Using density of training data to improve evolutionary algorithms with approximative fitness functions

Christina Plump*
*Cyber-Physical Systems
DFKI GmbH
Bremen, Germany
christina.plump@dfki.de

Bernhard J. Berger†
†Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
bernhard.berger@tuhh.de

Rolf Drechsler*‡
‡Institute of Computer Science
University of Bremen
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Evolutionary algorithms are a well-known optimisation technique, especially for non-convex, multi-modal optimisation problems. Their capability of adjusting to different search spaces and tasks by choosing the suitable encoding and operators has led to their widespread use in various application domains. However, application domains sometimes come with difficulties like fitness functions that can not be evaluated or not more than a few times. In these situations, surrogate functions or approximative fitness functions allow the evolutionary algorithm to work despite this complication. Still, using approximative fitness functions comes with a price: The fitness value is no longer correct for every individual, and the algorithm can not know which value to trust. However, statistical methods yield knowledge about the preciseness of the approximation. We propose using this knowledge to adapt the fitness value to ease the effects of the approximative nature. We choose to use the information given in the density of the training data, which has computational merits over the use of other techniques like cross-validation or prediction intervals. We evaluate our method on four well-known benchmark functions and achieve good optimisation success and computation time results.**

I. Introduction

Optimisation is an essential task in many areas. We see it in many application areas: Economics, logistics, disaster planning, scheduling problems, placement problems, e.g. in chip design [1], and the list goes on and on. Additionally, optimisation comes in many different forms: There are binary, integer, and real-valued optimisation problems but also optimisations for processes, programs, and strategies. Optimisation problems can have single or multiple objectives; they may be constrained or unconstrained. Furthermore, they can be linear or quadratic, convex or non-convex. Researchers or practitioners choose different techniques depending on the optimisation problem at hand. The most straightforward ones are deterministic solutions. However, whenever the problem becomes multi-modal, non-convex or very distorted, the choice usually defaults to heuristics, i.e. approaches with a stochastical component that do not necessarily yield the correct results. Evolutionary algorithms have long been a thriving, very flexible approach in this area and are one of the more dominant representatives of bio-inspired methods. The possibility to adjust almost every feature of an evolutionary algorithm makes them widely applicable and, at the same time, a challenge for every applicant. Small changes in features like parameters for chosen operators can considerably impact the quality of the proposed solution. Additionally, the chosen encoding is a rather intricate feature that needs lots of experience if the problem at hand is not a standard numeric one. Nonetheless, many successful solutions have been found, and quite a lot of the theoretical behaviour of evolutionary algorithms is understood well.

Given an optimisation problem, despite all the abovementioned differences they might have, one usually expects one essential part to be present and well-defined: The optimisation function. Unfortunately, many real-world applications do not directly yield an optimisation function that allows the researcher to use it as the fitness function. This inapplicability could — for one — stem from the complexity of the computation of the optimisation problem, i.e. one computation might take several days, e.g. complex numerical equations. Hence, a surrogate function is trained and used for a significant part of the optimisation process. A second reason for the inapplicability may be that the actual function is unknown, and its behaviour is only known through data possibly achieved by experiments. Today, companies and researchers try to find many explanations or advantages by analysing data, resulting in approximative functions (using machine learning techniques) that are part of an optimisation task.

However, applying an optimisation problem to an estimated function (estimating the actual fitness function with means of training data) has its difficulties. An estimated function is seldom perfect, i.e. it deviates from the original function in some input values. The estimated function may compute a worse fitness for a search candidate in the evolutionary algorithm's population than the actual function or the other way around. In both cases, the population of the evolutionary algorithm may drift away from a favourable distribution — not due to a poor design of the algorithm but simply because it is what the estimated function dictates. Additionally, the estimation does not need to be equally precise (or imprecise),

i.e. some regions of the estimation may be closer to the original function than others.

Some researchers have proposed including statistical knowledge about the estimation in the fitness function or the evolutionary algorithm to deal with the disadvantages of using an estimated function. We follow this path and propose to use the distribution of training data as a quality measure for the preciseness of the estimation. This quality measure can then be used to adapt the estimated fitness values. The main idea is that a high density is related to a better estimation (in general). The more training data one has for a specific area, the better the estimation is. Hence, using the information from the density, we can estimate the likeliness of whether a computed fitness value (from the approximation) is the correct fitness value. Then, we adjust the fitness such that fitness values which are more likely to be correct, remain unchanged, and fitness values, which are more likely to be incorrect, are balanced to smooth out extreme values. The balancing part decreases high fitness values and increases low fitness values. The goal is to reduce the influence of potentially wrong values on the evolution of the population.

We evaluate our approach using four benchmark functions. To model the situation of an estimated fitness function, we generated noised training data from those benchmark functions and trained estimated fitness functions of different degrees of precision. Then, we compare the results of our approach to the traditional one using the estimated function and find that, on average, our approach leads to an improvement over using the estimated function. We additionally compare our results to an approach that uses prediction intervals in terms of success in finding an optimum and the necessary computation time. Our approach surpasses the prediction interval approach and is significantly faster.

This work is structured in a usual manner: The following section introduces standard approaches for surrogate-based evolutionary computation and quantitive measures on estimation quality. Section 3 describes our approach and the proposed adjustment of a fitness function, whereas Section 4 explains how we implemented our approach. Section 5 reports on our evaluation and its results, and finally, Section 6 concludes this paper.

II. Background

This section deals with the motivation and some related work for this topic.

A. Genetic Algorithms and approximative fitness functions

Genetic algorithms are a vast research area, even if they are employed in a standard situation—clean data, precise fitness function, trivial encoding (if this exists). However, the number of problems increases when these standard attributes get more complicated. The genetic algorithm can not compute the true fitness function in many real-world applications. This incapability can be due to several reasons: First, the fitness function may be hard to calculate, making it impossible to compute it as many times as necessary for the evolutionary algorithm. Second, the fitness function may be subjective, i.e. evaluated by a person, and therefore non-deterministic and only available for a given amount of evaluations. Third, the setup may not define the fitness function given but only training data that may induce a fitness function. Researchers employ different techniques in these cases (see [2] and [3] for surveys on the topic). However, they have one thing in common: A surrogate (or approximative) function replaces the fitness function used for the evolutionary algorithm. Hence, we obtain the following situation, where $\hat{f}$ symbolises the estimated function, and f the original function:

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x}) \tag{1}$$

When the fitness function is only hard to compute, e.g. with complex differential equations that may take days to compute, it is at least possible to obtain some accurate results. Hence, it is possible to validate the approximate fitness values every now and then. This is still an open research field despite many good results that have been obtained over the years. The usual technique is to use a surrogate function that estimates the true fitness function(see, e.g. [4]). However, at certain points during the algorithm, the true fitness function is used to obtain exact values. The timing of these time points nevertheless varies from application case to application case. However, when the fitness is actually unknown, there is no possibility to include some true references. It may be possible to somehow validate the results with experiments, nevertheless, depending of the cost of this experiment, there should not be too many wrong results. Additionally, it is impractical to include the experiment while the evolutionary algorithm runs. Therefore, it could only be used to validate final results. In this case, many researchers have proposed including statistical knowledge to their approximative fitness function. The work of Plump et al. [5], for example, includes prediction intervals to adjust the final fitness function. They obtain reasonable results, although their method does seem to not work well on multi-modal functions. However, from their evaluation it is unclear, if this is rooted in their approach or the setup of their evaluation. They model the approximative situation by generating data and estimating an approximative function through support vector regression. Support vector regression does not necessarily perform smoothly on multi-modal functions which may disadvantage their result overproportionately. Nevertheless, their technique has one important drawback—their computation time for the prediction interval in their online phase is quadratic in the number of training data (or rather the support vectors in their machine-learning model). Which leads to an unfortunate situation: Less training data is favourable for computation time, more training data helps with the preciseness of the approximative fitness function.

B. Estimation quality

There are several statistical methods to describe (or estimate) the preciseness of an estimation. These are often called goodness-of-fit measures. However, not only the measure itself but also the process of evaluating is important to avoid pitfalls

like estimation or selection biases. We therefore define the standard MSE, discuss two evaluation processes and finally present a point-based evaluation measure contrasting the usual overall measures.

Please, assume for the rest of the paper unless stated otherwise, that we have m training data in n dimensions, mapping to one dimension, i.e. $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)$, where $\boldsymbol{x}_j = (x_{j1}, x_{j2}, ..., x_{jn})$. Furthermore, $y_j = f(\boldsymbol{x}_j)$. We denote the estimated function with $\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon(\boldsymbol{x})$. ϵ is the mistake the estimation makes. The smaller ϵ is, the more precise the estimation function is. Quality measure for estimations therefore are usually based on ϵ. Most times, the computed quality measure is the mean squared error (MSE):

$$MSE(T, \hat{f}_T) = \frac{1}{m} \sum_{j=1}^{m} (\hat{f}_T(T_{j,x}) - T_{j,y})^2 \tag{2}$$

1) Three Way Holdout: The three way holdout divides the training data into three different sets: One, for training, one for validation, one for testing. Using the training data, different models are trained and validated on the validation data set. Depending on the goodness of each of the models on the validation data, the final candidate is chosen. Subsequently, this candidate is evaluated on the testing set to estimate its quality. It is important to use a different set for the final evaluation to not fall victim to a selection and estimation bias.

2) Cross-Validation: Cross-Validation is a technique to use, when there is not enough training data available for a three-way holdout. The main idea is to split the training data set into k chunks. Then, $k-1$ chunks are used for training, and the kth chunk is used for evaluating ϵ. This is repeated until every chunk once served as the testing set. All results are averaged and their standard deviation is reported. Again, depending on the final results, a model is chosen, and its cross-validation is reported as quality estimation [6].

3) Prediction Interval: Prediction intervals differ from both techniques above in that they report a quality measure for every point in the domain rather than for the entire model. A prediction interval specifies the range that includes the true value $f(\boldsymbol{x})$ with a given confidence. It basically works as a confidence interval, however, it includes a bias to account for the fact that it considers single data points and not the average of a population.

$$\mathscr{I}(\boldsymbol{x}) = \left[\hat{f}(\boldsymbol{x}) \pm z_{1-\frac{\alpha}{2}} \sqrt{SE + bias(\boldsymbol{x})}\right] \tag{3}$$

Usually, prediction interval computation is rather computation intensive (usage of Bootstrapping). However, Brabanter et al. developed a technique that reduces the online computation significantly [7].

III. Methodology

The following section will motivate and describe our methodology. It is divided into four subsections, first we will state the problem, second, present our fitness adjustment methodology, third discuss configuration possibilities and fourth, justify the usage of density values to use for the quality measure.

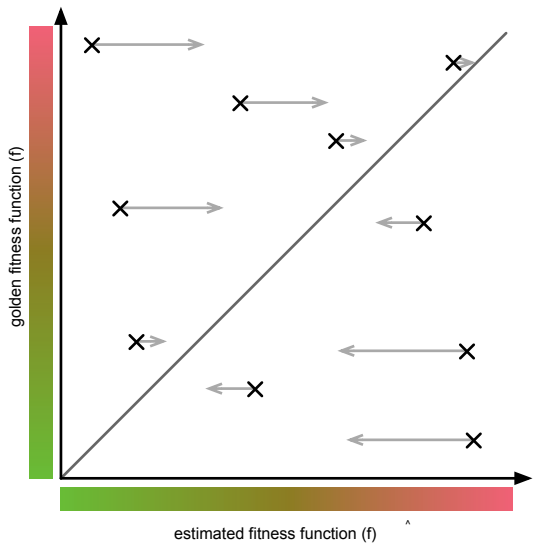


Fig. 1. Depiction of approximative fitness values and golden fitness values and their prospective change.

A. Problem Description

Please assume the following situation: f is the function to be minimised (recap from Section 2: $f : \boldsymbol{X} \to Y$, with $dim(Y) = 1$), i.e. a solution $\boldsymbol{x}^* \in \boldsymbol{X}$ to be found, such that $f(\boldsymbol{x}^*) < f(\boldsymbol{x})\ \forall \boldsymbol{x} \in \boldsymbol{X}$. We assume $dim(Y) = 1$ because we restrict ourselves to single-objective optimisation. Furthermore, this function f is unknown. However, training data T is present, and an estimation has been computed, namely, $\hat{f}_T$. The minimisation is supposed to be done using an evolutionary algorithm, i.e. f is the golden fitness function, while $\hat{f}_T$ has to be used as an approximative surrogate.

Figure 1 depicts a comparison of golden and approximative fitness values. The x-axis shows the approximative values, the y-axis the golden ones. If all points lay on the bisector, the estimation was perfect. The closer the points are to the bisector, the smaller the error. The shades in green, yellow and red represent the interpretation of both fitness value scales. A point in a green region will probably survive onto the next generation, while a point in a red region will not. It is now easy to see which points pose a problem and which do not. Of course, points on the bisector are perfect. Points close to it are unproblematic. Nevertheless, points in the upper left and lower right corner are problematic. Points in the upper left corner have a small approximative value; thus, they will most likely stay in the population, although they have a high golden fitness value; hence, they would have been removed from the population with a relatively high probability. Points in the lower right corner face the opposite problem: They are *fit* regarding the golden fitness function, have an influence on the next generation to reproduction and most likely stay in the population, only—their approximative fitness value is high, and therefore, in reality, they will possibly not survive.

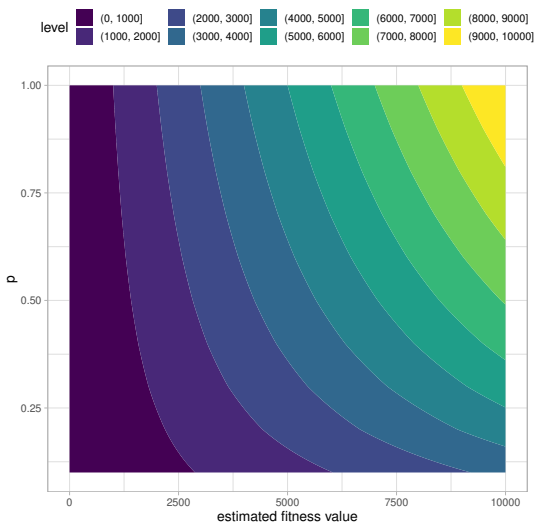


Fig. 2. Depiction of the adjusted fitness dependent on the approximative fitness and the probability

B. Adjustment of fitness

Adjusting the approximative fitness should now move points on the right of the bisector to the left and vice versa. Nevertheless, points that are already well estimated should not be moved as far as poorly estimated points. Figure 1 visualises this adjustment, where the length of the arrows symbolises the degree of the shift.

The above adjustment proposal can be reformulated in terms to allow the definition of an formalisation for f:

1) If the estimation is well, the adjusted fitness function should basically equal the approximative fitness.
2) If the estimation is poor, the following should hold:
   a) High fitness values are decreased
   b) Low fitness values are increased.

Assume that $p \in (0, 1]$ represents the precision of the estimation, i.e. $p = 1$ means the estimation is perfect ($\hat{f}_T = f$), $p = 0$ means the estimation is completely off.

Taking all of the above mentioned points into account, we propose the following adjustment (see detailed explanation below):

$$\tilde{f}_T = \hat{f}_T \cdot \sqrt{p} + \frac{1-p}{p^2} \tag{4}$$

This adjustment may seem arbitrary at first, but a closer look reveals its structure: First, the adjustment is in the form of a linear equation: It has slope $\sqrt{p}$ and step $\frac{1-p}{p^2}$. If $p = 1$, the step becomes 0, and the slope 1, i.e. we achieve identity of $\tilde{f}_T$ and $\hat{f}$ for a perfect estimation. With decreasing p, the significance of the step is dependent on the size of $\hat{f}$: The larger $\hat{f}$, the more significant the effect of the slope factor in decreasing it. Hence, for high fitness values, we achieve a decrease (with the exception of very small values of p—but we will turn to this later). However, if $\hat{f}$ is small, the size of $\frac{1-p}{p^2}$ has the higher impact, so the value will be increased. It can be shown that (excluding very small values of p, depending on the given range of fitness values, Equation 4 fulfills the criteria set up above. As the mathematical proof will take up too much space, we visualise Equation 4 in Figure 2. It shows the adjusted fitness value as color coded level, depending on the approximative fitness value and the value p. The evenly spaced contour lines at the upper end of figure show the linear behaviour - there $p = 1$ and therefore the adapted fitness value (shown through the colouring) equals the estimated value. It also shows the increase in step size for very small values of p.

The switch between the increasing and decreasing effect highly depends on the occuring ranges of fitness values. The higher the potential fitness values, the higher the y-intercept needs to potentially be, to have a chance of counteracting the effect of the fitness value. So, we introduce some configuration parameter into our formula: The exponents can be adapted to the given domain problem at hand as shown in the following equation. Please note that we have now formally introduced the dependency on x which we motivated during the last paragraphs. Furthermore, the density-based fitness function is now dependent on the training data as well as both configuration parameters, for which $s > r > 1$ should hold.

$$\tilde{f}_{T,r,s}(x) = \hat{f}_T \cdot \sqrt{p_T^r(x)} + \frac{1 - p_T(x)}{p_T^s(x)} \tag{5}$$

The question remains of how to decide which exponents to use. This is, fortunately, where math comes in. The conditions, we originally defined for our adjustment, can be reformulated via the fraction $\tilde{f}_{T,r,s}(x)/\hat{f}_T$. This fraction should equal 1 for the case of high probabilities, it should be smaller than 1 for low values of p and high fitness values, and it should be higher than 1 for low values of p and low fitness values. Assuming $p_T(x) \neq 1$, we can derive the following:

$$\begin{aligned}
&\frac{\tilde{f}_{T,r,s}(x)}{\hat{f}_T} && > 1 \\
\Rightarrow\ &\hat{f}_T \cdot \sqrt{p_T^r(x)} \frac{1 - p_T(x)}{p_T^s(x)} && > \hat{f} \\
\Rightarrow\ &\frac{1 - p_T(x)}{p_T^s(x)} && > \hat{f}(1 - \sqrt{p_T^r(x)}) \\
\Rightarrow\ &\frac{1 - p_T(x)}{p_T^s(x) \cdot (1 - \sqrt{p_T^r(x)})} && > \hat{f}
\end{aligned}$$

Please note that the same reformulations hold for $<$ inequalities and the equality. Therefore, $\frac{1-p_T(x)}{p_T^s(x)\cdot(1-\sqrt{p_T^r(x)})} = \hat{f}$ is the boundary between decreasing fitness values and increasing fitness values.

C. Configuring the adjustment

Figure 3 visualises this boundary for exemplary values for r and s. The domain expert (or evolutionary algorithm expert) can now—factoring in his knowledge about the usual fitness values—choose those configuration values that he thinks fit the given situation best.

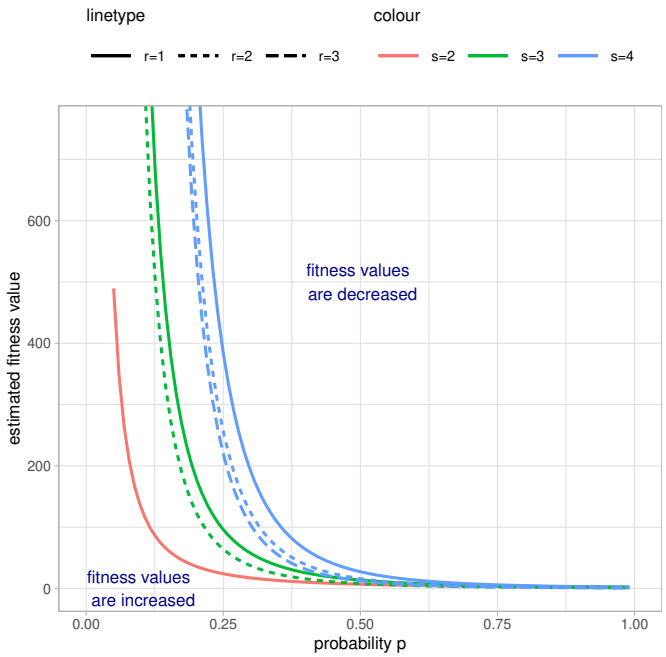


Fig. 3. Curves of the formula derived from Equation 5. Colors denote different values of s, the linetype different values of r. On the upper side of the curves, the adjusted fitness value will be decreased (improved), on the lower side of the curves increased (worsened).

D. *Influence of training data on estimation precision*

Up until now, we assumed a value $p \in (0, 1]$ to represent the precision of the estimation. However, we never clarified what value this might be. It is possible to use a fixed value, e.g. stemming from a Three-Way-Holdout estimation or a crossvalidation (see Section II). Nevertheless, Figure 1 shows that the value p needs to be dependent on the current individual, i.e. $p = p(x)$. Therefore, a fixed value is not sufficient. Another possibility is the usage of prediction intervals, they can be computed pointwise and their length might hold the information we need for our approach. However, to obtain a value in the range of $(0, 1]$ from a length seems not straightforward. Additionally, the computation of prediction intervals is quadratic in the size of training data and dependent on the used prediction algorithm (support vector regression, neural networks, gaussian processes). Instead, we propose using the probability of the current value given the distribution of the training data. The main assumption here is that the prediction will (in general) be better in regions where a lot of training data is present and will be worse in regions where little training data is present.

Estimating the probability density function (pdf) of the training data allows the computation of the probability of a *new* value given this density. This value will lie in the required interval and is dependent on the search space candidate $x \in X$.

Figure 4 visualises the principal idea that the precision of estimation usally depends on the distribution of training data for a two-dimensional case. The main part of the figure shows training and test data with their true values as well as their

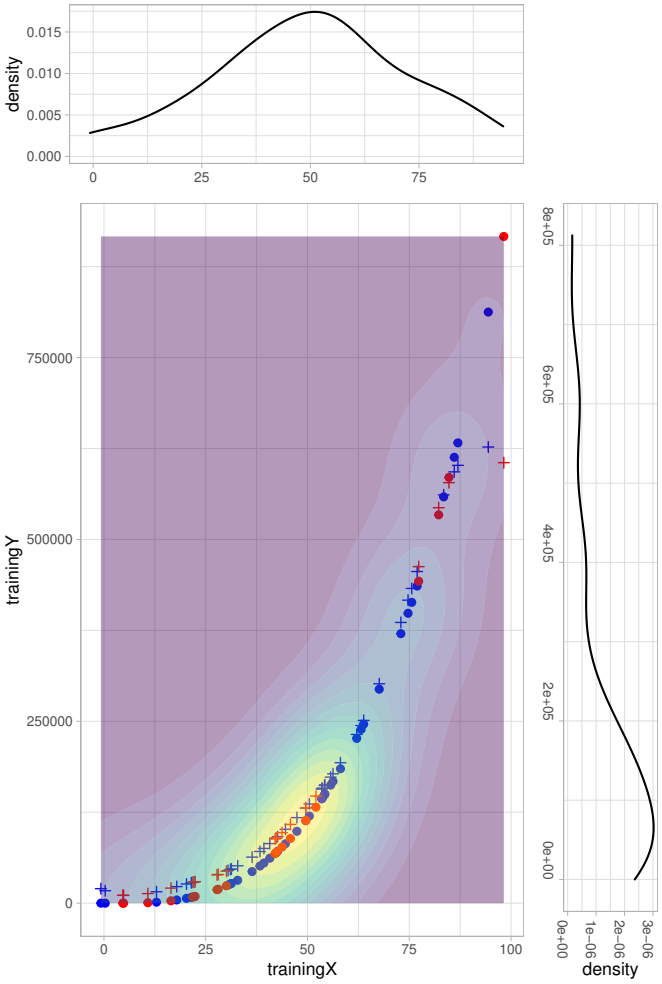


Fig. 4. An estimation example for a quadratic function with marginal density function for X and Y and the two-dimensional density in contouring colors. The points show the true function, while the crosses show the estimation. Colors red and blue separate the testing and training data.

estimated ones. The background shows a 2D-kernel density estimation. The top and right curves are the estimated marginal distributions, i.e. the distribution of X and the distribution of $Y = f(X)$. One can see that the estimation's precision is worse (distance of crosses to points) where the marginal distribution of Y is smaller. It is important to note that this will not be true in every case and there certainly can be setups constructed where this does not hold. The chances of this general idea to hold true are higher with correct data than with noised data. Another important point is the fact that these probabilities can turn out to be very small, in fact, a probability of 1 can almost never be achieved—as the training data needed to consist of only one data point (or many repetitions of this one point). The reader might get an impression of the values when looking at the axis of both marginal density graphs. Therefore, one more adjustment to this probability is necessary: We take the range of values into account. We assume here that a higher range of values will lead to smaller densities overall (keep in mind that this assumpition is a heuristic one,

it will certainly not always be true), and we therefore scale the probability with this density. That is, the same number of values has a higher chance of higher probabilities when defined upon a small range, as they do upon a large range. Therefore, we divide the computed probability by the length of the range of the data points considered. This leads to our final choice of $p(x) = \mathbb{P}(Y_T = \min \hat{f}(x)) \frac{\max Y - \min Y}{2\epsilon}, 1$, where ϵ is the boundary around x that is taken for computation of the probability based on the density. Usually, the minimum function should not be necessary, however, there may be distorted distributions that lead to this effect.

IV. Implementation

To evaluate our proposed approach, we created a Java-based implementation. We use SMILE, the statistical machine intelligence and learning platform, a programming library supporting support vector regression and kernel-based density estimation [8]. We use SMILE to calculate the support vector regression and to estimate the density in the learning phase. Furthermore, we use Jenetics, a framework for genetic algorithm, evolutionary algorithm, genetic programming, and multi-objective optimisation for implementing the evolutionary algorithm [9]. In Jenetics, it is possible to register a custom-based fitness function. We used this mechanism to add our modified fitness calculation. This extended calculation usees SMILE to apply the learned regression function to an individual and then adjusts the fitness value using SMILE again.

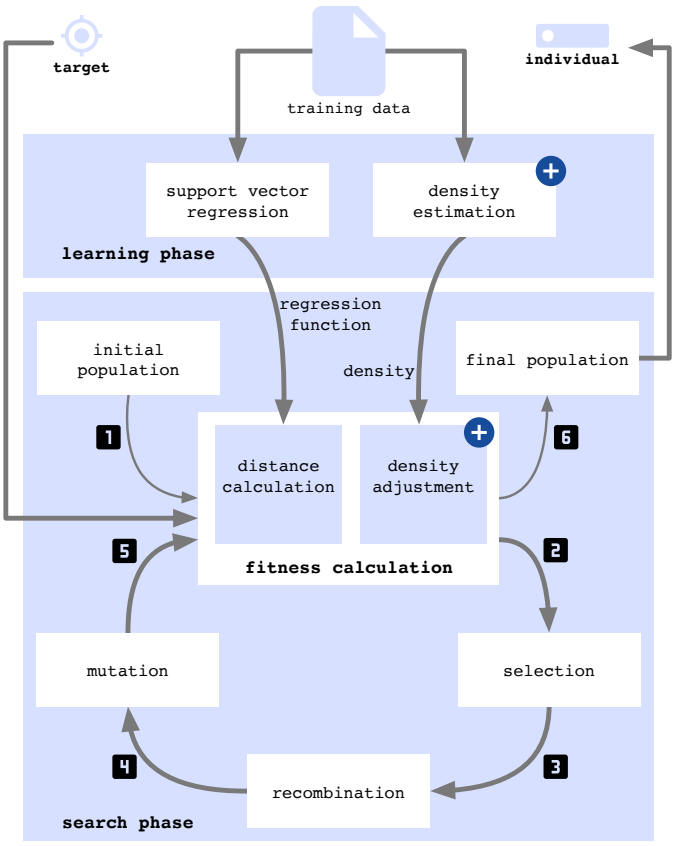


Fig. 5. Implementation overview

Figure 5 shows an overview of our approach's implementation. First, the learning phase learns a regression function based on existing training points. Besides, it estimates the training's data density using a Gaussian kernel approach. This step executes before the actual search phase runs. Second, the search phase works as a usual evolutionary algorithm does. It starts with an initial population for which the Jenetics framework calculates the fitness. The fitness calculation first maps an individual using the learned regression and calculates the distance to the searched target properties. We implemented the density adjustment to modify the distance according to the estimated probability using Equation 4 to support our approach. Afterwards, the normal selection, recombination, and mutation process take place. The individual with the best fitness value is selected from the final individual generation when the evolutionary algorithm stops. The adaptions made to implement our approach are highlighted using a plus sign.

V. Evaluation

This section presents the empirical evaluation of our approach. The evaluation was conducted on a commercially available laptop, running on an Intel i7-8565U CPU which operates at 1.80 GHz. The system has 24G of memory available.

In total, we work on three research questions:

RQ 1: Does our approach significantly differ from the standard approximative approach in terms of best individuals as result of an evolutionary optimisation?

RQ 2: How does our approach compare to the use of prediction intervals in terms of success of the optimisation process?

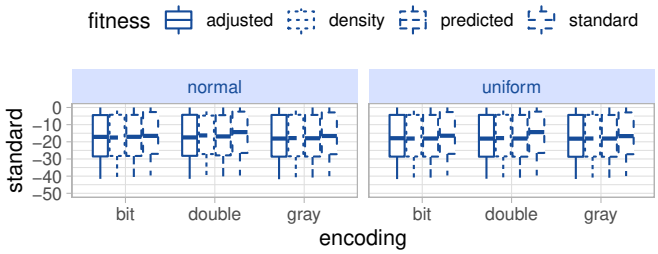
RQ 3: How does our approach compare to the use of prediction intervals in terms of computation time?

We will first describe the evaluation setup as well as the statistical evaluation process. Subsequently, we will present the results of our evaluation and finally, discuss them. Please note that while we reproduced the results of [5], evolutionary algorithms have a stochastic component, therefore, the results presented here may differ slightly from their published ones.

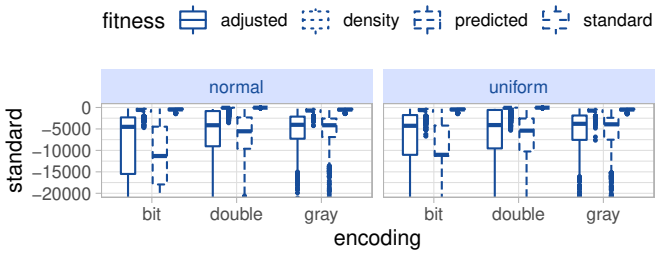
A. Setup

As has been mentioned in Section IV, we simulate the problem of an approximative fitness function by sampling training data from a distribution, computing correct values based on the benchmark function, noising them and finally estimating the functions using a support vector regression with different kernels. This result is used as approximative fitness function. That way, we are capable of comparing our results to a golden truth (the real fitness function) because the benchmark function are our fitness functions. Additionally, we include so called targets y^*. They adapt the original fitness function to $f_{y^*}(x) = |f(x) - y^*|$. This is done to give some variation to the benchmark functions.

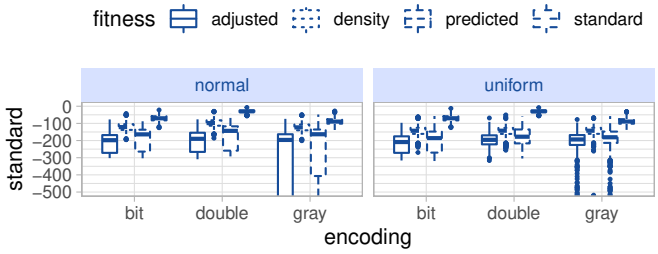
We vary several components of the setup for our evaluation. Every distinct combination is called one setup in the following. One setup is run 10 times, with a population of 100 individuals



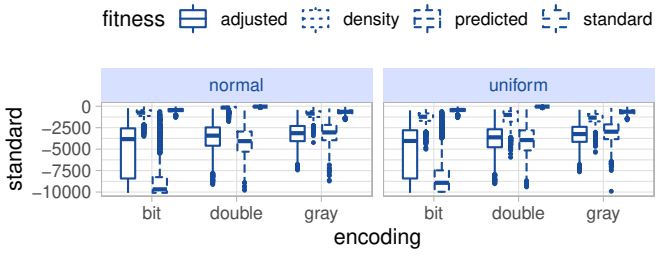
(a) Results of Ackley-function with $n = 10$, $m = 1000$



(b) Results of Rosenbrock-function with $n = 10$, $m = 1000$



(c) Results of Rastrigin-function with $n = 10$, $m = 1000$



(d) Results of WeightedSphere-function with $n = 10$, $m = 1000$

Fig. 6. Evaluation results, results of *adjusted* have been reproduced according to Plump et al. [5]

and for 100 generations, to smooth the stochastic component of evolutionary algorithms. The varied components are the following:

- **Benchmark function:** Ackley Function, Rosenbrock Function, Rastrigin Function, Weighted Sphere Function, each with $n = 10$.
- **Distribution of Training Data:** Multidimensional Normal Distribution, Multidimensional Uniform Distribution
- **Goodness-of-fit of estimation:** Good, average, bad (classified based on 10-fold crossvalidation)
- **Evolutionary Algorithm**[1]**:**
  - **Encoding:** Bit-Encoding, Gray-Encoding, Real-Valued-Encoding
  - **Selector:** elitism , stochastic
  - **Recombinator:** uniform crossover (Gray), single point crossover (Bit), line crossover (real-valued)
  - **Mutator:** Swap Mutator (Bit, Gray), Gaussian Mutator(Real-Valued)
- **Targets:** 50 uniformly chosen values from the respective ranges of the benchmark functions.

All in all, we have $144 \cdot 50 = 7200$ distinct setups. We run each setup once with the golden fitness function $(f_{gold}(x) = |f(x) - y^*|)$, once with the estimated fitness function $\left(f_{app}(x) = \left|\hat{f}(x) - y^*\right|\right)$, and once with our density-adjusted approach $\left(f_{dens}(x) = \left|\tilde{f}(x) - y^*\right|\right)$.

[1]The detailed parameters are as follows: for the elistism selector, we have a survivor proportion of 0.4, and an off spring proportion of 0.2. For the recombination operators, we have the following: uniform crossover with a crossover probability of 0.3, a swap probability of 0.4 and a threshold of 0.6, the single point crossover has a probability of 0.3 and a threshold of 0.6, the line crossover has a probability of 0.4 and again, a threshold of 0.6. For the mutation operators, we had a probability of 0.6, and a threshold of 0.6 for the swap mutator and the gaussian mutator was run with the same parameters.

B. Computation of presented results

As mentioned above, we repeat every setup ten times. Our evaluation is then mainly based on the comparison of the average fitness of the best individual. That is, for every setup S we compute: $avgBest(S) = \frac{1}{10}\sum_{i=1}^{10} f_{true}(\arg\min_{ind \in P} f_S(ind))$. It is important to note that the best individual is chosen based on the fitness function specified in the setup S, i.e. the approximative one, our modified one, or the one using the prediction intervals of [5]. However, the average value is computed using the true fitness value of that individual. This is necessary, because otherwise due to the adaptions in the fitness functions, their absolute values would be incomparable.

We use these golden fitness values of the best individuals to compare the different fitness approaches, mainly the approximative one (using the standard estimation) and our approach, i.e the density-based fitness. Later on, we will also compare our approach to the one of Plump et al. [5] who used prediction intervals to adapt their approximative fitness.

We compare each setup (that was run 10 times) with its counterpart with the changed fitness function to determine which fitness function performed better. To decide upon the winner of each of these contests, we use a hypothesis test to ensure statistical significance. We choose a one-sided hypothesis test (because one-sided comparison) for the comparison of means with different standard deviations, also known as Welsh's t-test[10]. This test fits our situation perfectly as we are indeed comparing means (mean of the best individuals over 10 runs) and have different standard deviation in the

population considered as contestants). In general, we run our hypothesis tests with a confidence niveau of 0.95 That is, we limit our error probability when deciding for the alternative hypothesis to 5%. We count the result of a hypothesis test as *win* for the density-approach, when we can reject the hypothesis that the approximative fitness function has a higher mean. Vice versa, we count it as *win* for the approximative approach, when the hypothesis that the density-based fitness function has a higher mean can be rejected. When the null-hypothesis cannot be rejected, we count the result as *borderline*. This follows the general idea of [11]. Furthermore, we store the p-value to analyse how significant the potential wins were.

We perform one hypothesis test for every setup, i.e. in the end, we obtain 7200 results for the comparison of the approximative fitness approach and our denisty-based fitness approach.

C. Results

Before presenting the hypothesis test results, Figure 6 gives an overview of all obtained results. The figure shows the results of all four benchmark-functions separately and additionally, divides the data according to the encoding and the distribution of the setup. The boxes depict the inner 50% of data points (i.e. from the first quartile to the third quartile), the middle line denotes the median - regarding the negative golden fitness value. Please remember, optimisation goal is zero, i.e. the higher the box, the better. For each depicted setup, we show all four optimisiation modi, i.e. what fitness function was used in the evolutionary algorithm. These are: adjusted (reproduced for comparison from Plump et al. [5]), density (our presented approach, $\tilde{f}$), predicted (the approximatve version, $\hat{f}$), and standard (the original fitness function, f). First, we notice what we expected: The standard approach outperforms all others, i.e. using the golden fitness function yields the best results. Second, however, we notice that our density-based approach is relatively close to the standard approch and obtains much better results than the predictive one. Third, we see, that the density-based approach outperforms the adjusted approach as well - except for the Ackley Function. In that case, all approaches are equally bad or good. It is interesting to note, however, that, similarly to the adjusted approach, the density-based approach works better on both unimodal functions (Rosenbrock and Weighted Sphere). The density-based approach comes even close to the standard fitness function in some setups. For the Rastrigin Function, a multimodal function, where the adjusted approach could not show its merit, our density-based approach still outperforms the predictive fitness function but it is obviously not as good as for the unimodal functions. The Ackley function, however, seems to not react at all to these different fitness approaches. One last observation: The distribution does not seem to have an effect on the ranking of the approaches but sometimes on the quantitative values. However, there is no clear tendency to be observed.

To underlie the above observations with some numbers, we performed Welsh's t-test as explained above. Please keep in mind that for hypothesis tests, we can only ever compare two approaches, i.e. we have one statement for the performance of our density-approach against the predictive setup, and one for the performance of our density-approach against the adjusted approach from [5]. For this primary comparison, we run the hypothesis test with confidence niveaus of 0.90 (called *significant*), 0.95(called *statistically significant*), 0.99(called *highly statistically significant*) to ensure there is no information left out through usage of confidence niveau of 0.95. Figure 7 shows the number of *wins* for the predictive, and the density-approach, as well as the borderline category (remember, these were the setup comparison where no approach was deemed a better than the other).

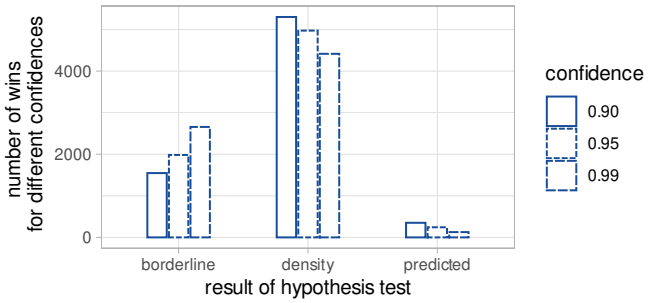


Fig. 7. Results of hypothesis test for predictive vs. density-based with confidences of 0.90, 0.95, 0.99. Height of bars shows number of *wins* for this category, borderline includes all setups for which no significant *win* was found. Total number of setups was 7200.

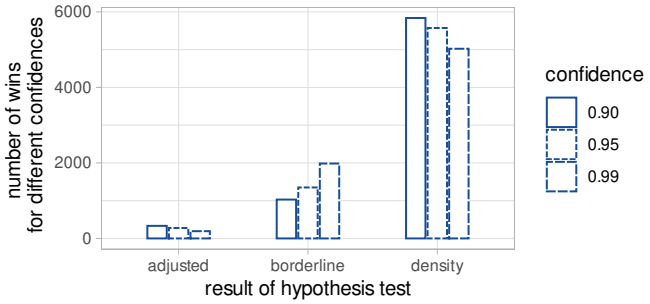


Fig. 8. Results of hypothesis test for adjusted vs. density-based with confidences of 0.90, 0.95, 0.99. Height of bars shows number of *wins* for this category, borderline includes all setups for which no significant *win* was found. Total number of setups was 7200.

The density-based approach *wins* between 4200 and 5200 hypothesis tests (of 7200) depending on the chosen confidence, while for 1800-2800 the results are borderline. These are — in all likelihood — the setups from the Ackley Function and some of the Rastrigin Funcion. Additionally, one can see that the borderline results increase with higher confidence which is by design as - of course - a smaller amount of wins is deemed highly significant (for any contestant) than significant. But, what is reassuring is that the decreases for the denisty approach and the predicted are relatively speeaking comparable. This means, there is distortion in the data depending on the confidence niveau. Figure 8 shows the same but for the

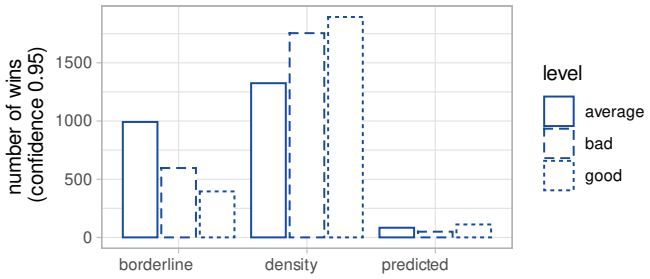


Fig. 9. Number of *wins* for the respective categories divided by the level of estimation precision.

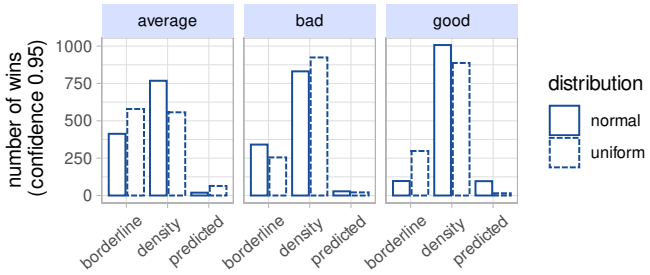


Fig. 10. Number of *wins* for the respective categories divided by the level of estimation precision and distribution of training data.

comparison with the adjusted method. The adjusted method gains more wins than the predictive version did originally, however, the density-based approach still outperforms it.

Additionally, we investigated the dependence of the hypothesis test results on the remaining properties of setups. That is, we investigated the influence of the precision of the given estimation (see Figure 9), the influence of the distribution together with the precision (see Figure 10), and the influence of encoding and selection mechanism (see Figure 11). In Figure 9, we see that still the density-based approach outperforms the others in every subcategory. Nevertheless, it performs weaker in combination with an average precision level. Here, the borderline category takes up more *wins*. It is also interesting that the good precision level has the least amount of *wins* in borderline. This suggests that for good estimations, either the density-based approach is better or the predictive (in comparison to the other levels) but there is less undecisiveness in the middle.

This impression manifests when looking at Figure 10: For the normal distribution and the good precision level, the predictive approach actually gains roughly as much *wins* as the borderline category, i.e. a significant number of borderline cases switched to the predictive category. Additionally, for the average level and uniform distribution, we find the worst performance of the density-based approach so far - there are more borderline cases then for density-based. It still outperforms the predictive fitness function, however, uniformly distributed training data with an average precision of estimation seems

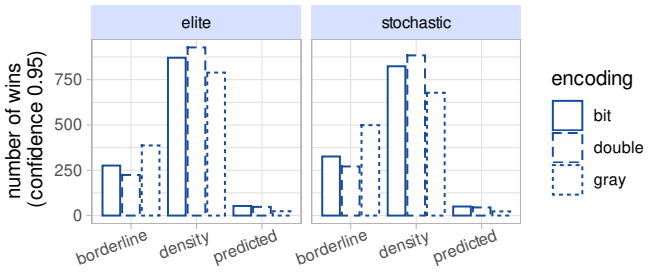


Fig. 11. Number of *wins* for the respective categories divided by the used encoding and selection mechanism.

to be the setup where our approach performs weakest. This is actually quite understandable, as the density will have only a small effect with uniformly distributed data as well as small values, and the average estimation works well without adjustment.

Finally, looking at Figure 11 we observe that the density-based approach has its merits especially when combined with an elitism approach, while it performs weaker with more stochastic approach. This may be because the stochastic approach already aims at the same goal as the density-based approach - keeping individuals in the population even when they do not have perfect fitness. Therefore, the density-based approach can not play out its strengths as much as it can with the elitism approach, which would kill all individuals but the best and possibly move away from the right search region.

Additionally, we analysed the computation time of our approach to show its applicability to real optimisation problems. Table I shows the computation time of the standard fitness function, the predictive, the density-based approach and the adjusted approach of [5]. We give the mean, the standard deviation, as well as minimum and maximum computation times. We see that the predicted and the standard optimisation types perform equally fast. Our approach roughly double the time needed for one setup, however is still only $0.5s$. The maximal duration were roughly two seconds. However, this increase seems small, when compared to the adjusted approach which takes roughly $9s$ for one setup. This difference in computation time is quite obvious, when looking at the necessary computation during the online-phase: The adjusted approach needs to perform matrix calculation with dimension of the size of the training data, whereas the density-based approach only searches its position in a sorted array and then computes the probability computation. The estimation process is done during the learning phase. Therefore, the density-based approach is roughly linear in the number of training data, while the adjusted approach is quadratic.

D. Discussion

With the above presented results, we can now answer our posed research questions. RQ 1 can be answered positively: Yes, our approach differs from the predictive approach, and significantly improves using the predictive approach. Only

TABLE I
COMPUTATION TIME EVALUATION FOR ALL FOUR OPTIMISATION MODI. ALL TIMES ARE IN MILLISECONDS.

optimisation type	mean	standard deviation	max	min
standard	250	157	1373	42
predicted	295	151	1370	64
density	459	143	1717	157
adjusted	9710	1877	61850	2932

for a multimodal and inseparable function (Ackley Function), we see no significant difference. This is a challenge to be investigated in further research. RQ 2 as well can be answered positively. Our approach also outperforms the adjusted approach which used prediction intervals. However, as we did not only change the information about the preciseness of the estimation (density vs. length of prediction interval) but also the way we incorporated this information into our density-based fitness function, it is unclear where this outperformance stems from – the usage of density or the different adjustment of the fitness function. It should be interesting to compare both approaches more thoroughly to obtain a more detailed view. RQ 3 also receives a positive answer: Whether theoretically or empirically, the density-based approach is less computation-intensive. However, we would like to point out that we only used one prediction technique (support vector regression) - so we could not investigate the behaviour of both approaches for different predictions techniques like neural networks and gaussian processes. Additionally, we used two different training data sets (one for the normal distribution, and one for the uniform distribution). It may be that this influenced the results because they randomly favoured our approach. It might be interesting to repeat this evaluation with more sets of training data from more distributions. All in all, however, we positively answered all our research questions and found that our presented approach outperforms the standard technique of simply using the predicted function as fitness function when the true fitness function is not available.

VI. CONCLUSION AND FURTHER RESEARCH

We address the problem of approximative fitness functions in evolutionary algorithms. The outcome of the evolutionary algorithm is highly dependent on the preciseness of the estimation of the fitness function. We propose to use the knowledge about the distribution of the training data to adapt the approximative fitness function. Our method is evaluated against four benchmark functions as well as a similar method which uses prediction intervals to adapt the fitness function. We obtain good results and are significantly faster than the similar method. For future work, we would like to explore our work on different machine learning techniques, e.g. neural networks, and extend our work to multi-dimensional optimisation problems.

REFERENCES

[1] S. Shirinzadeh, M. Soeken, D. Große, and R. Drechsler, "An adaptive prioritized ϵ-preferred evolutionary algorithm for approximate bdd optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1232–1239. [Online]. Available: https://doi.org/10.1145/3071178.3071281

[2] L. Shi and K. Rasheed, *A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms*, 01 2010, vol. 2, pp. 3–28.

[3] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, pp. 3–12, 10 2005.

[4] D.-P. Yu and Y.-H. Kim, "Predictability on performance of surrogate-assisted evolutionary algorithm according to problem dimension," 07 2019, pp. 91–92.

[5] C. Plump, B. J. Berger, and R. Drechsler, "Improving evolutionary algorithms by enhancing an approximative fitness function through prediction intervals," in *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021*. IEEE, 2021, pp. 127–135. [Online]. Available: https://doi.org/10.1109/CEC45853.2021.9504722

[6] T. Fushiki, "Estimation of prediction error by using k-fold cross-validation," *Statistics and Computing*, vol. 21, pp. 137–146, 2011.

[7] K. Brabanter, J. De Brabanter, J. Suykens, and B. De Moor, "Approximate confidence and prediction intervals for least squares support vector regression," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 22, pp. 110–20, 11 2010.

[8] H. Li, "Smile," https://haifengl.github.io, 2022.

[9] F. Wilhelmstötter, "Jenetics," https://jenetics.io, 2021.

[10] B. L. Welch, "The generalization of 'student's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947. [Online]. Available: http://www.jstor.org/stable/2332510

[11] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodologyfor comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, pp. 3–18, 2011.