implementation.

General conclusion The improvement of the white box optimization is largely dependent on the efficiency of the partial evaluations. My implementation proved insufficient. It is also dependent on the way partial evaluations are valued. When valued at $changed/total$ parameters it proved to be an improvement in many cases while unreliable at the start of the algorithm.

In order to better understand why the algorithm performed as bad as it did. Another observation is that in Figure 42, which was conducted over a fairly large number of runs, the white-box optimization performs much better than the different figures which may mean that upon further study prove that the algorithm is actually better than the current figures indicate.

4 COMPARISON ALGORITHMS

In previous sections the algorithms were optimized for black-box optimization. In this section the optimized algorithms are compared. Figure 43 and 44 show the hypervolumes obtained vs number of evaluations for various algorithms. K was set to 10 and if an elitist archive was used it was set to size 500.

The shaded areas in figure 43 show the standard deviations. All algorithms seem to converge to roughly the same hypervolume except for NSGA-II. The algorithm with the largest standard deviation is UHV-GOMEA. MaMaLGaM seems to converge the fastest. MO-RV-GOMEA seems to be somewhere in between MaMaLGaM and the other algorithms. Although not visible in figure 43 it also has the fewest number of data points due to the massive amounts of evaluations done per generation. NSGA II seems to be performing the worst.

The shaded areas in figure 44 show the minimum and maximum obtained hypervolume for each algorithm. The threshold value used for population optimization has been visualized as the black line. Although NSGA II was performing the worst it was able to stably obtain the hypervolume threshold. UHV-GOMEA on the other hand seems rather unstable, as discussed in 3.4.2, but is able to obtain the best hypervolume between the other three algorithms.

MaMaLGaM, MO-RV-GOMEA and UHV-GOMEA use similar methods. It's interesting too see the similarities and differences to try and understand what makes the algorithms obtain the results it has obtained.

Similar to MaMaLGaM, MO-RV-GOMEA uses samples the estimated distributions to create offspring. Solutions of a population are put into clusters and each cluster creates offspring using data from their respective cluster. A difference in creating offspring is that MO-RV-GOMEA uses the Gene-pool Optimal Mixing (GOM) variation operator to iterative samples parts of the offspring. In other words MO-RV-GOMEA is able to use linkage models. UHV-GOMEA can also use linkage models. MaMalGaM on the other hand can't use .... Although the use of linkage models should be perfect for our problem, all algorithms that can handle linkage models used a fully linked linkage model mainly because of how expensive it is to do evaluations given the limited number of evaluations.

Why MaMalGam is converging faster is then attributed to its variation operator. The variation operator uses previously found covariance to estimate the current covariance. This scales back the population size, which in turn requires fewer evaluations per generation.

5 CONCLUSIONS

We have concluded that MaMaLGaM is the most suitable choice among the algorithm's that were analyzed because it has desirable features, mainly that it converges fast and has achieves a stable hypervolume. The hypervolume is not the highest among the algorithms but it is the most stable. The (slightly) lower hypervolume is more than compensated by its efficiency and stability.

The comparison was done based on a single dataset so in order to further confirm our conclusions more tests must be done using different datasets as well as separate training and testing datasets.

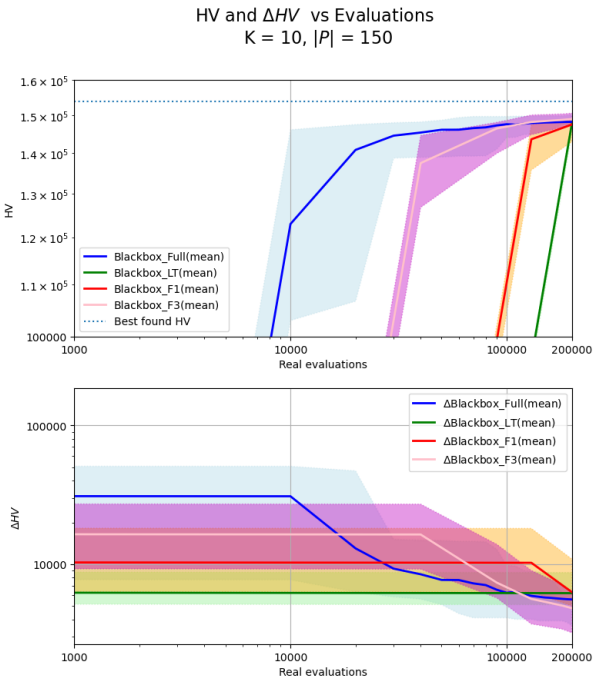

Figure 1: Convergence of MO-RV-GOMEA for different linkage models, where $K = 10$ and $|P| = 150$ and $|A| = 500$. The shaded areas are the minimum and maximum hypervolumes found.

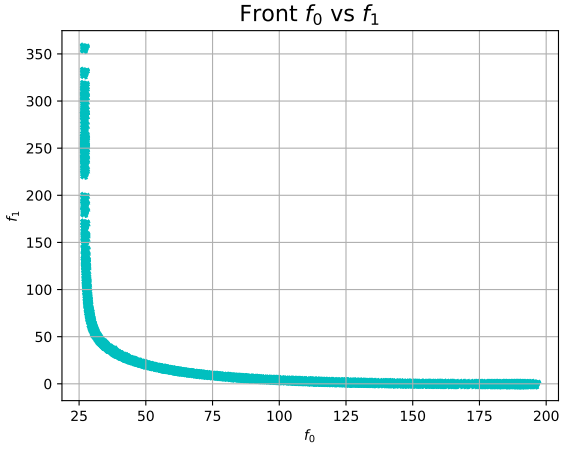

Figure 2: The front obtained by black-box MO-RV-GOMEA when $K = 10$, $|P| = 150$ and the maximum evaluations was set to 1.000.000. All other parameters are set to the recommended value.

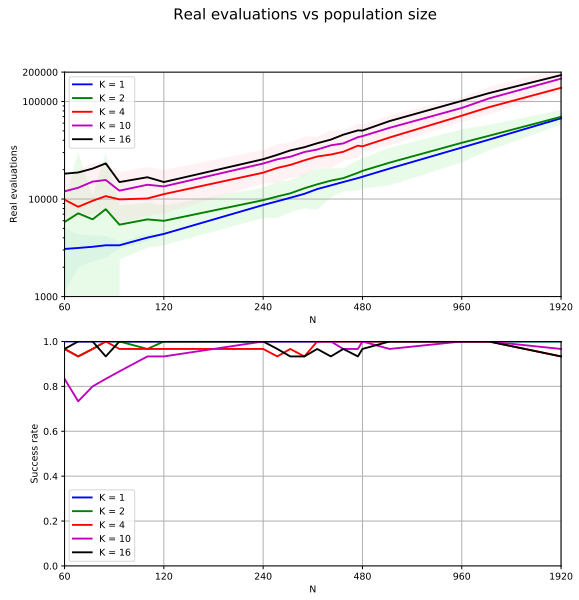

Figure 3: The number of real evaluations required to reliably solve various problem sizes and the respective success rate of 30 runs. Each run is considered successful if it finds 90% of the best respective hypervolume 10 times. The standard deviation is shown for $K = 1, 2, 10$ in order not cloud the figure. The size of the elitist archive is 500.

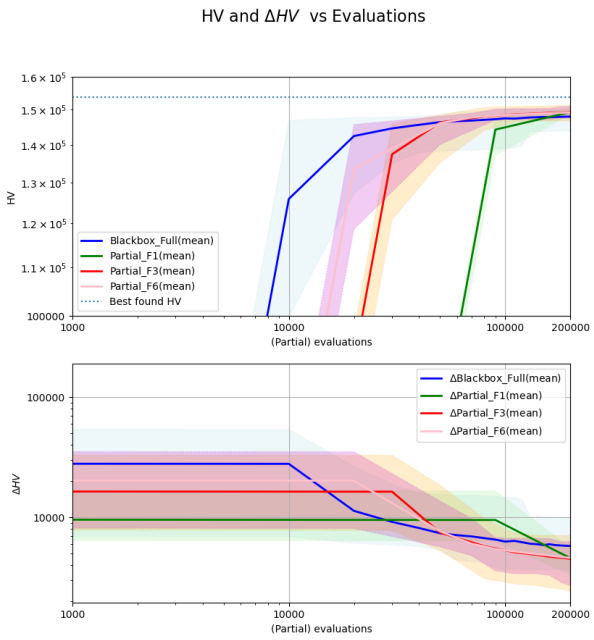

Figure 5: Convergence of `MO-RV-GOMEA` with partial evaluations for different linkage models, where $K = 10$ and $|P| = 480$ and $|A| = 500$. The shaded areas are the minimum and maximum found hypervolumes.

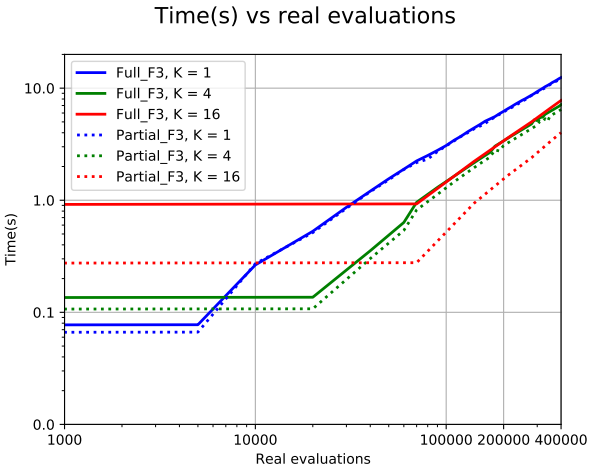

Figure 4: Fully and partially evaluated algorithms with `FOS` subsets of 3 consecutive parameters for various problem sizes. The parameters of the run are the recommended values except for $|P| = 150$, $|A| = 500$. The shaded areas are the standard deviations.

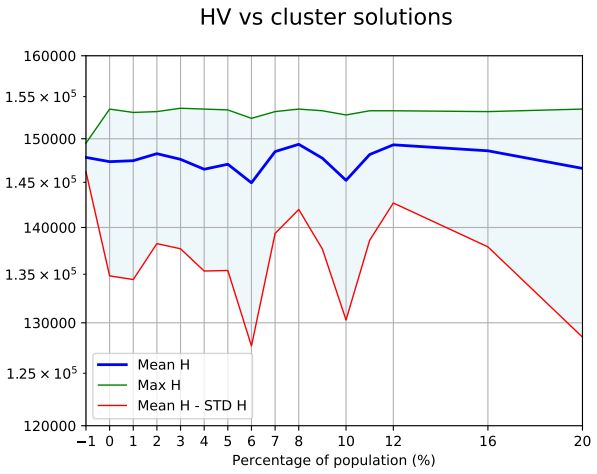

Figure 6: Various proportions of cluster solutions in the initial population vs hypervolume. The experiments were done with a fully evaluated algorithm where each parameter is considered dependent on each other, $K = 10$, $|P| = 100$ and $|A| = 500$ Each experiment was run 60 times and terminated at 200.000 evaluations. -1% initializes in the initial range $[-100, 100]$ without cluster solutions. 0% initializes using the cluster data without adding cluster solutions.

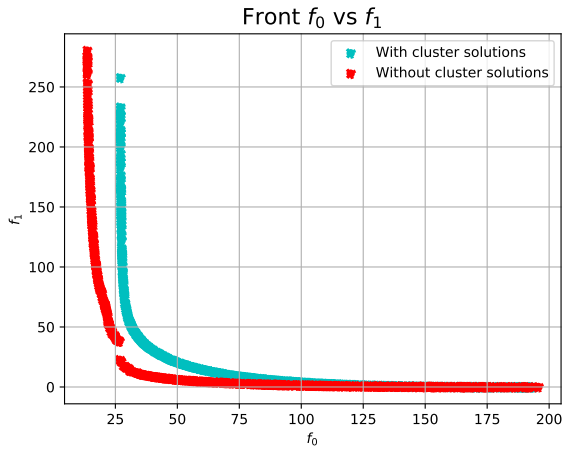

Figure 7: Two random above average fronts. One front was obtained with cluster solution, the other is without. Both algorithms were fully evaluated where $K = 10$, $|P| = 100$ and $|A| = 500$. Both runs terminated at 200.000 evaluations.

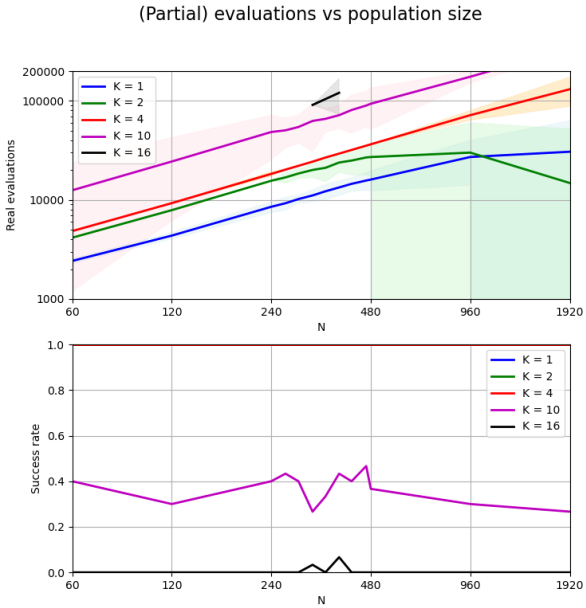

Figure 9: Convergence of MO-RV-GOMEA with partial evaluations, cluster solutions and linkage model size of 3 where $K = 10$, $|A| = 500$ and the number of cluster solutions is 8%. The shaded areas are the minimum and maximum evaluations. Each experiment was run 60 times.

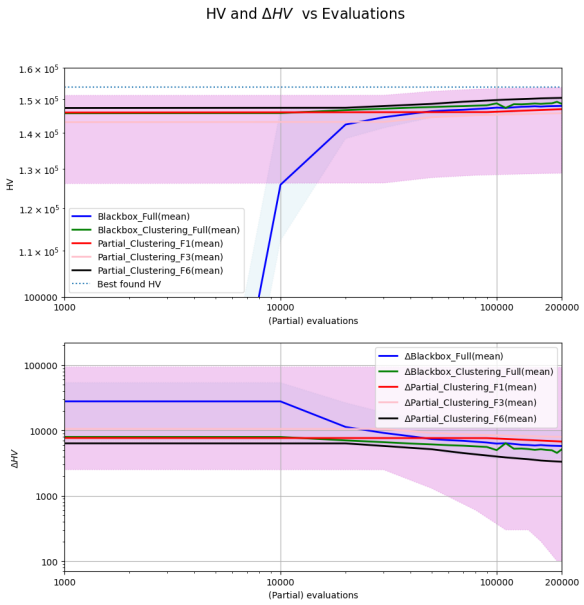

Figure 8: Convergence of MO-RV-GOMEA with partial evaluations and cluster solutions for different linkage models, where $K = 10$, $|P| = 480$, $|A| = 500$ and the number of cluster solutions is 8%. The shaded areas are the minimum and maximum found hypervolumes and are only shown for the black-box and 3 sized linkage set to not cloud the figure.

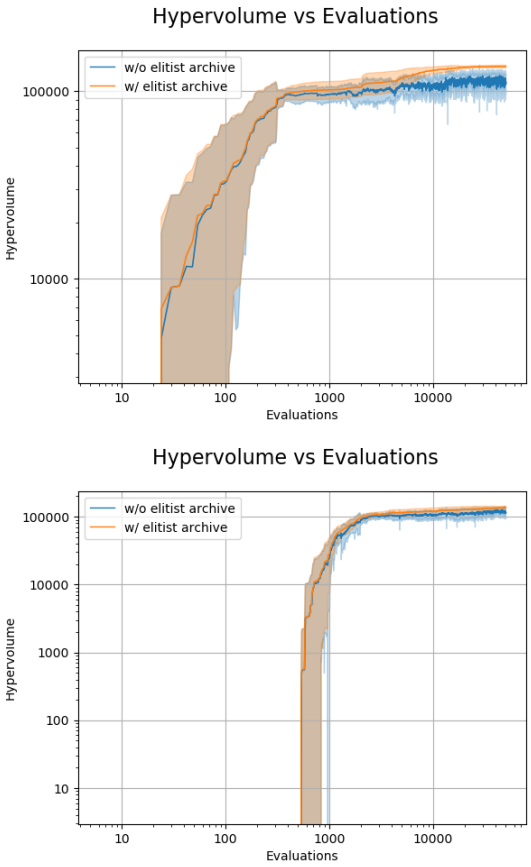

Figure 10: Hypervolumes of the NSGA2 algorithm with and without the use of an elitist archive, run for population size 6 with a budget of 50000 function evaluations. The top figure has $K = 3$ and an elitist archive of size 500, while the below has $K = 10$ and 100. The lines and shaded areas represent the mean and standard deviation respectively.

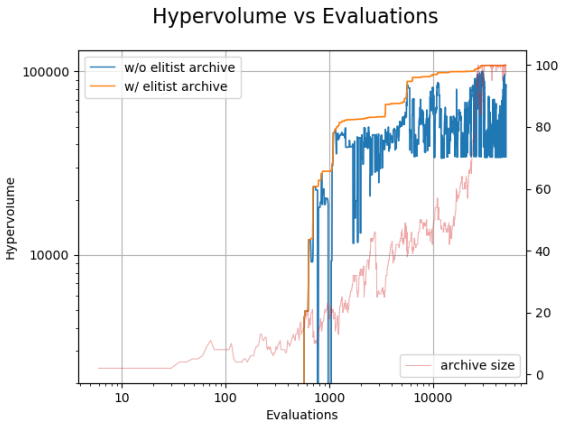

Figure 11: Single run of NSGA2 shown with and without the use of an elitist archive (size 100) for $K = 10$, population size 6 and 50000 function evaluations. The thin red line corresponding to the right y-axis represents the current size of the elitist archive.

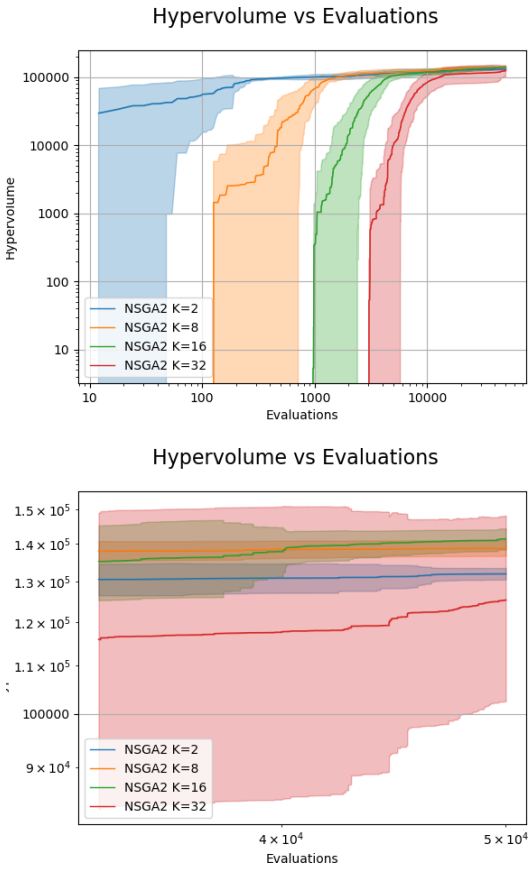

Figure 12: NSGA2 run with different values of K, a population size of 6 and a budget of 50000 function evaluations. The lines and shaded areas represent the mean and standard deviation respectively. The bottom graph is a zoomed in version, showing the last third of the data.

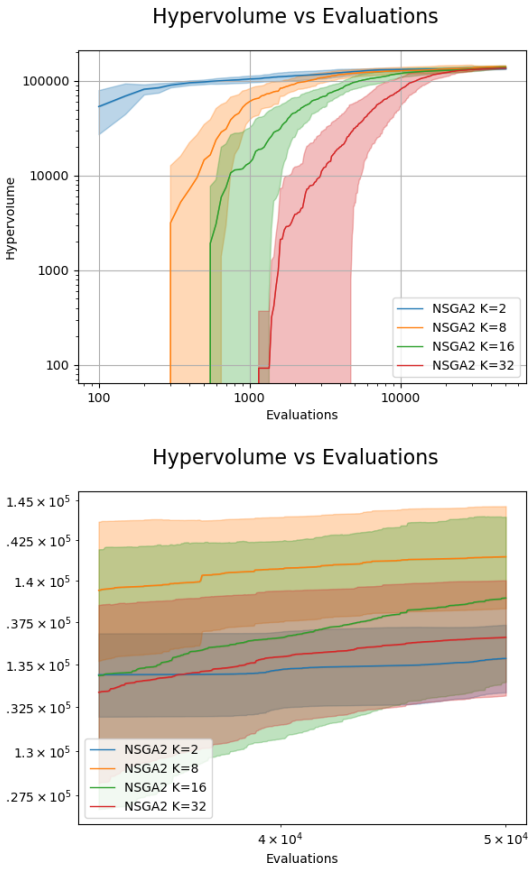

Figure 13: NSGA2 run with different values of K, a population size of 50 and a budget of 50000 function evaluations. The lines and shaded areas represent the mean and standard deviation respectively. The bottom graph is a zoomed in version, showing the last third of the data.

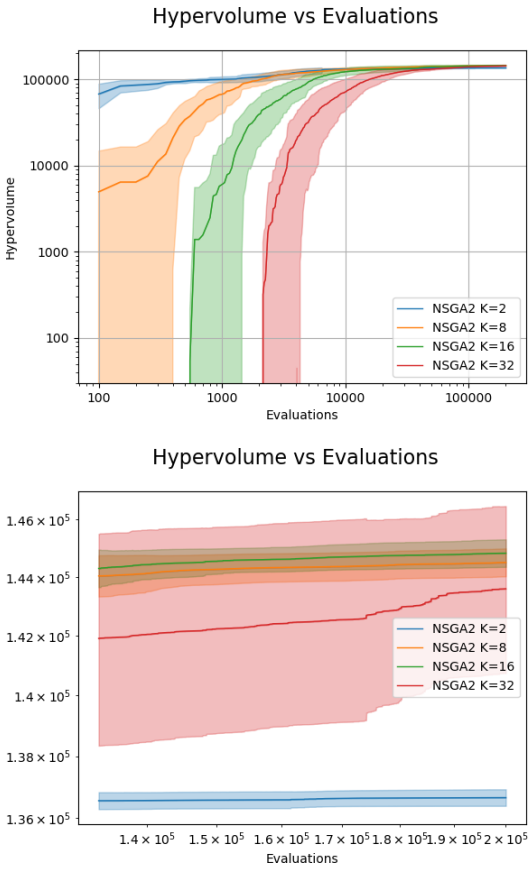

Figure 14: NSGA2 run with different values of K, a population size of 50 and a budget of 200000 function evaluations. The lines and shaded areas represent the mean and standard deviation respectively. The bottom graph is a zoomed in version, showing the last third of the data.

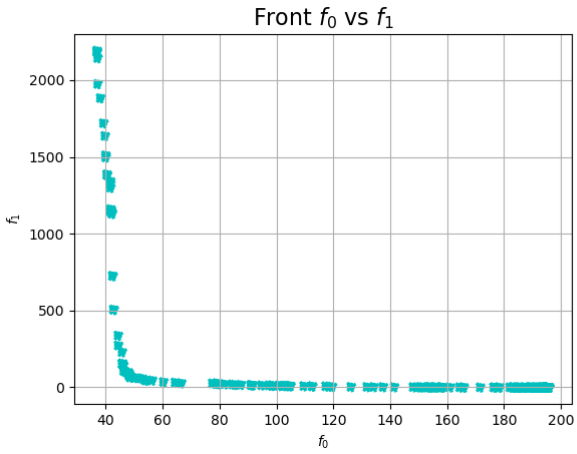

Figure 15: The approximation front obtained with NSGA2 using population size 6, elitist archive size 500, K=10, $\eta_c = 1$, $\eta_m = 2$ and a budget of 200000 function evaluations.

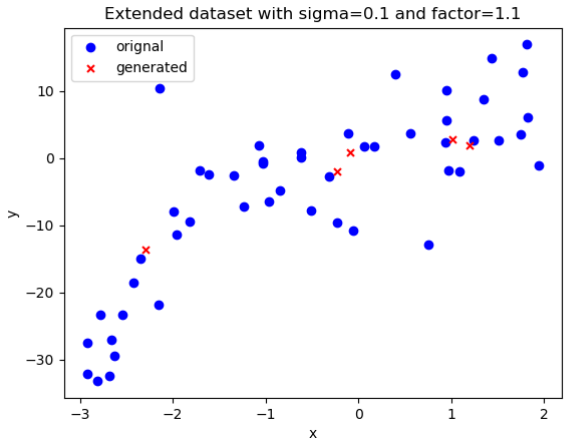

Figure 16: The extended dataset visualized for $\sigma = 0.1$ and $f = 1.1$.

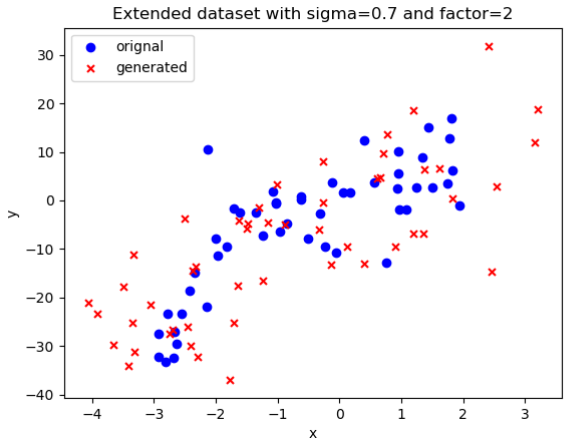

Figure 17: The extended dataset visualized for $\sigma = 0.7$ and $f = 2$.

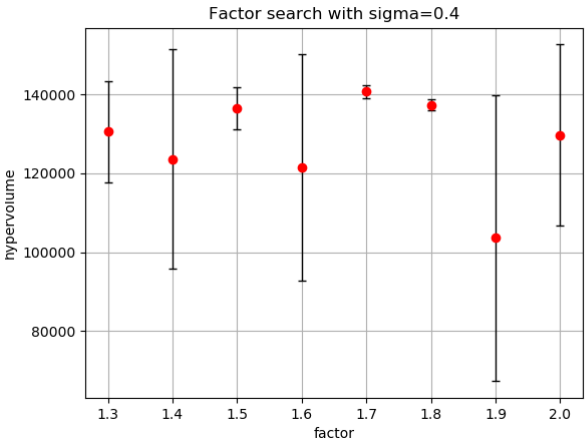

Figure 19: A search for the factor parameter for the improvement of NSGA2. σ is kept constant at 0.4.

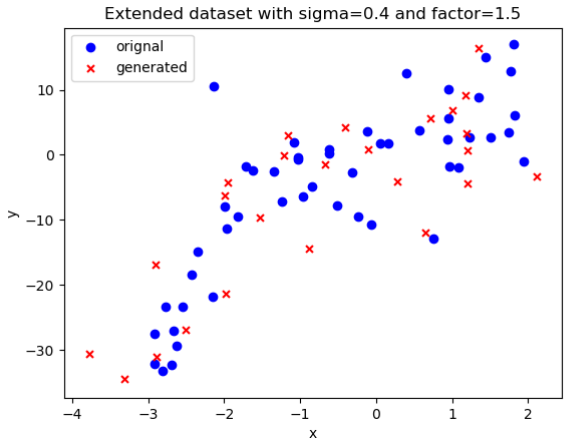

Figure 18: The extended dataset visualized for $\sigma = 0.4$ and $f = 1.5$.

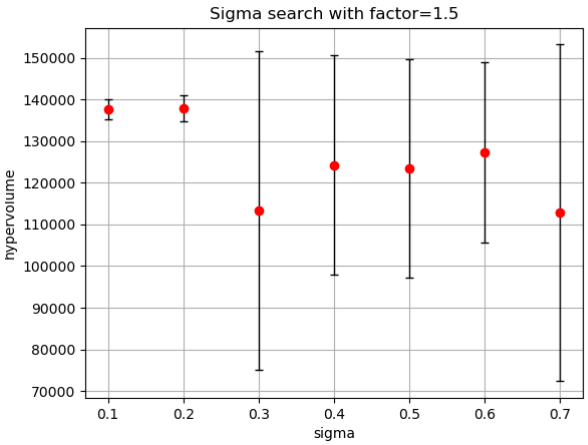

Figure 20: A search for the σ parameter for the improvement of NSGA2. f is kept constant at 1.5.

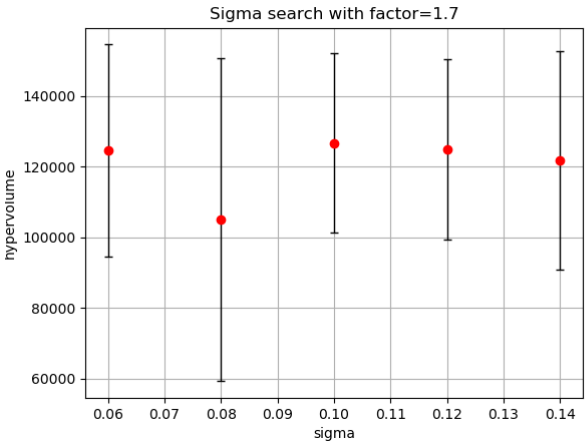

Figure 21: A search for the σ parameter for the improvement of NSGA2. f is kept constant at 1.7. The evaluations budget is set to 50000 and the settings are repeated 20 times.

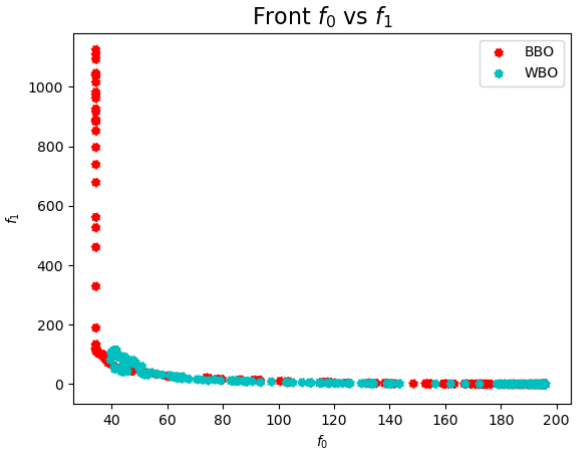

Figure 22: The front resulting in the highest hypervolume found by the WBO using $\sigma = 0.1$ and $f = 1.7$, compared with the best front found by the BBO.

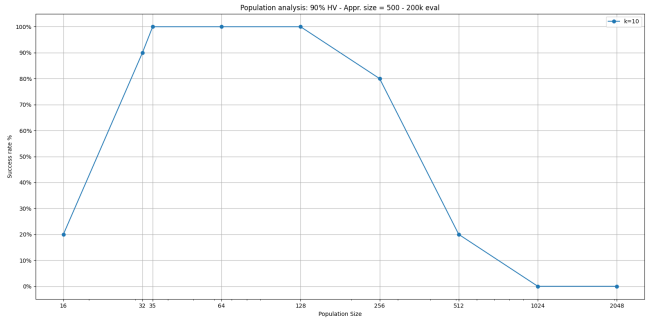

Figure 23: MAMaLGaM Success rate for different population sizes.

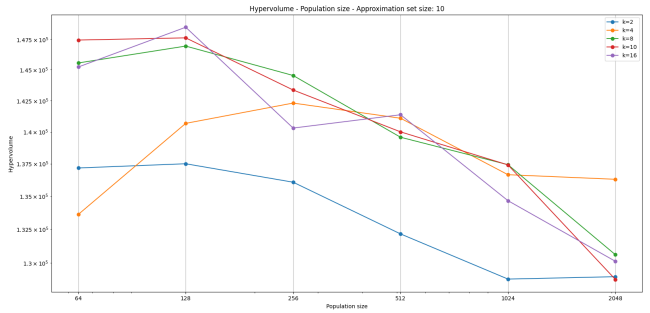

Figure 24: MAMaLGaM Approximation set size analysis among different problem dimensions and population sizes - Set size = 10.

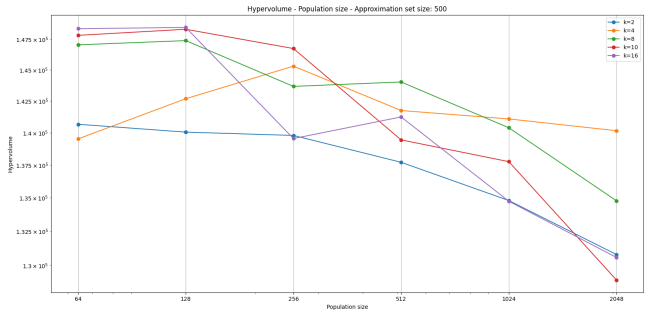

Figure 25: MAMaLGaM Approximation set size analysis among different problem dimensions and population sizes - Set size = 500.

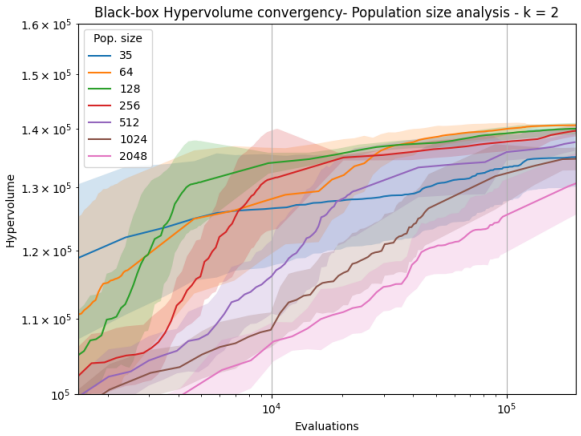

Figure 26

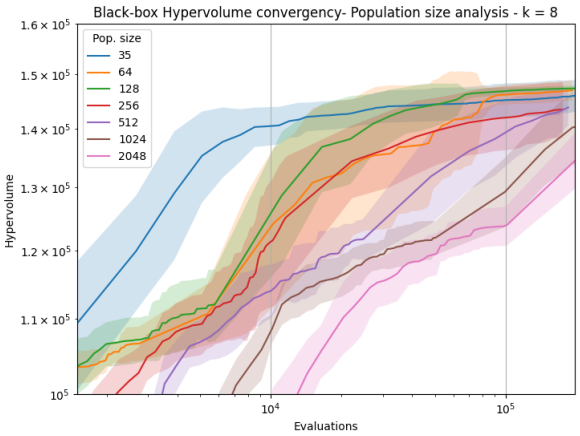

Figure 28

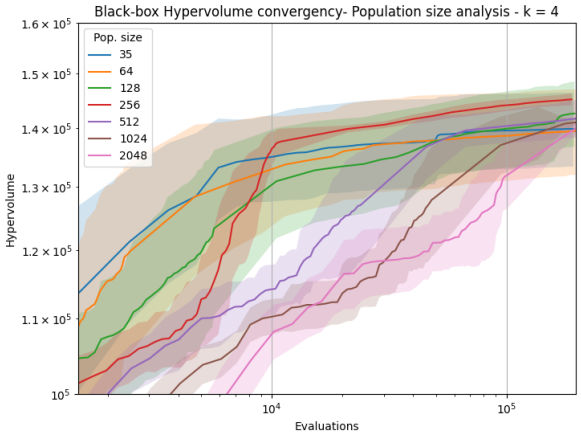

Figure 27

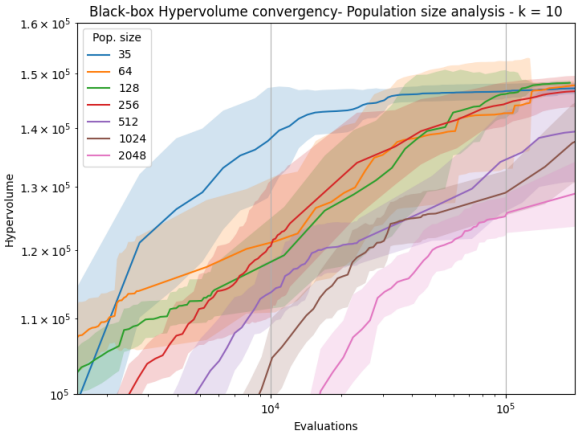

Figure 29

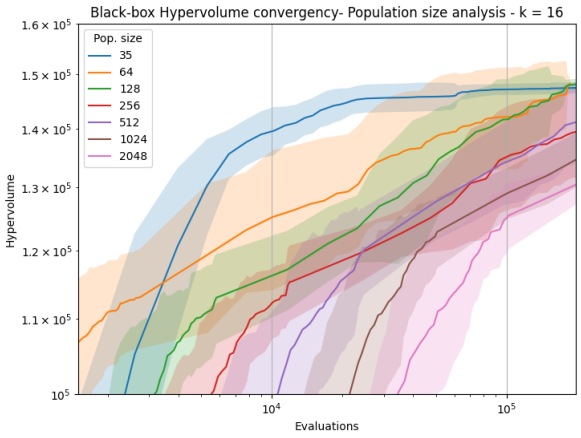

Figure 30

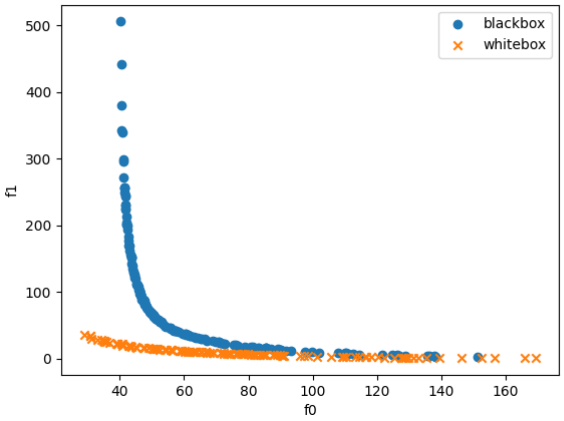

Figure 32

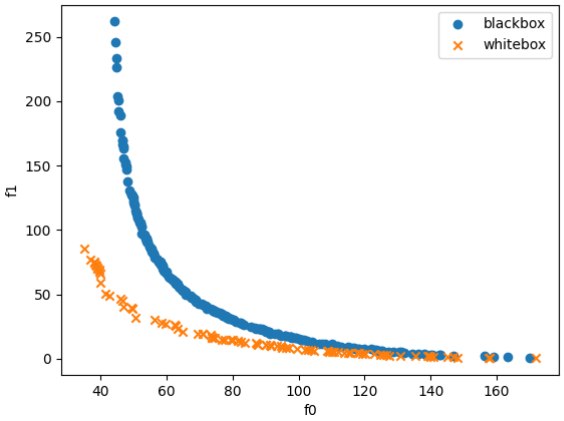

Figure 33

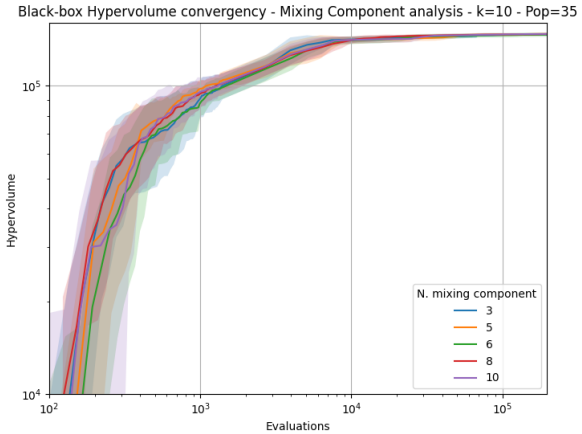

Figure 31

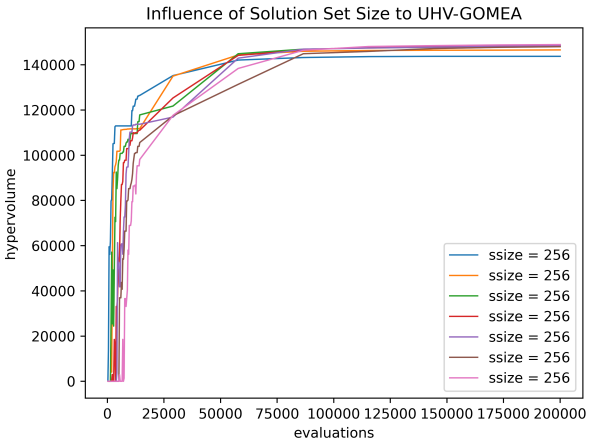

Figure 34: Hypervolume vs number of evaluations for different K

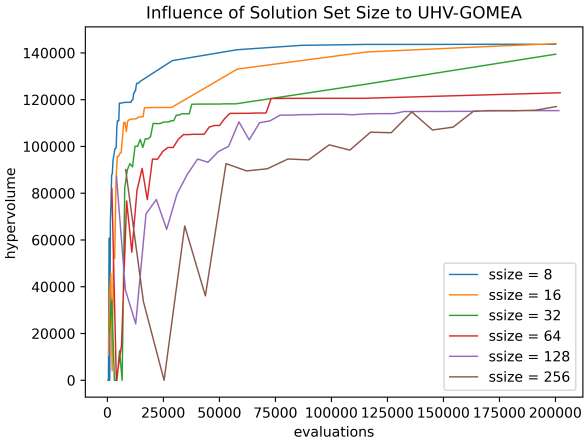

Figure 35: Hypervolume vs number of evaluations for for different solution set sizes

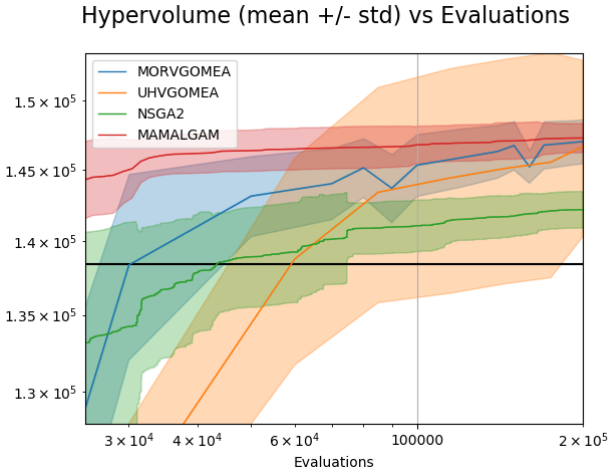

Figure 43: Convergence of MO-RV-GOMEA, UHV-GOMEA, NSGA2, MAMALGAM in black-box setting, where $K = 10$ and $|A| = 500$. The shaded areas show the standard deviations.

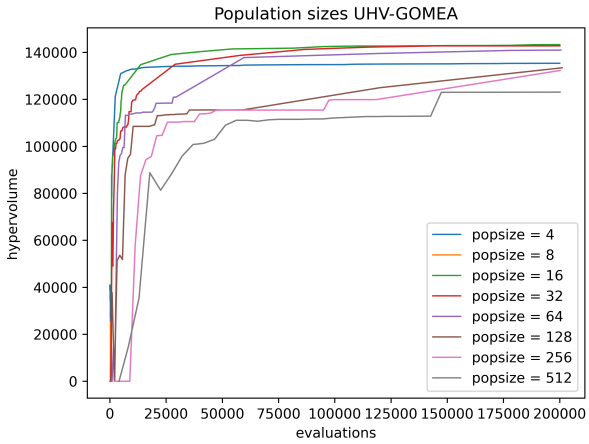

Figure 36: Hypervolume vs number of evaluations for different population sizes

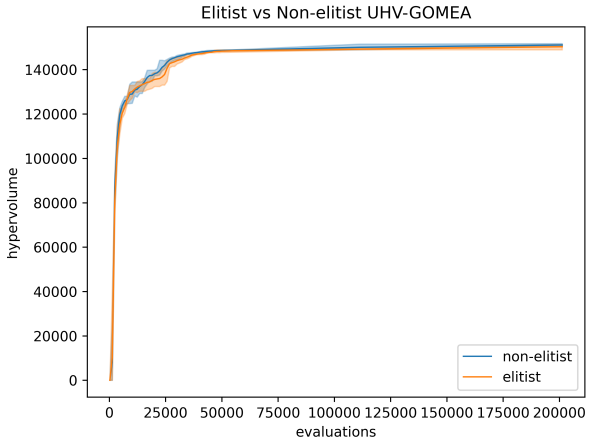

Figure 37: Hypervolume vs number of evaluations for elitist and non-elitist runs for white box optimization

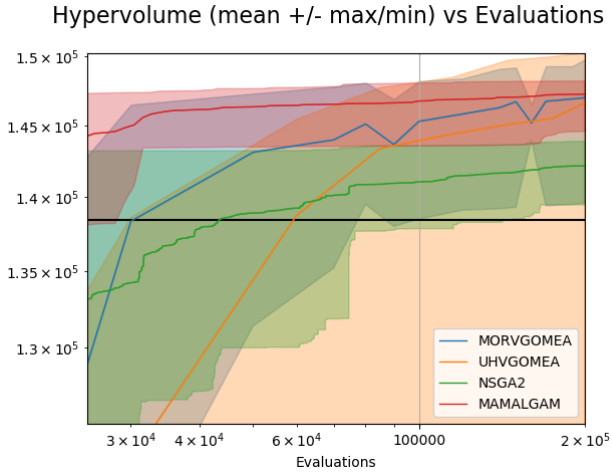

Figure 44: Convergence of MO-RV-GOMEA, UHV-GOMEA, NSGA2, MAMALGAM in black-box setting, where $K = 10$ and $|A| = 500$. The shaded areas show the minimum and maximum found hypervolume.

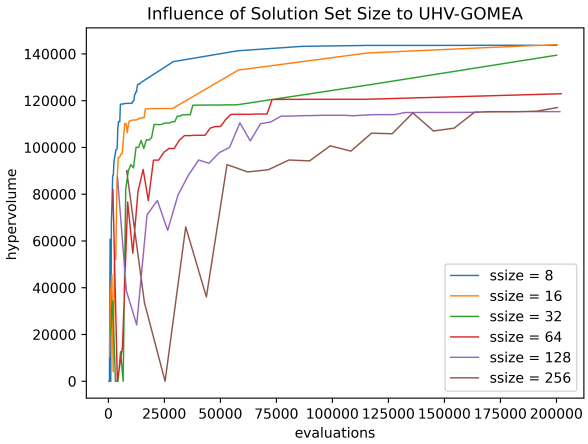

Figure 40: Hypervolume vs number of evaluations for for different solution set sizes

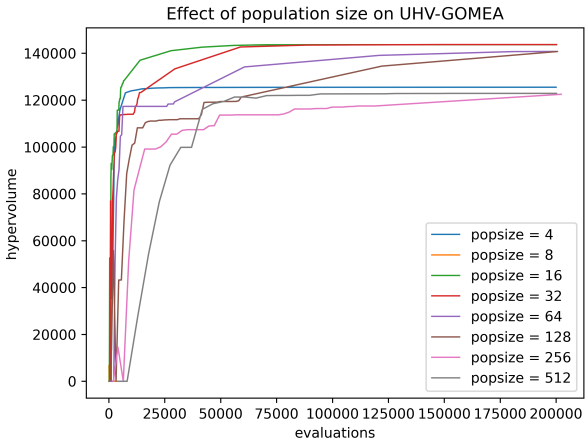

Figure 41: Hypervolume vs number of evaluations for different population sizes

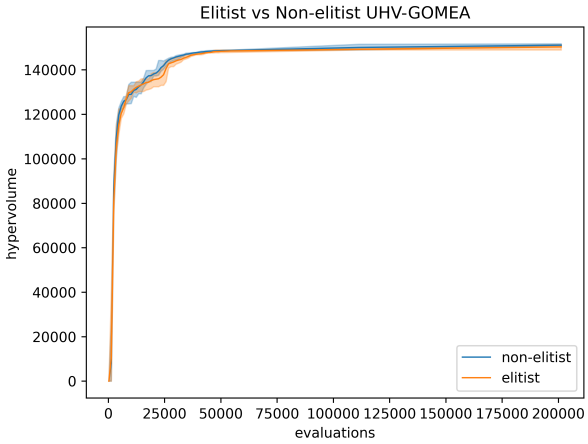

Figure 42: Hypervolume vs number of evaluations for elitist and non-elitist runs

	hypervolume	
	mean	std
BBO	142168.56	1272.21
WBO	138993.76	3337.46

Table 2: The results of the BBO and WBO for NSGA2, both using the best parameters that were found.

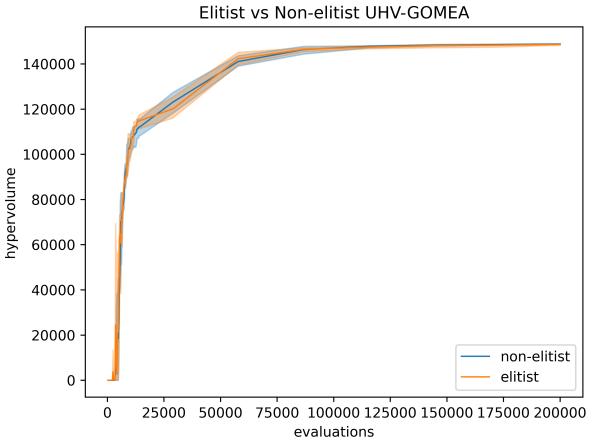

Figure 38: Hypervolume vs number of evaluations for elitist and non-elitist runs for black box optimization

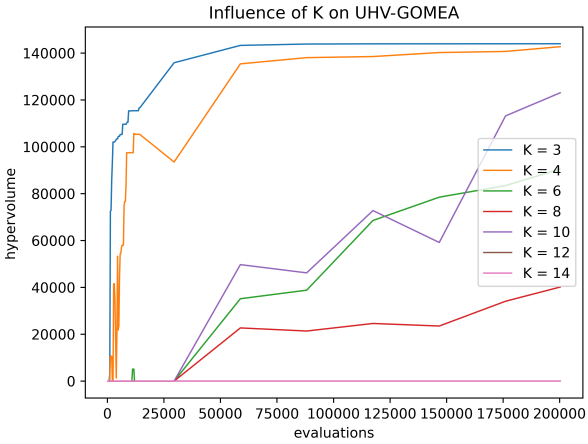

Figure 39: Hypervolume vs number of evaluations for different K

		hypervolume	
η_c	η_m	mean	std
1	20	143428.74	1029.12
20	2	141490.24	1935.99
1	2	142828.02	1331.20

Table 1: Results of the hyperparameter search for the crossover and mutation parameters for NSGA2.