Universiteit Leiden

Master Computer Science

Sequential Model-Based Optimisation using Symbolic Regression

Name:	Romme Knol
Student ID:	s3680800
Date:	30/06/2024
Specialisation:	Artificial Intelligence
1st supervisor:	Prof. dr. Thomas Bäck
2nd supervisor:	Dr. Niki van Stein

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Chapter 1

Introduction

This thesis investigates the usage of symbolic regression through genetic programming (SRGP) as part of a sequential model-based optimisation (SMBO) algorithm. Until now, the possibility of using an SRGP algorithm as a model in SMBO has not been investigated. Recent work has been done to evaluate and compare a large selection of modern SRGP algorithms and other machine learning (ML) methods by La Cava et al. [3], which indicates that SRGP algorithms can reach high degrees of model quality on a diverse selection of datasets. It also provides a good basis to determine which techniques can be considered state-of-the-art, allowing us to select the top-performing algorithm Operon [2], which uses genetic programming to generate mathematical expressions, to use in our research.

Sequential model-based optimisation is an approach to optimising functions or processes that are expensive to evaluate, whether in terms of computational power, time spent waiting, or labour. Examples include complex industrial simulations and large neural networks, for which the runtime or training time may be measured in hours or days. In each case, we want to leverage the knowledge gained from previous evaluations when choosing a new set of inputs or parameters for our next evaluation. By constructing a model after each evaluation and using it to make an informed choice for the next inputs, we can reduce the total number of evaluations required. In practice, reducing the number of evaluations reduces the amount of time, money, and labour required to optimise the objective function. In our research, we therefore choose the number of objective function evaluations as the metric by which we evaluate and compare SMBO algorithms. A popular and robust example of such algorithms is Efficient Global Optimisation (EGO) [12], which uses Gaussian processes as models.

Our aim in this thesis is to evaluate the suitability of SRGP algorithms as components in SMBO approaches. We test this suitability by using the state-of-the-art SRGP algorithm Operon to construct the models in a novel sequential model-based optimisation algorithm. We compare this new algorithm to the popular EGO algorithm to determine whether it is competitive. We formulate our primary research question as follows:

"Can an SMBO algorithm which uses an SRGP method to model the objective function find better optima than EGO on a set of test problems?".

We also identify two sub-questions.

- Is there a subset of problems on which SRGP performs especially well or poorly?

- In cases where the optima found by EGO and Operon are (close to) equal, is there a difference in the total number of objective function evaluations required to find these solutions?

This thesis is structured as follows. In chapter 2, we provide explanations of the concepts used in this thesis, most importantly SRGP and SMBO and provide examples of applications of these concepts. In chapter 3, we delve into the properties of the Operon algorithm. We focus on the properties that indicate whether Operon is a suitable choice for the modelling step in an SMBO algorithm. We look at the effect that the amount of available data has on the quality of the generated expressions and the (order of magnitude of the) time it takes to generate expressions. In chapter 4 we cover the full workings of our novel algorithm that uses Operon and describe the setup and results of an optimisation experiment in which we compare our approach to EGO. In chapter 5, we draw conclusions from our results regarding the suitability of Operon for SMBO. We also discuss possible ways to improve the algorithm and alternate design choices that could be made.

Chapter 2

Background

In this thesis, we investigate the intersection of two domains, symbolic regression (through genetic programming) and sequential model-based optimisation. Below, we will explain the core concepts of both domains, highlight practical applications, and explain choices in our selection of techniques, such as the selection of Operon as our preferred symbolic regression implementation. In section 2.1 we cover the fundamental workings and different modes and applications of symbolic regression, as well as the general strengths of Operon. In section 2.2 we explain the way sequential model-based optimisation works. We also give more details about EGO.

2.1 Symbolic Regression

The problem of symbolic regression can be defined formally as follows: given some dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, find a function f such that $y_i \approx f(x_i)$ for all i. In contrast with other regression methods such as linear regression, random forests, or neural networks, symbolic regression algorithms do not presuppose a full model architecture. For example, linear regression learns only coefficients of an assumed linear function and a neural network has a predefined architecture of neurons for which it learns parameters. In symbolic regression, on the other hand, we learn both the fundamental structure of the function, expressed in mathematical operators, and learn the parametrization of this function, in terms of coefficients that may apply to terms in the function.

Symbolic regression (SR) arose as an application of the more general algorithmic family of genetic programming (GP), in which small programs are evolved using the principles of evolutionary algorithms. Koza [13] applied the concept of genetic programming to the task of finding mathematical expressions. Mathematical operators, such as addition and subtraction, and functions such as sine and cosine form the basic building blocks of these evolved expressions. Through the repeated application of selection, crossover, and mutation, expressions are evolved and optimized to describe the given dataset more accurately. In each iteration of the algorithm, the quality of each expression is evaluated by determining how well it fits the (training) data. One way of quantifying is fitness is through the determination coefficient R^2. A subset of the population that performs well is selected and used as the progenitors of the next generation of candidate solutions. New expressions are generated by combining parts of the selected progenitors. Finally, a small amount of mutation is applied, such as changing a

single operator, adding a single operation, or removing a small part of the expression.

Aside from genetic programming, symbolic regression has also been approached with other strategies. These include methods rooted in Bayesian optimisation [11], recurrent neural networks [19], and divide-and-conquer methods [24].

A comprehensive evaluation of symbolic regression methods and (classical) machine learning techniques was recently performed by La Cava et al. [3], providing the field with an open-source platform (SRBench) to perform further evaluations and giving a robust basis to claims regarding the state of the art in symbolic regression. The study included ten symbolic regression algorithms based on genetic programming and four SR algorithms grounded in other techniques. Furthermore, seven non-SR machine learning methods, such as random forests, linear regression, and multi-layer perceptrons, were included. These algorithms were applied to 252 different problems and ranked by the average predictive quality of the generated models, as well as their final model size and required training time.

Based on this robust evaluation, we can draw conclusions regarding the state-of-the-art in symbolic regression. Operon, developed by Burlacu et al. [2], achieved the highest test performance while keeping the final models small and training times reasonable compared to other well-performing algorithms. Other high-scoring genetic programming methods are SBP-GP [25], FEAT [5] and EPLEX [4]. The classical approach XGBoost [6] also performed well, though its models were significantly larger and more complex.

La Cava et al. [3] make a distinction between black-box problems and ground-truth problems. In the first case, the data is observational, relating to some (real-world) phenomenon. In the second category, data is generated from a known mathematical expression. Certain algorithms, such as AI Feynman [24], are well-suited to finding exact solutions for ground-truth problems while being ill-suited to approximate the black-box class of problems. Conversely, many of the top algorithms are not meant to find exact solutions but do well on the approximation task. In this study, we are solely concerned with the ability to approximate, as sequential model-based optimisation does not require a perfect model. As long as the generated model has a global optimum for the same inputs as the function we wish to optimise, it will be effective.

2.2 Sequential Model-Based Optimisation

In sequential model-based optimisation (SMBO), the goal is to optimise[1] a function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}$ with a very expensive evaluation, called the objective function, by using a cheaper surrogate model from which we iteratively sample new inputs and then update again. Examples of functions we might want to optimise include complex industrial simulations and the training of large neural networks, for which the runtime or training time may be measured in hours or days. It may also be applied to non-digital cases, such as the optimisation of the yield of a chemical reaction. Such functions take a set of input parameters and ultimately yield

[1] In the experiments performed for this thesis, we will focus on minimisation problems. However, the concepts we discuss here apply to both minimisation and maximisation. We will usually use the term 'optimisation' to encompass both variations.

some numerical output. In the case of the neural network, the inputs would be the values of the hyperparameters and the achieved accuracy (or other metric) would be the output. For a chemical reaction, the inputs may include the mass of each included substance, as well as the temperature at which the reaction is performed, and the output could be the total yield of the desired substance in grams. Other applications include A/B testing, recommender systems, robotics, environmental monitoring, combinatorial optimisation, and natural language processing [21].

Because the evaluations of these objective functions are usually expensive in terms of computational power, time, or labour, it is intractable (or the very least undesirable) to optimise the function through a naive procedure like random search or grid search. A more informed alternative would be to have an expert practitioner or researcher in the loop who might analyse previous results to select new promising inputs. However, this approach is susceptible to human error and bias, does not always provide a structured way to cover the input space, and is still time and labour-intensive. Other optimisation approaches, such as CMA-ES, which are sophisticated and powerful, would also not be suitable for such tasks as they require many function evaluations.

Instead, in SMBO, after each evaluation of the objective function we use the data we have obtained so far to build a surrogate, a mathematical model that is orders of magnitude faster to construct and evaluate than the objective function. We use the model to select a new input vector that will likely perform well. Ideally, we also choose inputs that give a large amount of information to improve the model. This is especially relevant when the objective function has many local optima which we might get stuck in using a purely exploitative approach. We define an acquisition function that determines the quality of potential inputs, usually by balancing the value of the predicted output and the information that could be gained by selecting the input. Common options include the probability of improvement [14], expected improvement [12], and upper confidence bounds [22]. We obtain the most promising input candidate by maximising this acquisition function, which we then evaluate on the objective function. The input and the obtained value are then added to our dataset, allowing us to construct a better model in the next iteration. We can run this procedure until we observe convergence in the found values, or until we exhaust some predefined budget.

A popular algorithm in SMBO is Efficient Global Optimisation (EGO), which uses a Gaussian process to model the objective function. The Gaussian process represents a distribution of possible functions that could yield the sampled values. The mean of this distribution represents the most likely value for a given input. Additionally, we have information about the variance for a given input. EGO uses Expected Improvement (EI) as its acquisition function. We provide a definition for EI in equation 2.1, where Y is the random variable that models the objective function f. EI balances exploration and exploitation by combining the variance for a given input with the difference between the predicted mean and the current known optimum. In this way, inputs can be considered high quality if they either have a predicted mean that far exceeds the current known optimum or if their variance is very high. This balances the importance of exploration and exploitation in the algorithm.

$$EI(x) = E[\max(f_{min} - Y, 0)] \tag{2.1}$$

Chapter 3

Investigating Operon

Based on the evaluation of La Cava et al. [3] that we mentioned in the previous chapter, we identify Operon as the most effective SRGP algorithm, and we therefore elect to use Operon in our sequential model-based optimisation algorithm. Before we define and test the algorithm, we want to examine some relevant properties of Operon by itself. Recall from the previous chapter that in SMBO we want to keep the number of evaluations of the objective functions to a minimum, and thus we will have relatively few data points to construct (or in this case evolve) our model. Also, we need the generation of expressions and their evaluation to be orders of magnitude faster than the objective function we wish to optimise. This chapter evaluates Operon on these two aspects, data and time, to justify its integration in an SMBO algorithm and to identify potential weaknesses. Section 3.1 discusses the data efficiency of Operon, that is to say, the impact of the size of the training set on the predictive quality of the generated expressions. Section 3.2 considers examples from the literature to assess how much data will be available in an SMBO experiment. In section 3.3 we examine the time it takes Operon to generate expressions and the time it takes to evaluate points on them. Note that we are not yet concerned with highly precise benchmarking, only with the order of magnitude of the durations.

3.1 Data Efficiency

To determine Operon's suitability as a surrogate, we examine the relationship between the amount of available data points and the quality of generated models. We will refer to this relationship as Operon's data efficiency. In any optimization procedure, we would like to find the optimum of the objective function in the least amount of time. Because most of the time will be spent evaluating the objective function, we want to minimize the number of evaluations needed to find a good optimum. Consequently, we want our surrogate to perform well with a limited number of sample points. We, therefore, conduct an experiment in which we generate expressions using Operon with different subsets of a full training set and evaluate their quality in terms of the coefficient of determination R^2, as defined in equation 3.1. An R^2 of 1 indicates that the model perfectly predicts the data, while a value of 0 would be obtained by always predicting the mean of the dataset, and negative values indicate the model performs worse than this baseline. We evaluate R^2 over a test set of size N, which is separate from the training set. We want to understand the relationship between the size of the training set n and the number of input variables of the problem d on the predictive quality of the generated expressions, as measured by R^2. Understanding this relationship allows us to determine whether

Operon is likely to do well as a model in SMBO given a problem with known dimensions and a known budget. If we find that Operon does not approximate well with that number of data points, we can consider it unsuited to that specific task. By evaluating the quality of generated expressions over a wide range of dimensionalities, we can also assess whether Operon will be able to deal well with high-dimensional problems in SMBO. Ultimately, a surrogate that does not accurately model the value of the objective function, may still model its shape correctly, in which case it would lead us to the same optima. Therefore, we will also look at some visualised expressions that may reveal a more nuanced view than just the R^2 metric.

$$R^2 = 1 - \frac{\sum_i^N (y_i - \hat{y}_i)^2}{\sum_i^N (y_i - \overline{y}_i)^2} \tag{3.1}$$

We showcase the results of these experiments grouped into three categories of datasets. Two of them are from the PMLB framework [18] [20], the first being black-box datasets, and the second ground-truth datasets. The black-box problems range from having 1 to 124 input variables. The ground-truth problems have between 1 and 9 input variables. The final category is a group of self-generated synthetic datasets, based on popular optimisation test functions. Each function is fundamentally based on a summation of terms and can thus be scaled to an arbitrary number of dimensions. We used functions ranging from 1 to 100 input variables. The functions we used are the sphere function (3.2), the Rastrigin function (3.3), the Ackley function (3.4), the Rosenbrock function (3.5), and the Styblinski-Tang function (3.6).

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{3.2}$$

$$f(x) = An + \sum_{i=1}^{n} \left(x_i^2 - A \cdot cos(2\pi x_i)\right) \tag{3.3}$$

A = 10

$$f(x) = -a \cdot \exp\left(b\sqrt{\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{n} cos(cx_i)\right) + a + e \tag{3.4}$$

a = 20, b = 0.2, c = 2π

$$f(x) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right) \tag{3.5}$$

$$f(x) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2} \tag{3.6}$$

We first define a range of training set sizes. Then, for each dataset, we generate an expression with a random training sample of the full dataset, for each of the sizes. We repeat this process ten times to obtain averages. We then plot the data sample size against the achieved R^2, grouped by their dimensionality. Figure 3.1 shows the results for black-box datasets and figure

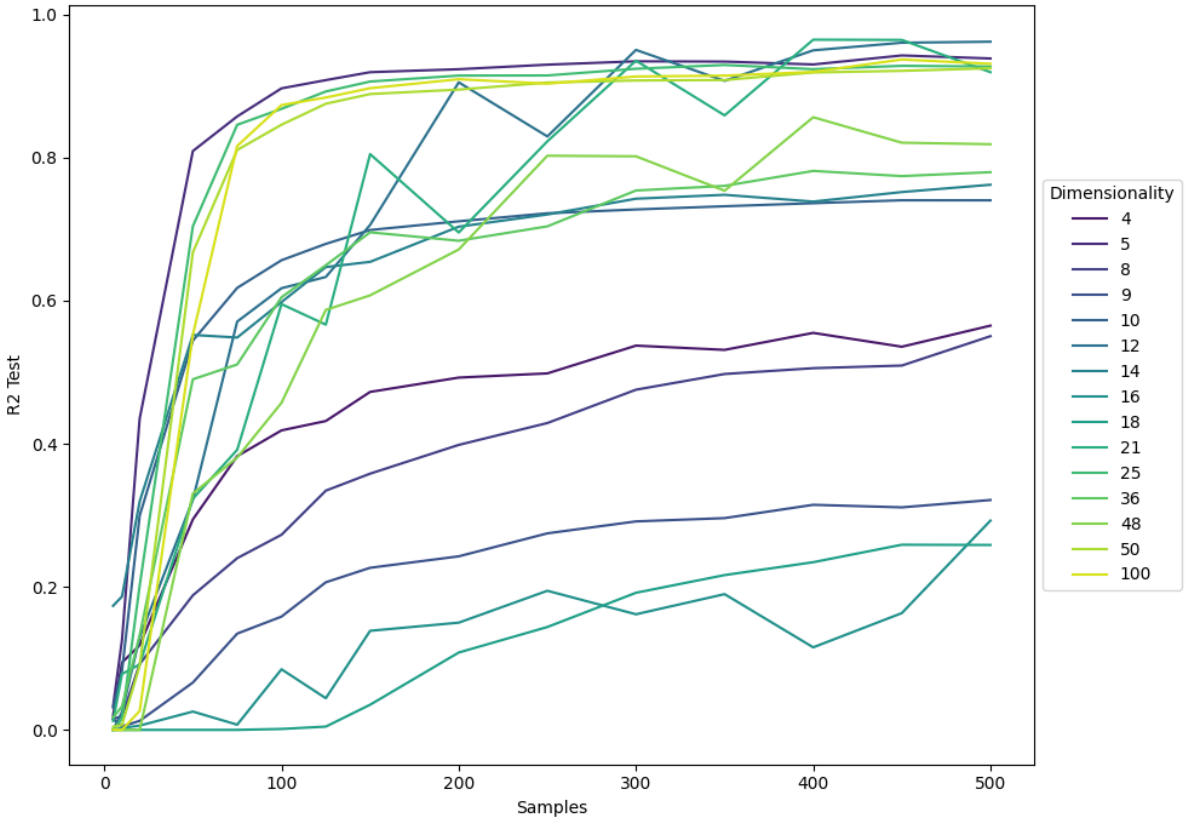

Figure 3.1: Lineplot of sample size against R^2, grouped by dimensionality for the black-box PMLB datasets.

3.2 shows the results for the ground-truth datasets. The results for the self-generated datasets are shown in figure 3.3. It should be noted that not all datasets are included in the figures of the PMLB data. To make the aggregations fair and useful, only datasets with enough data to be evaluated at each subset size are included (i.e. datasets with fewer points than the largest sample size are excluded). The dataset sizes used in these experiments are 5, 10, 20, 50, 75, 100, 125, 150, 200, 250, 300, 350, 400, 450, 500. We use the Symbolic Regressor framework provided by PyOperon [10], the Python interface to the C++ implementation of Operon. For the most part, we use the default parameters as configured in that package. However, we change the operator set to consist of addition, subtraction, multiplication, analytic quotient [16], and sine. This choice of operators is informed by findings reported by Nicola and Agapitos [17] concerning the impact of the operator set on the quality of models in SRGP.

For the black-box datasets in figure 3.1, we see that dimensionality is a poor predictor of the number of training samples required to achieve good or best-possible model quality. The dimensionalities are haphazardly distributed over the achieved qualities. Looking at the point where the curves flatten or where noticeable changes in the slope occur, it seems that most of the improvement occurs in the interval [5, 150], after which most curves settle into a gently sloped linear improvement. Here too, there is no obvious link to the dimensionality of datasets. As these datasets are constructed through observations of real-world phenomena, it is likely that not all variables that play a role in the relevant systems are captured in the data, and as such the missing data plays a greater role than the mere dimensionality. For most black-box problems it seems that upwards of a hundred samples may be required.

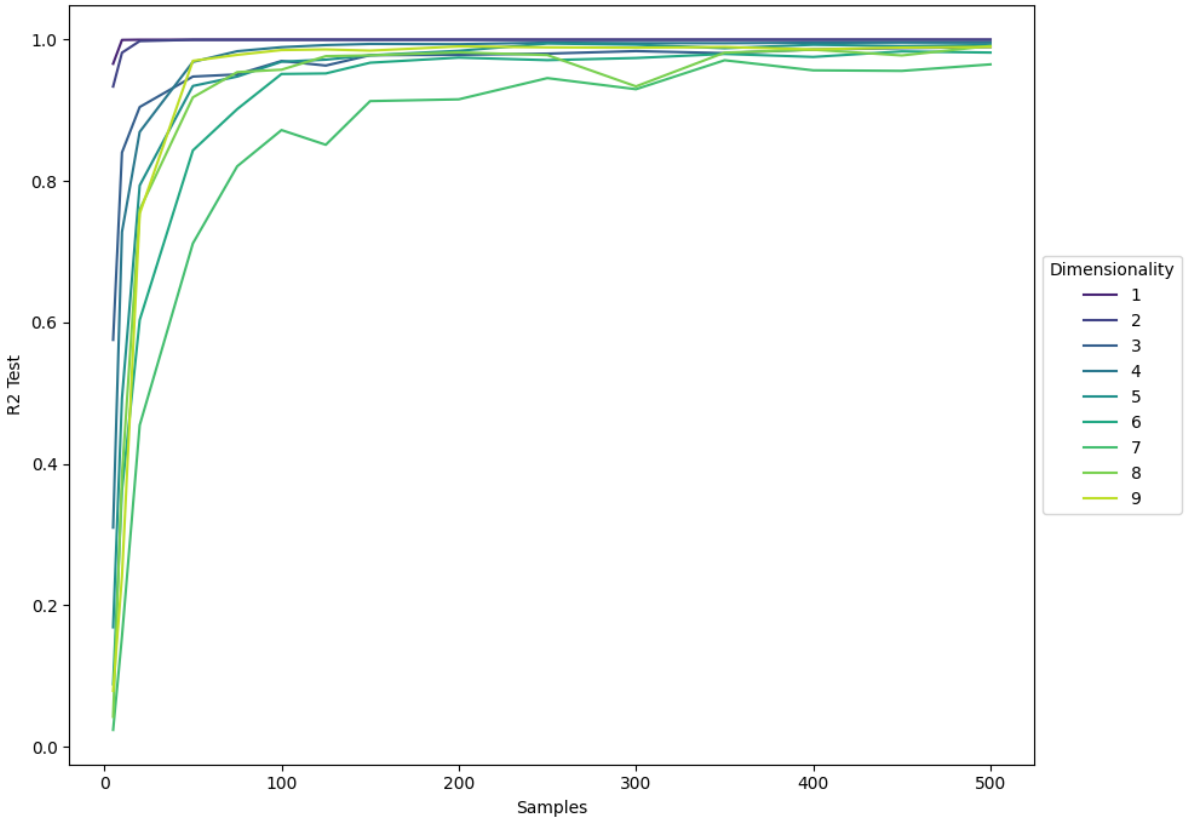

Figure 3.2: Lineplot of sample size against R^2, grouped by dimensionality for the ground-truth PMLB datasets.

For the ground-truth datasets in figure 3.2, we see that while datasets of each dimensionality reach a high level of quality there is some difference between the point at which they reach their maximum. Especially eye-catching are dimensionalities 1 and 2, which achieve maximum model quality at the lowest sample sizes. The relationship is not absolute, however. For example, looking at the yellowish curve of dimensionality 9, we see it surpasses other curves of lower dimensionality problems in terms of maximum achieved quality and in terms of the speed of growth. In a general sense, we can say that training sets in the size of hundreds of samples are required for high-quality models. These datasets cover a wide array of equations, and it seems that there are other aspects, besides the dimensionality, which determine its difficulty. Total size or the occurrence of certain operators and sub-functions may play a key role here.

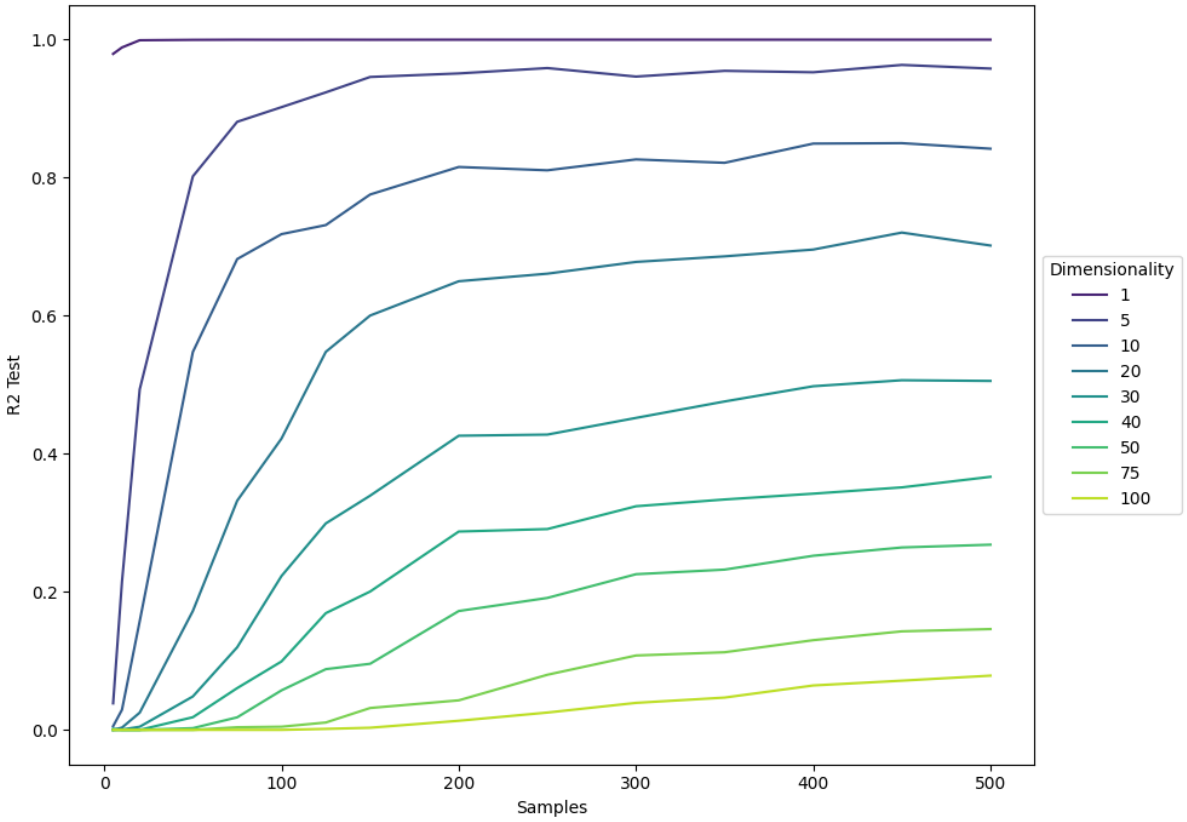

Figure 3.3: Lineplot of sample size against R^2, grouped by dimensionality for the self-generated ground-truth datasets.

In figure 3.3, we see our own ground-truth datasets paint a clearer picture. We see the expected drop in quality as the dimensionality of the problems grows. In contrast to datasets from PMLB, we have high and low-dimensional versions of each of our used functions, leading to averaged curves of similar shapes, as other features of each problem are the same between dimensionalities. Now we also see more clearly the different points at which the curves flatten. For a 1-dimensional case, we only need a handful of samples, while the high point for the 5-dimensional case is reached around 250 samples. For dimensionalities 10 and 20, the number of required samples looks to be 400 and 450 respectively. Beyond that, it is not clear when the curves will go flat, but we can expect the trend to hold. Naturally, the underlying function still plays a role, but as we won't know the specifics of the underlying function when looking at optimisation later, we cannot use such information to select our surrogate model.

Next, we show a few visualisations of the expressions that Operon generates and we give the closed-form expressions that were generated, simplified using SymPy [23]. Figure 3.4 through 3.7 show expressions evolved through Operon based on a small sample of points from some of the functions mentioned in section 3.1 on the domain $[-5, 5]$. The expression trees these visualisations are based on are limited to a depth and total node count of 5, to keep the expression human-readable.

In figure 3.4 we examine the 1-dimensional version of the function, where Operon had a training set of 3 samples. In table 3.1, we show the corresponding expressions. We see that expressions 2 and 3 while lacking the many local optima of the true function, do predict its minimum in the correct region. Expression 1 has two local minimal that are somewhat close to the Ackley function's global minimum, however, it predicts even lower values at the edges of the input interval. We note that expressions 2 and 3 are highly similar, being constructed out of the same operators but having slightly different coefficients.

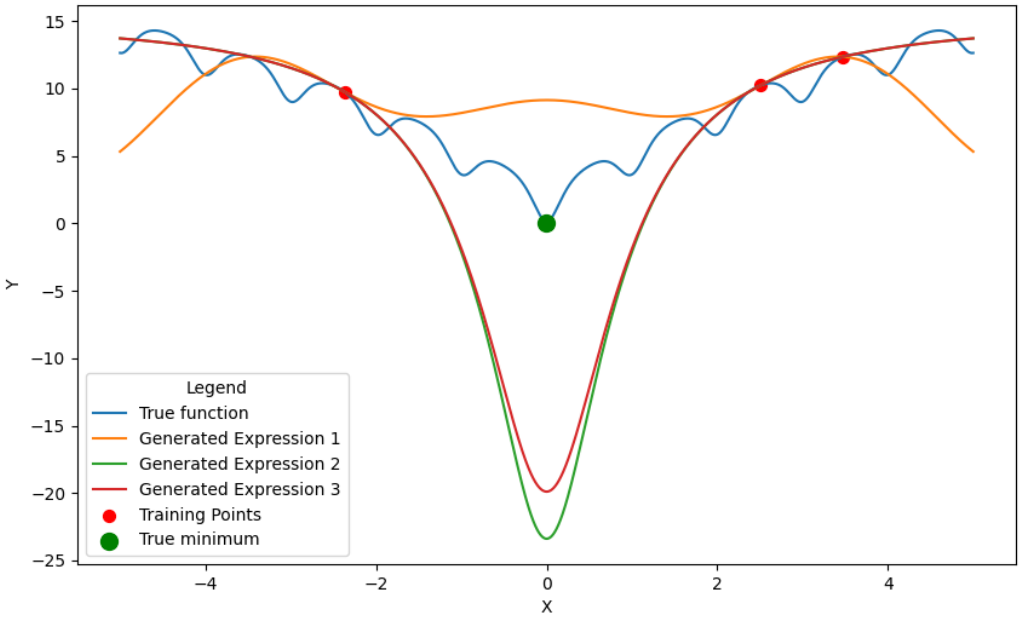

Figure 3.4: The 1-dimensional Ackley function along with three expressions generated by Operon using three training points.

Legend Name	Expression
Generated Expression 1	$-0.964x_1 * \sin x_1 + 9.144$
Generated Expression 2	$15.131 - \frac{24.516}{\sqrt{(0.436x_1)^2+1}\cdot\sqrt{x_1^2+0.404}}$
Generated Expression 3	$15.127 - \frac{25.622}{\sqrt{(0.480x_1)^2+1}\cdot\sqrt{x_1^2+0.535}}$

Table 3.1: Generated expressions for the 1-dimensional Ackley function using 3 sample points.

In figure 3.5 and table 3.2, we show the results for the 1-dimensional Ackley function when Operon is given 10 sample points to train on. All three generated expressions now have a highly similar structure, with expressions 2 and 3 being identical. In each expression, the global minimum now matches that of the Ackley function.

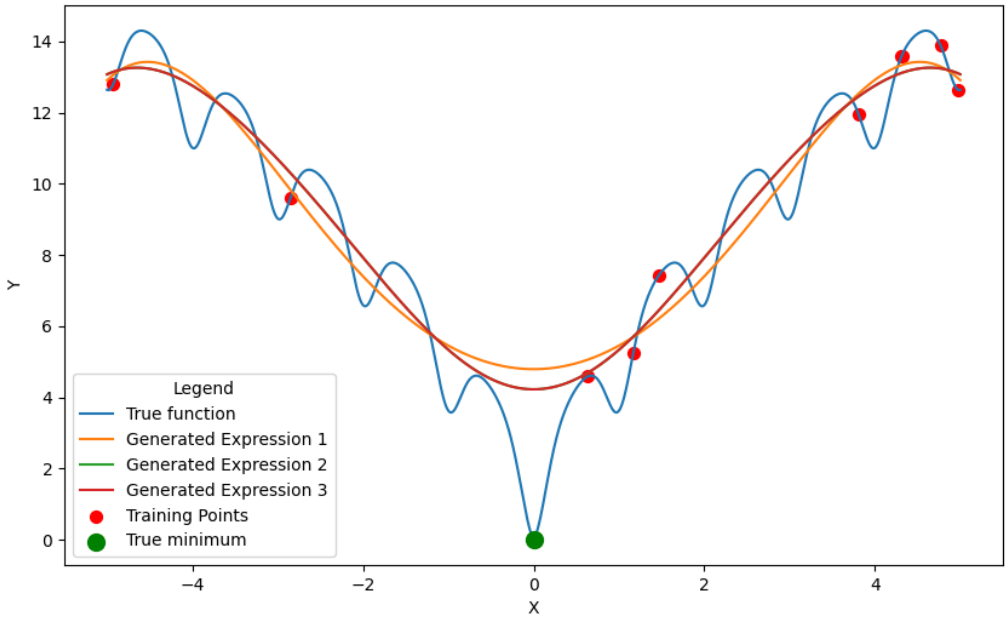

Figure 3.5: The 1-dimensional Ackley function along with three expressions generated by Operon using ten training points.

Legend Name	Expression
Generated Expression 1	$8.633 \cdot \sin(0.077x_1) + 4.791$
Generated Expression 2	$2.593x_1 \cdot \sin(\sin(0.458x_1)) + 4.225$
Generated Expression 3	$2.593x_1 \cdot \sin(\sin(0.458x_1)) + 4.225$

Table 3.2: Generated expressions for the 1-dimensional Ackley function using 10 sample points.

In figures 3.6 and 3.7, we show a visualisation of the Ackley function (left) and an expression generated by Operon (right) based on training sets of 6 and 20 samples respectively. Tables 3.3 and 3.4 contain the corresponding expressions. We see that Operon has essentially generated a much smoother version of the true function. Both the maxima in the corner and the minimum in the middle match the Ackley function. The expressions have a similar structure as those generated for the 1-dimensional case.

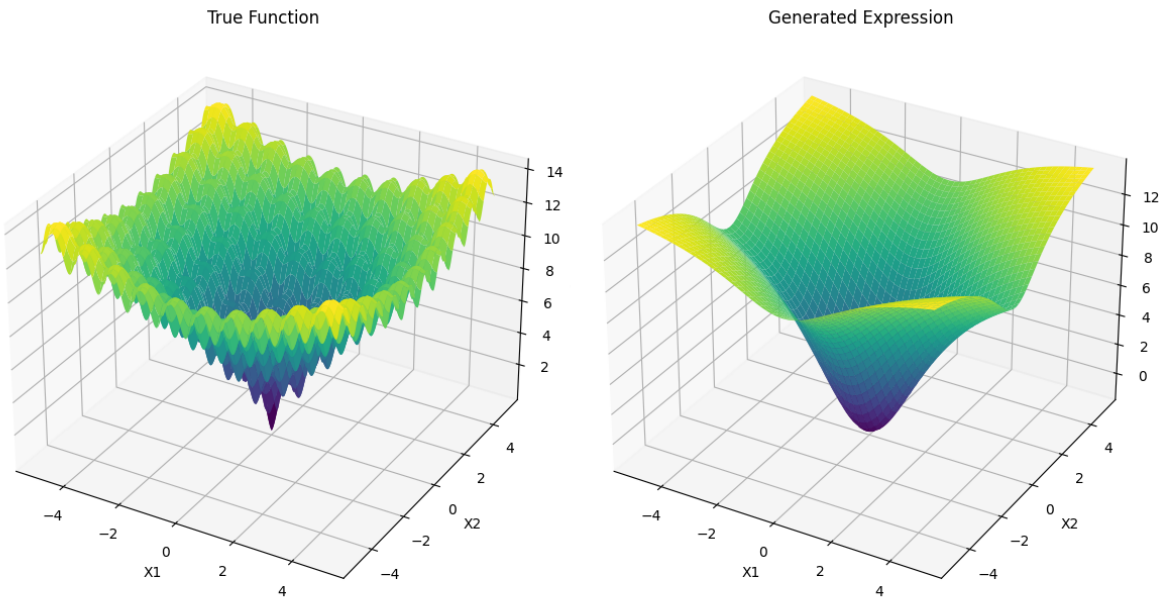

Figure 3.6: The 2-dimensional Ackley function and an expression generated by Operon using 6 sample points.

Legend Name	Expression
Generated Expression	$15.96 - \frac{17.353}{\sqrt{(0.255x_1)^2+1}\cdot\sqrt{(0.417x_2)^2+1}}$

Table 3.3: Generated expression for the 2-dimensional Ackley function using 6 sample points.

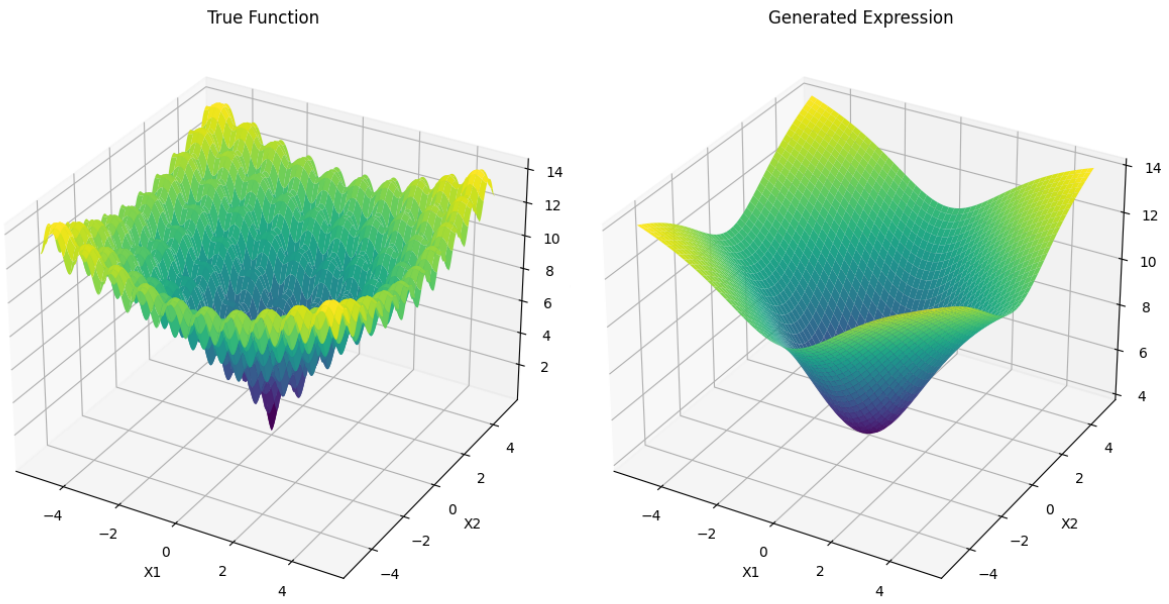

Figure 3.7: The 2-dimensional Ackley function and an expression generated by Operon using 20 sample points.

Legend Name	Expression
Generated Expression	$16.712 - \frac{12.753}{\sqrt{(0.16x_1)^2+1}\cdot\sqrt{(0.144x_2)^2+1}}$

Table 3.4: Generated expression for the 2-dimensional Ackley function using 20 sample points.

From these visualisations, we conclude that Operon will often generate expressions whose global shape resembles that of the true function, even when it is unable to accurately model a complex landscape containing many local optima. This bodes well for the optimisation task, where we need the generated expressions to indicate where the optimum of the objective functions is.

3.2 Optimisation Budgets

Now that we have an idea of the amount of data Operon needs to perform well, we examine the budgets used in optimisation, which are measured in the number of evaluations of the objective function, to see if we can reasonably expect Operon to be a viable surrogate. Recall that in an SMBO algorithm, we evaluate a new input in each iteration and add the resulting data point to the set from which we construct future models. The evaluation budget of our algorithm thus also determines the amount of available data for our model construction. In practical use, the cost and availability of computational resources and deadlines will play a role in determining how much budget is allocated. A reasonable range of what to expect can be found in the scientific literature. Bossek et al. [1] use a range of static budgets of sizes (2^4, .., 2^9). Mueller [15] uses budgets scaled to the dimensionality of the problem ($50d$). Scaling the budget based on the problem's dimensionality seems sensible given the results we have seen in section 3.1, which indicate that problems with higher dimensionality require more data.

Eriksson et al. [7] examine different problems with different budgets, ranging from two thousand to twenty thousand evaluations. Given this wide range of options, and comparing them to the required sample sizes for Operon we observed in the previous section, we can conclude that there would be enough data in optimisation procedures for Operon to make viable models. Considering the computational resources available for this study, as well as keeping a fair comparison to the Gaussian process (which may not need as many data points as Operon), we consider a total budget of $10d$ for the experiments in chapter 4.

3.3 Modelling Time

Aside from data efficiency, we have another aspect important to Operon's viability as a surrogate: the amount of time Operon needs to generate an expression. Recall that the purpose of using a surrogate model is to be able to make an informed decision regarding the selection of new inputs in order to cut down the number of required evaluations of the expensive objective function. The combined construction of the model and the selection procedure should thus be much faster than the full evaluation. Since the exact evaluation time of the objective function will vary from case to case, we deal in orders of magnitude. In the study done by La Cava et al. [3], the training speed of SR methods was evaluated. For Operon, they report an average training time of around 10^3 seconds. However, they only allowed each algorithm to use a single processor core. Operon, however, is explicitly designed to leverage parallel hardware to achieve high performance. The prevalence of multi-core processors in modern hardware leads us to assume that any practitioner looking to optimise a process using SMBO will have access to at least some parallel hardware, in which case Operon's training time will be reduced. We also note that the number of training samples determines the time spent on each evaluation of an individual expression and thus plays a large role in the total training time. The data sets used by La Cava et al. range from a few dozen to hundreds of thousands of samples. Since the size of training sets in SMBO will be limited by the evaluation budget discussed in section 3.2, we will likely see shorter training times.

To get a more reasonable estimate of training times taking into account paralleled hardware and smaller training sets, we perform a simple evaluation of training times ourselves. We let Operon generate expressions using subsets of the data we have for the 20-dimensional Rastrigin function. We examine subsets of sizes 100, 1000, and 10000. We run Operon 20 times for each size and record the mean time it takes for it to terminate, as well as the mean time it takes to evaluate a test set of 10000 samples on the generated expressions. We use the default settings of the PyOperon [10] package, except we increase the number of threads Operon may use to 8. We run this test on an 11th Gen Intel Core i7-11700 @ 2.50GHz. The results are shown in table 3.5. The evaluation shows the combined effect of small training set size and parallel hardware, as the training times are two orders of magnitude smaller than the average reported by La Cava et al. Evaluation time is a negligible fraction of the training time. We can safely conclude that Operon is fast enough to generate models in an SMBO algorithm.

# Training Samples	Training Time (s)	Evaluation Time (ms)
100	0.723	0.490
1000	2.038	0.540
10000	13.288	0.574

Table 3.5: Mean training time in seconds and evaluation time in milliseconds of Operon using default settings and different numbers of training samples.

Chapter 4

SMBO Experiment

Given its adequate performance in terms of data efficiency and training speed, we conclude that Operon may be a viable surrogate model. Having clarified its general suitability, we proceed with its evaluation by comparing it to alternatives. We, therefore, conducted an experiment in which we compared it to two approaches. The most important one is EGO (algorithm 1) which uses a Gaussian process as a model and the Expected Improvement acquisition function. The other is a naive baseline method: random search using Sobol sampling. For Operon, we start with a purely exploitative method, in which we find the optimum of the generated expression and take the corresponding inputs for the next objective function evaluation. We also test another version which includes a simple exploration scheme. We generate q separate expressions, optimise each of them and then choose the best of those optima. The intuition behind this procedure is that sampling multiple possible expressions will result in a higher chance that we sample an expression with a very extreme shape leading to a new possible optimum. In the best case, such an extreme shape actually occurs in the objective function, helping us leave a local optimum. In the worst case, this will help us fill the gaps in the training set since extreme shapes are more likely to occur where data is scarce, as we discussed in section 3.1. This simple exploration scheme is expected to yield improved results over the purely exploitative approach where $q = 1$. We provide pseudocode for our SMBO approach using Operon in algorithm 2. The pseudocode includes the potential exploration scheme. In principle, any maximisation method that can be performed in a small amount of time is suitable to maximise the generated expressions. We choose to use CMA-ES for this task. We run the algorithm with a population of $n = 4 + \lfloor(3 \cdot \ln(d))\rfloor$. We stop after ten generations of no improvement or after 100 generations at the latest.

Algorithm 1 Efficient Global Optimisation (EGO)

Input: Objective function $f(x) : \mathcal{X} \to \mathcal{Y}$, initial sample size n, evaluation budget N
Output: Optimal function value $f(x^*) \in \mathcal{Y}$ and corresponding input $x^* \in \mathcal{X}$
Initialize the dataset $\mathcal{D}$ with n Sobol samples from $\mathcal{X}$: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $y_i = f(x_i)$
for $t = n$ to N **do**
    Fit a Gaussian Process (GP) model to $\mathcal{D}_{t-1}$
    select x_t by optimising the Expected Improvement (EI) acquisition function:

$$x_t = \arg\max_x \mathrm{EI}(x)$$

    Evaluate the objective function at the new point: $y_t = f(x_t)$
    Update the dataset $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$
end for
Return: The best observed point $x^* = \arg\min_x y$ and the corresponding value $f(x^*)$

Algorithm 2 Global Optimisation with Operon (GOO)

Input: Objective function $f(x) : \mathcal{X} \to \mathcal{Y}$, initial sample size n, evaluation budget N, function sample count q
Output: Optimal function value $f(x^*) \in \mathcal{Y}$ and corresponding input $x^* \in \mathcal{X}$
Initialize the dataset $\mathcal{D}$ with n Sobol samples from $\mathcal{X}$: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $y_i = f(x_i)$
for $t = n$ to N **do**
    Generate expressions $\{e_1, .., e_q\}$ using Operon and training set $\mathcal{D}_{t-1}$
    select x_t by finding the optimum of all expressions $x_t = \arg\min_x \{e_1, .., e_q\}$
    Evaluate the objective function at the new point: $y_t = f(x_t)$
    Update the dataset $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$
end for
Return: The best observed point $x^* = \arg\min_x y$ and the corresponding value $f(x^*)$

For Operon, we consider a few different hyperparameter configurations, as shown in table 4.1. Q refers to the number of generated expressions in each iteration, which influences our simple exploration scheme. Max depth refers to the maximum depth of the tree structures underlying the expressions generated by Operon and max length refers to the maximum number of nodes in said trees. These two values dictate the size and complexity of generated expressions. Values that deviate from the standard ones in the O-1 configuration are marked in bold. The allowed operator set for all configurations consists of subtraction, addition, multiplication, analytic quotient, and sine. For any unmentioned settings, the default values in the PyOperon package [10] were used, notably a population size of 1000 and a total evaluation budget of 1 million.

Configuration	Q	Max Depth	Max Length
O-1	1	20	100
O-2	**5**	20	100
O-3	1	**10**	**50**
O-4	1	**5**	**25**

Table 4.1: Hyperparameter configurations for Global Optimisation with Operon. O-1 is considered the standard, and deviating parameters are marked in bold.

For this optimisation experiment, we use the same functions that we used to generate our own datasets, as shown in equations 3.2 through 3.6. We conduct separate experiments for different dimensionalities d (2, 3, 5, 8, 13), totalling 25 different settings. We clip input values to the interval $[-5, 5]$ in each of the dimensions. Each experiment has a total budget of $10d$ evaluations of the objective function. The first $3d$ evaluations are obtained randomly through Sobol sampling, to give the surrogate models an initial training set to work with. Intuitively, this means the initial models are based on an evenly spread set of values, rather than just a singular value that may bias the initial model. Work by Bossek et al. [1] has investigated the optimal size or proportion of such an initial budget and concluded that small initial budgets are often optimal. Based on their findings, we consider that our initial ratio budget of 0.3 can be considered reasonably well-performant for EGO, while also providing Operon with a solid foundation. Each combination of an objective function, dimensionality, and surrogate model is run 20 independent times.

In table 4.2, we show the best function values found by each algorithm. Each row shows the median of the final objective function values achieved by each algorithm (configuration) on a combination of function and dimensionality d, as well as the known optimum $f(x^*)$ for that setting. The lowest value found in each setting is marked in bold. We also mark settings with an asterisk if both EGO and Sobol sampling are outperformed by all Operon configurations. For each setting, we provide plots for the best-so-far value at each iteration. The full collection of plots can be found in appendix A. We highlight a few of these plots with notable observations.

Table 4.2 shows that in 20 of the 25 settings, some configuration of our Operon-based algorithm outperforms EGO. In 5 settings Operon does not perform best. We see that 4 of the 5 settings where EGO finds a better optimum belong to the Styblinski-Tang function. Notably, the Operon-based configurations struggle to outperform the Sobol sampling for this function. The fifth setting in which EGO performs best is the 13-dimensional Rastrigin function.

In 7 of the settings EGO is outperformed by every configuration of Operon, but in 13 others, the specific hyperparameter values determine whether or not Operon beats EGO. In the remaining five, EGO outperforms every version of Operon. In table 4.3, we show the number of settings in which a specific Operon configuration beats both EGO and Sobol sampling. We see that the O-3 configuration, which does not use our exploration scheme and which has a reduced maximum size and complexity of expressions, has the most robust track record. The O-4 configuration, which is limited to even smaller expressions manages to find the best optimum in a few low-dimensional settings, but often yields a bad result in high-dimensional cases. In these settings, not all of the input variables can be modelled due to the restrictions, leading in predictably poor performance. Configuration O-3, however, does not suffer from this issue, as the dimensionalities used in this experiment do not exceed the capacity of its

Function	d	$f(x^*)$	EGO	Sobol	O-1	O-2	O-3	O-4
Sphere*	2	0.0	2.2e-4	0.721	4.9e-12	2.4e-12	1.3e-12	**4.1e-16**
Sphere*	3	0.0	8.6e-4	3.045	1.1e-6	2.2e-6	**7.1e-7**	2.8e-6
Sphere*	5	0.0	1.2e-3	7.967	8.4e-6	1.4e-5	**6e-6**	1.2e-4
Sphere	8	0.0	4.1e-3	19.554	1.6e-5	**1.3e-5**	1.8e-5	0.49
Sphere	13	0.0	0.011	46.092	1e-4	7.6e-5	**7.2e-5**	4.3
Rastrigin*	2	0.0	5.12	10.05	1.01	**0.41**	1.09	0.56
Rastrigin*	3	0.0	12.32	20.74	3.76	3.05	**1.08**	2.61
Rastrigin*	5	0.0	16.88	42.61	5.38	7.18	**1.37**	6.13
Rastrigin*	8	0.0	26.55	78.92	23.82	18.46	**16.0**	22.37
Rastrigin	13	0.0	**39.86**	146.92	57.84	59.19	43.77	77.76
Ackley*	2	0.0	0.17	4.72	5e-4	3.8e-4	**3.3e-4**	4.1e-4
Ackley*	3	0.0	0.39	5.09	7.5e-4	6.7e-4	**3.8e-4**	6.5e-4
Ackley*	5	0.0	0.73	6.31	**7.1e-4**	8.9e-4	6.9e-3	1.3e-3
Ackley*	8	0.0	1.61	7.21	1.3e-3	1.4e-3	**1.2e-3**	1.204
Ackley	13	0.0	2.92	7.93	0.014	0.015	**5.2e-3**	3.353
Rosenbrock*	2	0.0	1.61	12.03	0.38	0.7	0.15	**0.21**
Rosenbrock	3	0.0	14.3	119.04	19.16	16.98	5.54	**3.7**
Rosenbrock*	5	0.0	81.11	2268.68	62.67	63.92	32.76	**8.66**
Rosenbrock	8	0.0	134.44	8837.5	74.27	167.96	**50.55**	103.25
Rosenbrock	13	0.0	263.95	29997.73	401.68	255.1	**249.16**	1122.81
Styblinski-Tang	2	-78.332	-69.23	-61.59	-67.72	-71.21	**-73.82**	-65.49
Styblinski-Tang	3	-117.498	**-106.46**	-94.57	-92.36	-95.18	-97.01	-80.64
Styblinski-Tang	5	-195.831	**-179.75**	-141.89	-141.89	-125.76	-128.33	-123.37
Styblinski-Tang	8	-313.329	**-276.19**	-205.48	-195.35	-191.02	-189.66	-170.87
Styblinski-Tang	13	-509.160	**-438.43**	-281.84	-279.73	-285.7	-269.71	-277.22

Table 4.2: Median optimisation results from 20 independent runs for each algorithm in the 25 different settings. The best result per setting is marked in bold. An asterisk behind the function name indicates that all Operon configurations outperformed EGO and Sobol sampling in that setting.

Configuration	# Settings
O-1	18 / 25
O-2	17 / 25
O-3	20 / 25
O-4	16 / 25

Table 4.3: Number of settings in which a given Operon configuration outperforms both EGO and Sobol sampling.

expressions. Furthermore, the enforced simplicity seems to yield better results when compared to the larger expressions in O-1 and O-2. Comparing configurations O-1 and O-2, we see no consistent ranking between the two. This indicates that our simple exploration scheme yields no meaningful benefit. Neither does it cause the algorithm to find worse optima, but since it does increase the runtime of the algorithm proportionally to the value of q, overall it is detrimental.

The poor performance of the Operon-based algorithm on the Styblinski-Tang function could be a result of the chosen operator set for Operon. The function contains a summation over a 4th-degree polynomial, but the operator set does not contain exponentiation, which means these parts of the function will have to be modelled through repeated multiplication of the same input. This becomes especially difficult in high dimensions. Each of the used functions also includes a summation operator, which provides a succinct way to denote the application of the same term to each input, while Operon has to reconstruct this term for each input.

While Operon performs well in most cases, we observe that EGO shows a strongly early start in some cases. In the 13-dimensional Rosenbrock and sphere functions, for which the results are shown in figures 4.1 and 4.2, we observe that EGO has a lead on Operon for a large part of the run. In the sphere function, it is in the lead for around 40 iterations, and in the Rosenbrock this lead lasts around 60 iterations. In cases with very little budget, this could be a weakness for Operon, which needs more data to start generating solid expressions, especially in high dimensions. However, we do not observe this phenomenon in the results of the Rastrigin, Ackley, and Styblinski-Tang function, where Operon is either in the lead from the start and remains there, or the progress of both algorithms is similar until EGO overtakes Operon.

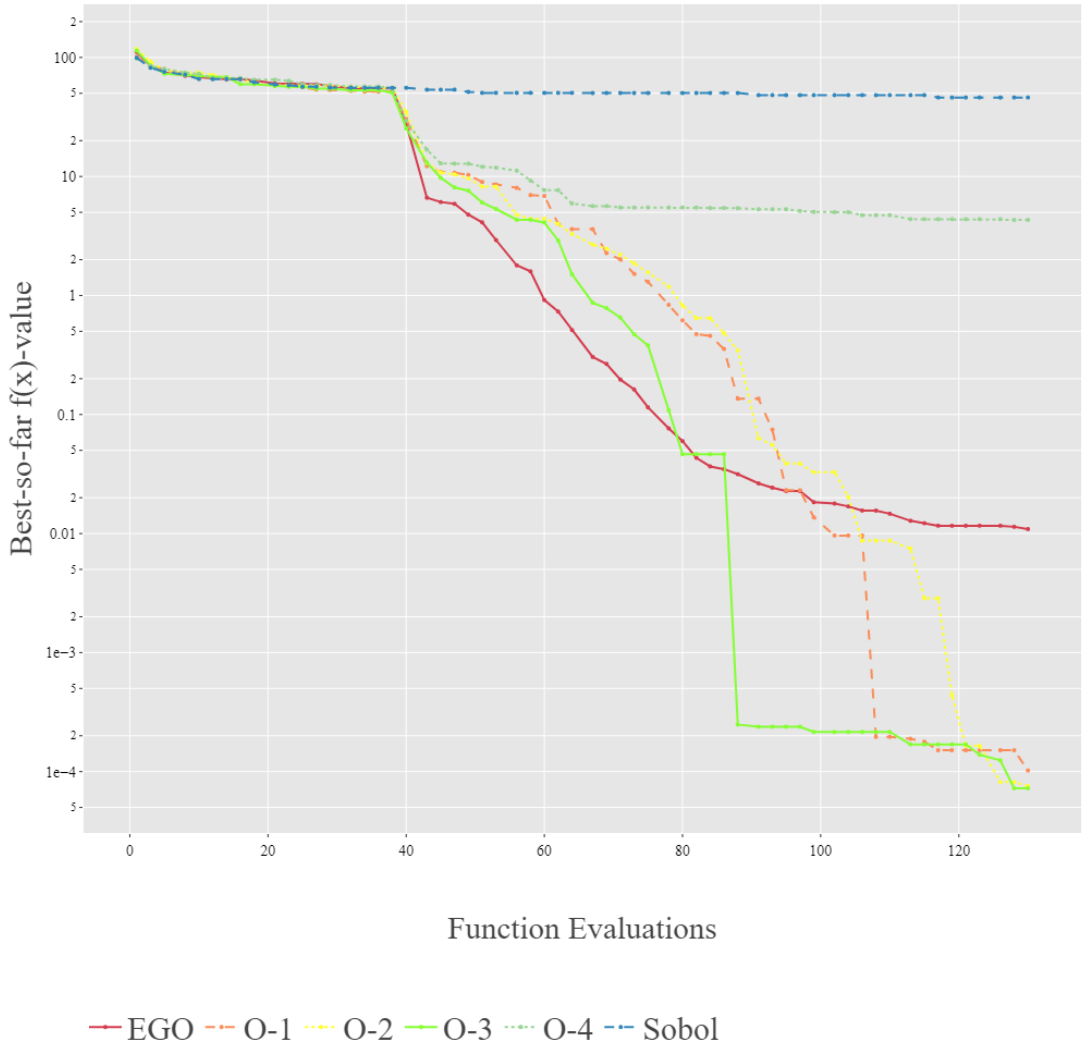

Figure 4.1: Results for the 13-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

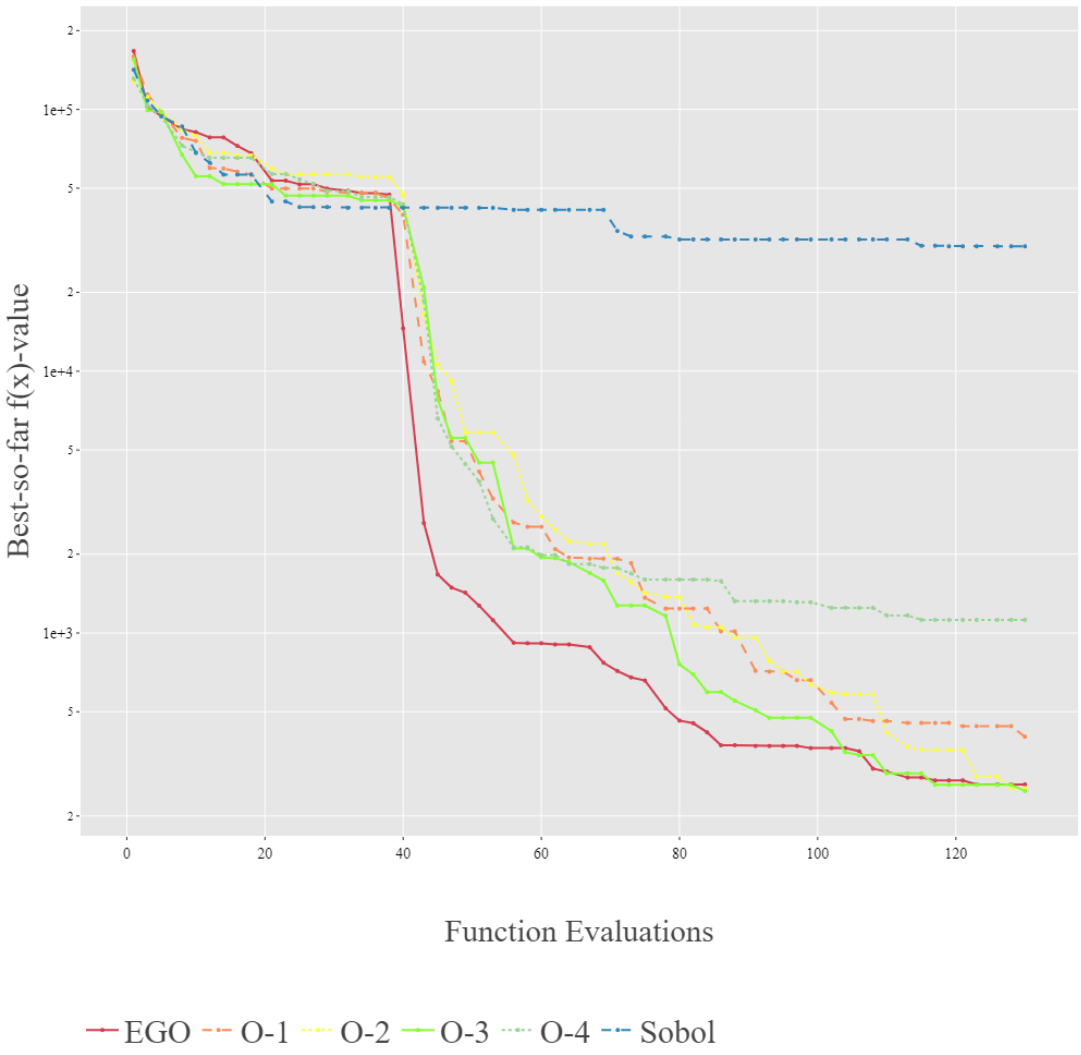

Figure 4.2: Results for the 13-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

In many settings, such as the 13-dimensional Ackley function (figure 4.3), we observe that all algorithms are still making progress toward better optima by the end of the budget. If more budget is available, even better values may be found. It is possible that an algorithm surpasses another that performs better on a smaller budget. However, this seems unlikely in this specific example.

Even though our simple exploration scheme does not offer a measurable benefit to the Operon-based algorithm, we still manage to outperform EGO, which includes a powerful exploration mechanism. Exploration is usually considered important in function landscapes with many local optima. Aside from the sphere function, all of our functions have multiple local optima. The Rastrigin and Ackley functions have a particularly large number of local minima, but barring the 13-dimensional Rastrigin setting, Operon is able to beat EGO without considering exploration in its acquisition function. Given the absence of long plateaus at the end of most

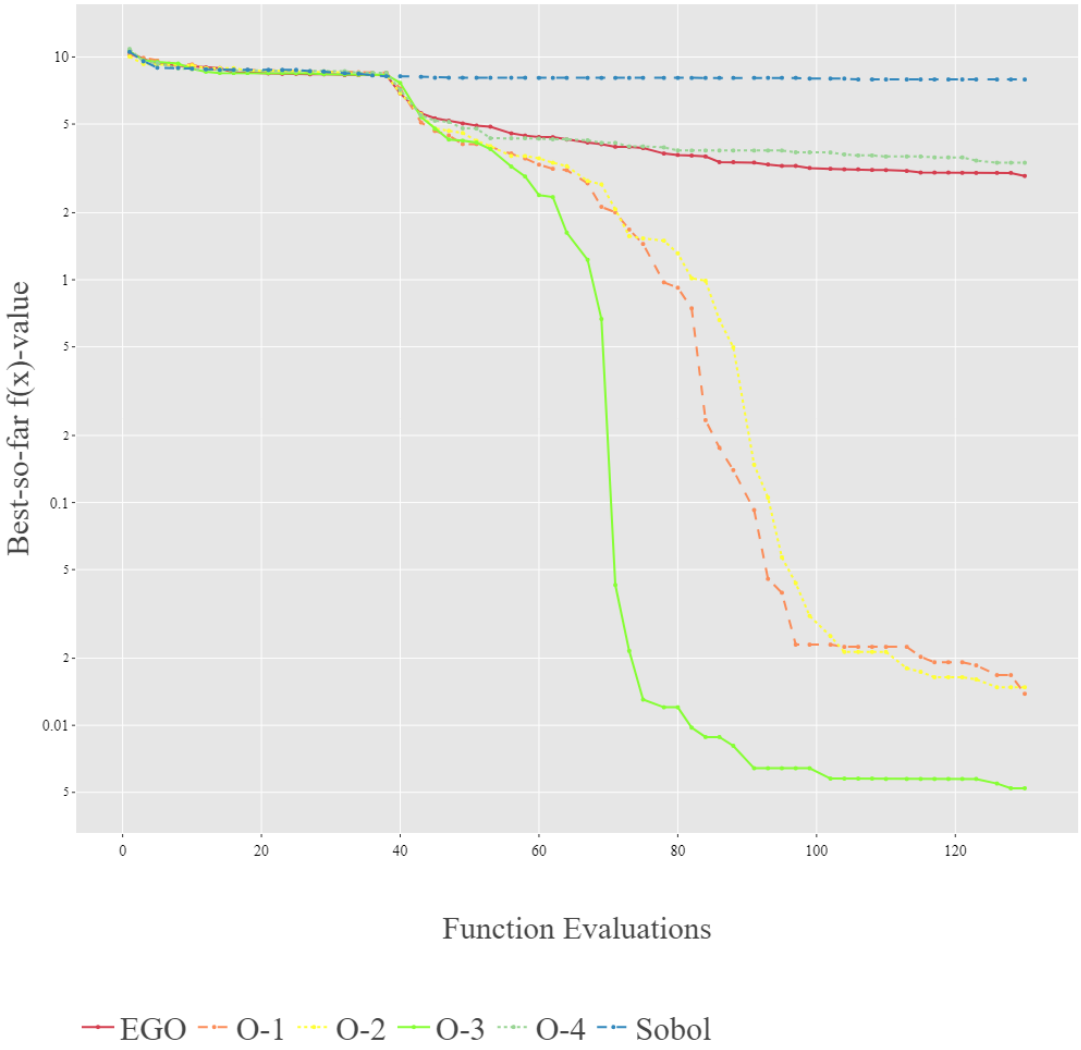

Figure 4.3: Results for the 13-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

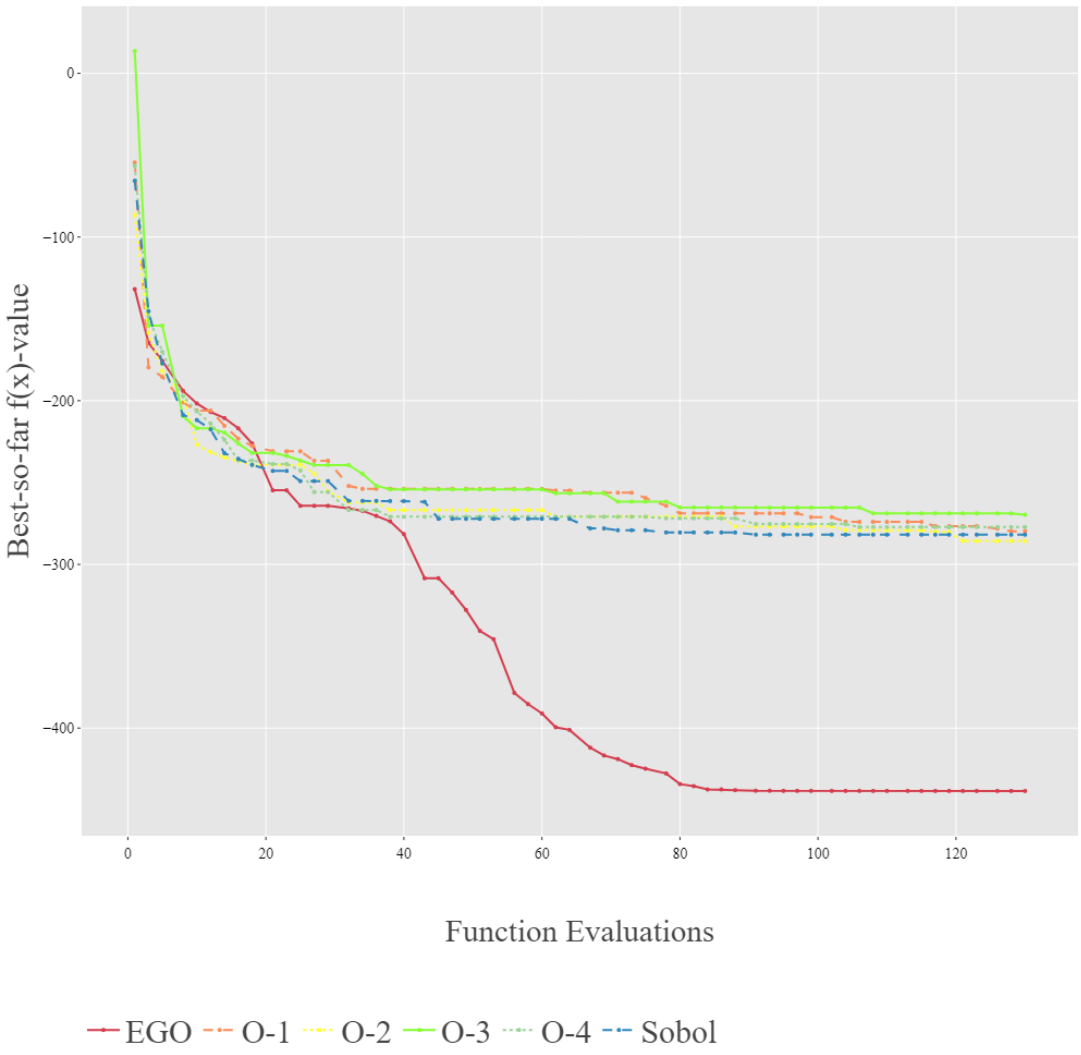

Figure 4.4: Results for the 13-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

settings, we do not think that the algorithm converges in most cases on the given budget and can therefore not rule out that the global optimum would be found on a larger budget. On the Styblinski-Tang function, we do see the algorithms converge. See, for example, the 13-dimensional setting in figure 4.4. Both EGO and all Operon configurations converge without reaching the global optimum. The Styblinski-Tang function does not have a landscape with as many local minima as the Rastrigin or Ackley function, though it does have some, unlike the sphere function. It seems likely that the problem of modelling the exponentiation that we mentioned before is the culprit here. A stronger exploration, if it results in a better spread of data points, could help Operon overcome this hurdle. However, parameters of the Operon algorithm such as maximum length and depth, or population size, may play a larger role.

Chapter 5

Conclusions and Discussion

In this chapter, we reflect on the results of our experiments. In section 5.1, we draw conclusions pertaining to the questions laid out in chapter 1. In section 5.2, we discuss possible improvements to our algorithm and suggest directions for further research into the application of SRGP in SMBO.

5.1 Conclusions

In this thesis, we have proposed and evaluated a novel sequential model-based optimisation (SMBO) algorithm that uses symbolic regression through genetic programming (SRGP) to construct models of the objective function. We used the state-of-the-art SRGP implementation called Operon. We compared this approach to the Efficient Global Optimisation (EGO) algorithm and to a Sobol sampling baseline. Our primary research question was: "Can an SMBO algorithm which uses an SRGP method to model the objective function find better optima than EGO on a set of test problems?".

Based on the results of the comparison, we conclude that, for most of the tested settings, our Operon-based algorithm finds better optima within the budget of $10d$ evaluations. We tested several hyperparameter configurations for our algorithm and observed that all of them exceeded the performance of EGO in a majority of settings. We, therefore, conclude that SRGP is indeed a viable choice for the modelling step in SMBO algorithms, as exemplified by Operon. On the relatively modest budget sizes we used, we do not always see convergence for our algorithm, nor for EGO. Relative performance may therefore be different on higher budgets, but in most cases, the slopes of the optimisation curves would indicate that our algorithm remains competitive.

Our evaluation specifically dealt with two subquestions. The first question is about the sort of problems SRGP is suitable for: "Is there a subset of problems on which SRGP performs especially well or poorly?".
In the settings where EGO was able to find better optima, the function is often hard to model for Operon due to the function's complexity. A high dimensionality is part of this, but more important is the lack of an exponentiation operator in our configuration of Operon. This makes it difficult for Operon to build accurate models of the Styblinski-Tang function, which includes summation over a 4th-degree polynomial. As the quality of models generated by Operon drops, the overall efficacy of our SMBO algorithm suffers. We also see that when we

reduce the maximum size and complexity of generated expressions significantly, our algorithm struggles with high-dimensional functions. We conclude that our algorithm underperforms in cases where the problem is too complex for Operon to model.
Our second subquestion concerns the number of required evaluations: "In cases where the optima found by EGO and Operon are (close to) equal, is there a difference in the total number of objective function evaluations required to find these solutions?". We do not observe this. When the algorithms find similar values, their speed toward those values is similar. In some cases, EGO starts faster than our algorithm and is only overtaken near the end of the budget. Since we used relatively modest budgets, we do not consider this a significant weakness of our approach.

Our algorithm can be considered purely exploitative. We evaluated the effect of an exploration mechanic based on a simple intuition regarding the way expressions are constructed. We conclude that this exploration scheme did not have a positive effect on the found optima. While our exploitative approach outperforms the EGO algorithm, which does include exploration, we find it reasonable to assume that the addition of a more sophisticated exploration scheme to our proposed algorithm could boost it even further.

We see the results of this study as a proof of concept for the use of symbolic regression algorithms in the domain of SMBO. Operon has proven itself as a suitable approach to generate expressions, and it is not unreasonable to think that other symbolic regression algorithms could also fulfil that role. We conclude that sequential model-based optimisation algorithms using SRGP are a viable and promising alternative to existing SMBO methods.

5.2 Discussion

While the results in this study are favourable for our novel algorithm, the comparison to EGO is only a start to determine where our algorithm shines. Future research may look to extend the evaluation to a larger set of test problems, such as the BBOB test suite of the COCO framework [9]. At the same time, a larger variety of optimisation algorithms may be used to put our novel algorithm into a broader context. If the COCO framework is used, work by Hansen et al. [8] may provide an indication of which algorithms are worthwhile to compare against.

Since the proposed exploration component for our algorithm in this study proved ineffective, we consider an effective exploration method to be a promising addition to the algorithm. One approach that could be explored is based on the distance of new inputs to their nearest neighbours in the training set. By defining an acquisition function that values a mix of the predicted value and the distance to the nearest neighbours, the algorithm could fill in the gaps between distant sampled points. Candidate inputs with a high distance to their nearest neighbours would be valued to encourage exploration, while inputs with an optimal predicted output would be valued to encourage exploitation. Such an acquisition function could include weights for both components to allow for tuning.

We have identified a potential weakness of our algorithm in functions that are complex to model, either because of operators that are difficult for Operon to replicate or because of high

dimensionality. Research dealing with complex or high-dimensional functions should focus on the operator set and the maximum size and complexity of generated expressions. The complexity of generated expressions also plays a role in the study of explainable AI. Determining the lower bounds of complexity for which expressions still adequately model functions in SMBO could help to create algorithms from which we derive not just an optimal value, but also a final model that is human-interpretable.

Finally, investigating the efficacy of the algorithm at higher budgets could be worthwhile. Currently, we have no clear view of the budget our algorithm needs to converge. In our experiments with a limited budget, the algorithm was not able to find the global optimum, but it also did not converge to a local optimum. Evaluating the algorithm on large budgets could tell us whether it can find the true optimum of a given function or if it will get stuck in a local optimum.

In conclusion, the results obtained in this thesis show great promise for the use of SRGP in SMBO. By improving on the proposed algorithm, and investigating a broad array of relevant properties, this new approach to SMBO can be developed into a full-fledged family of powerful algorithms.

References

[1] Jakob Bossek, Carola Doerr, and Pascal Kerschke. "Initial design strategies and their effects on sequential model-based optimization: an exploratory case study based on BBOB". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 778–786. ISBN: 9781450371285. DOI: 10.1145/3377930.3390155. URL: https://doi.org/10.1145/3377930.3390155.

[2] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. "Operon C++: An Efficient Genetic Programming Framework for Symbolic Regression". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1562–1570. ISBN: 9781450371278. DOI: 10.1145/3377929.3398099. URL: https://doi.org/10.1145/3377929.3398099.

[3] William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. "Contemporary Symbolic Regression Methods and their Relative Performance". In: *CoRR* abs/2107.14351 (2021). arXiv: 2107.14351. URL: https://arxiv.org/abs/2107.14351.

[4] William La Cava, Thomas Helmuth, Lee Spector, and Jason H. Moore. *A probabilistic and multi-objective analysis of lexicase selection and epsilon-lexicase selection*. 2018. arXiv: 1709.05394 [cs.NE].

[5] William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason H. Moore. *Learning concise representations for regression by evolving networks of trees*. 2019. arXiv: 1807.00981 [cs.NE].

[6] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[7] David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. *Scalable Global Optimization via Local Bayesian Optimization*. 2020. arXiv: 1910.01739 [cs.LG].

[8] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009". In: *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: Association for

Computing Machinery, 2010, pp. 1689–1696. ISBN: 9781450300735. DOI: 10.1145/1830761.1830790. URL: https://doi.org/10.1145/1830761.1830790.

[9] Nikolaus Hansen, Dimo Brockhoff, Olaf Mersmann, Tea Tusar, Dejan Tusar, Ouassim Ait ElHara, Phillipe R. Sampaio, Asma Atamna, Konstantinos Varelas, Umut Batu, Duc Manh Nguyen, Filip Matzner, and Anne Auger. *COmparing Continuous Optimizers: numbbo/COCO on Github*. Version v2.3. Mar. 2019. DOI: 10.5281/zenodo.2594848. URL: https://doi.org/10.5281/zenodo.2594848.

[10] Heuristic and Evolutionary Algorithms Laboratory. *PyOperon*. 2023. URL: https://github.com/heal-research/pyoperon.

[11] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. *Bayesian Symbolic Regression*. 2020. arXiv: 1910.08892 [stat.ME].

[12] Donald Jones, Matthias Schonlau, and William Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13 (Dec. 1998), pp. 455–492. DOI: 10.1023/A:1008306431147.

[13] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[14] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise". In: *Journal of Basic Engineering* 86.1 (Mar. 1964), pp. 97–106. ISSN: 0021-9223. DOI: 10.1115/1.3653121. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/86/1/97/5763745/97_1.pdf. URL: https://doi.org/10.1115/1.3653121.

[15] Juliane Mueller. *MATSuMoTo: The MATLAB Surrogate Model Toolbox For Computationally Expensive Black-Box Global Optimization Problems*. 2014. arXiv: 1404.4261 [math.OC].

[16] Ji Ni, Russ H. Drieberg, and Peter I. Rockett. "The Use of an Analytic Quotient Operator in Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 146–152. DOI: 10.1109/TEVC.2012.2195319.

[17] Miguel Nicolau and Alexandros Agapitos. "Choosing function sets with better generalisation performance for symbolic regression models". In: *Genetic Programming and Evolvable Machines* 22 (Mar. 2021). DOI: 10.1007/s10710-020-09391-4.

[18] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. "PMLB: a large benchmark suite for machine learning evaluation and comparison". In: *BioData Mining* 10.1 (Dec. 2017), p. 36. ISSN: 1756-0381. DOI: 10.1186/s13040-017-0154-4. URL: https://doi.org/10.1186/s13040-017-0154-4.

[19] Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=m5Qsh0kBQG.

[20] Joseph D Romano, Trang T Le, William La Cava, John T Gregg, Daniel J Goldberg, Praneel Chakraborty, Natasha L Ray, Daniel Himmelstein, Weixuan Fu, and Jason H Moore. "PMLB v1.0: an open source dataset collection for benchmarking machine learning methods". In: *arXiv preprint arXiv:2012.00058v2* (2021).

[21] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[22] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting". In: *IEEE Transactions on Information Theory* 58.5 (May 2012), pp. 3250–3265. ISSN: 1557-9654. DOI: 10.1109/tit.2011.2182033. URL: http://dx.doi.org/10.1109/TIT.2011.2182033.

[23] SymPy Development Team. *SymPy*. 2023. URL: https://www.sympy.org/en/index.html.

[24] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. *AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity*. 2020. arXiv: 2006.10782 [cs.LG].

[25] Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1084–1092. ISBN: 9781450361118. DOI: 10.1145/3321707.3321758. URL: https://doi.org/10.1145/3321707.3321758.

Appendix A

Experimental Results

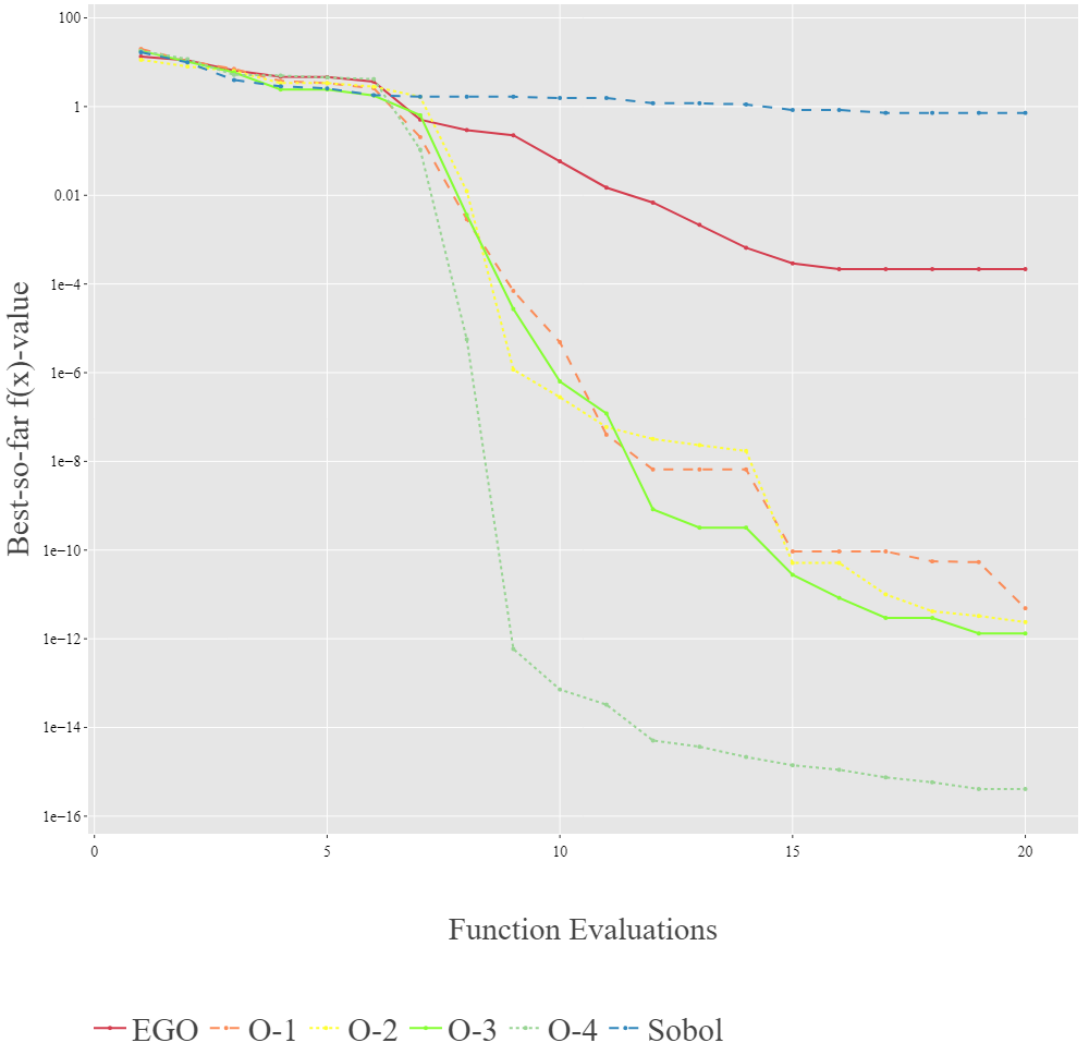

Figure A.1: Results for the 2-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 6 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

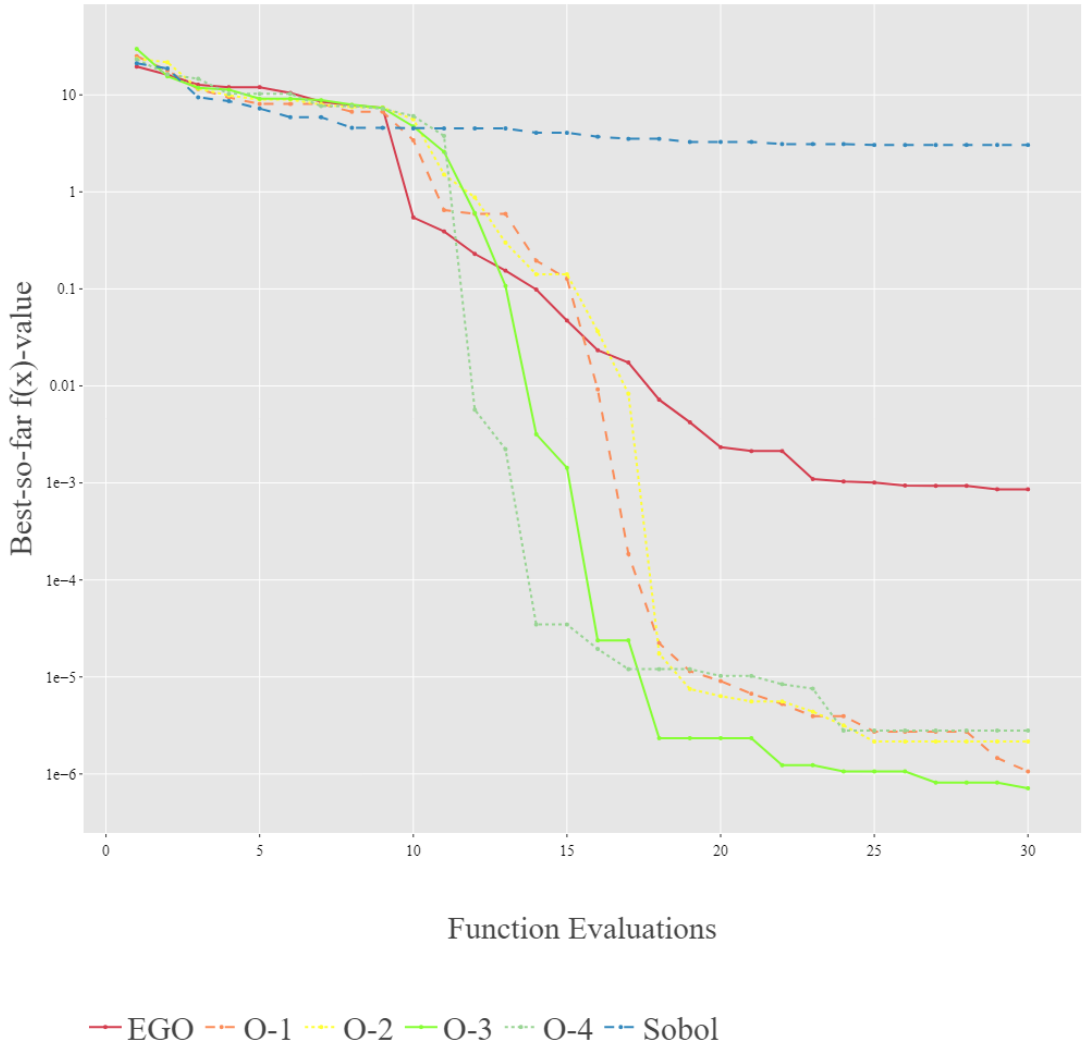

Figure A.2: Results for the 3-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 9 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

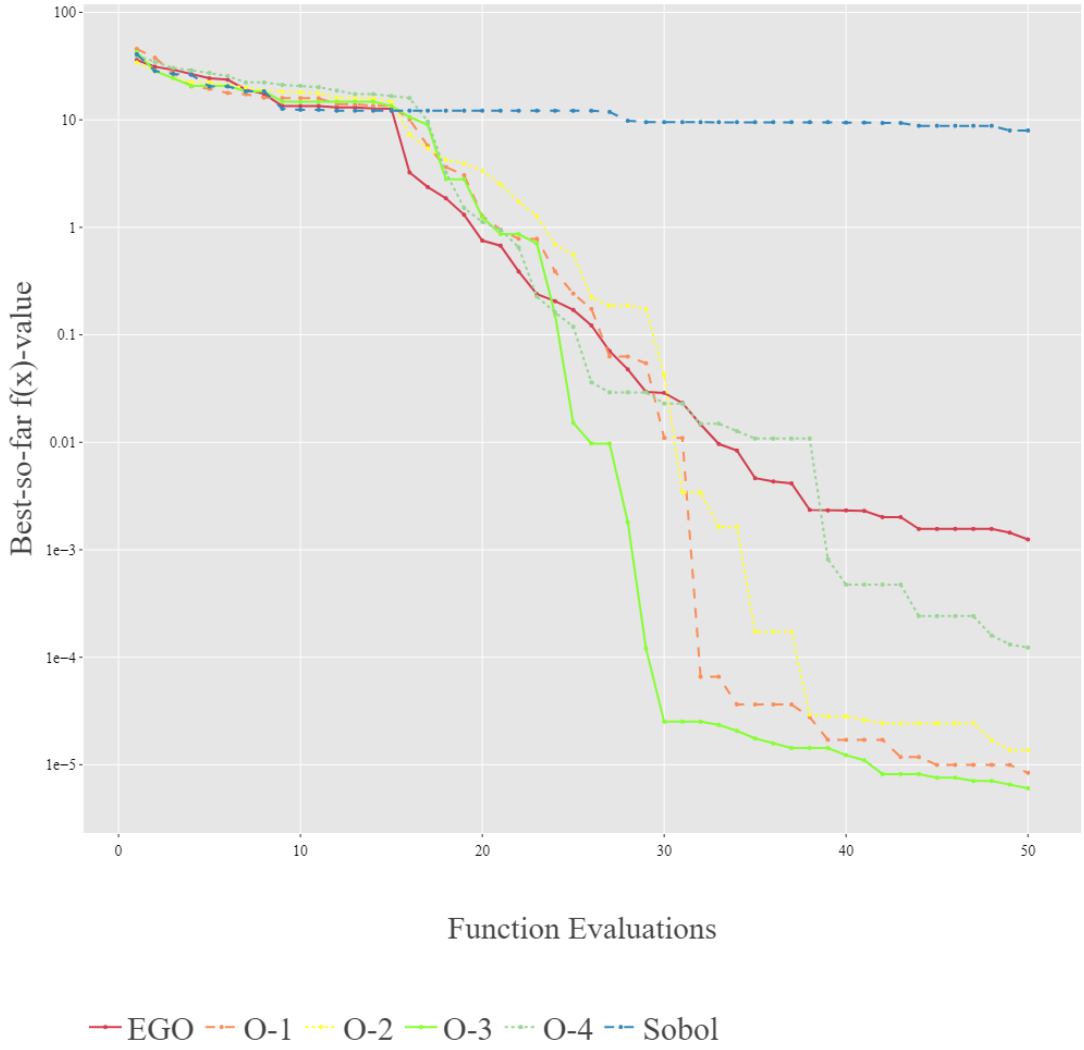

Figure A.3: Results for the 5-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 15 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

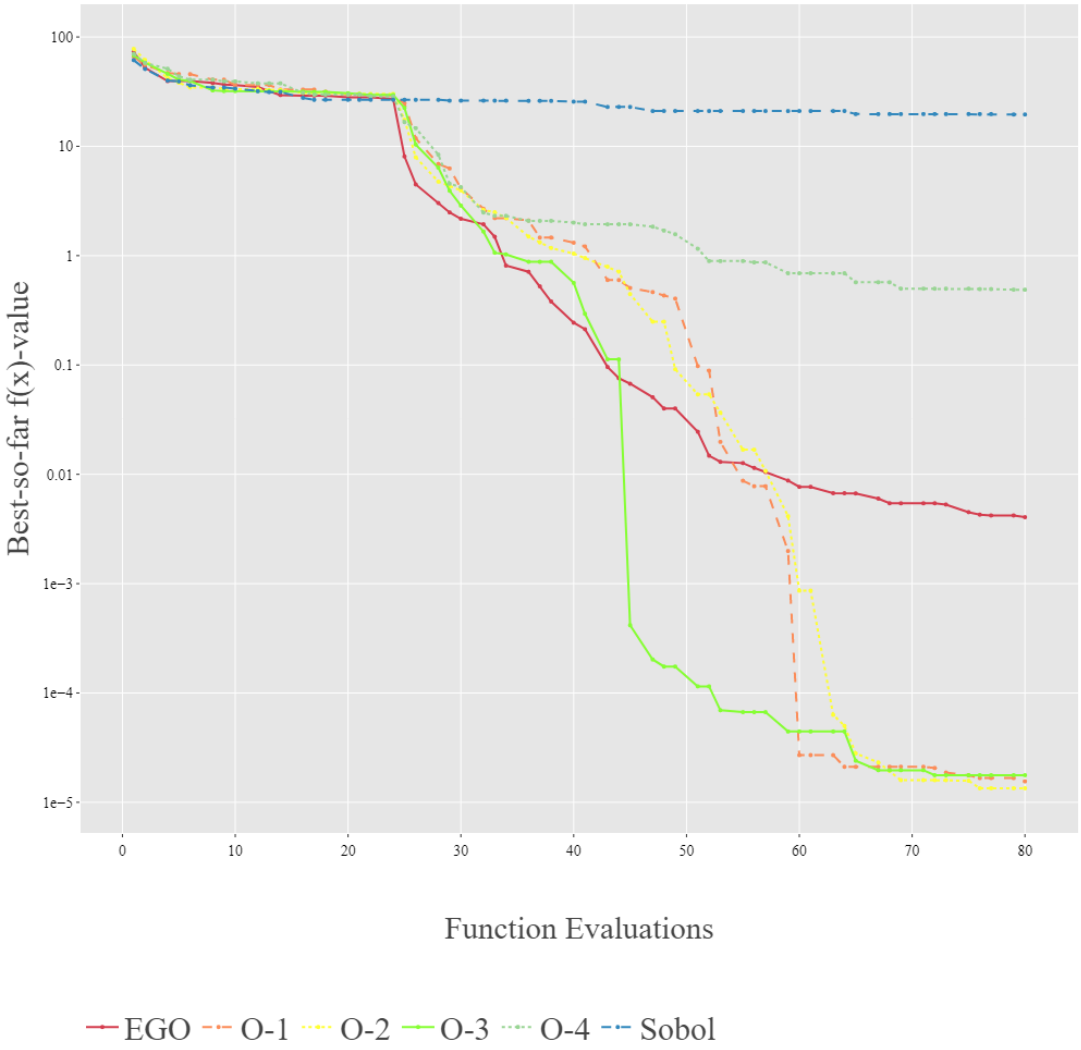

Figure A.4: Results for the 8-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 24 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

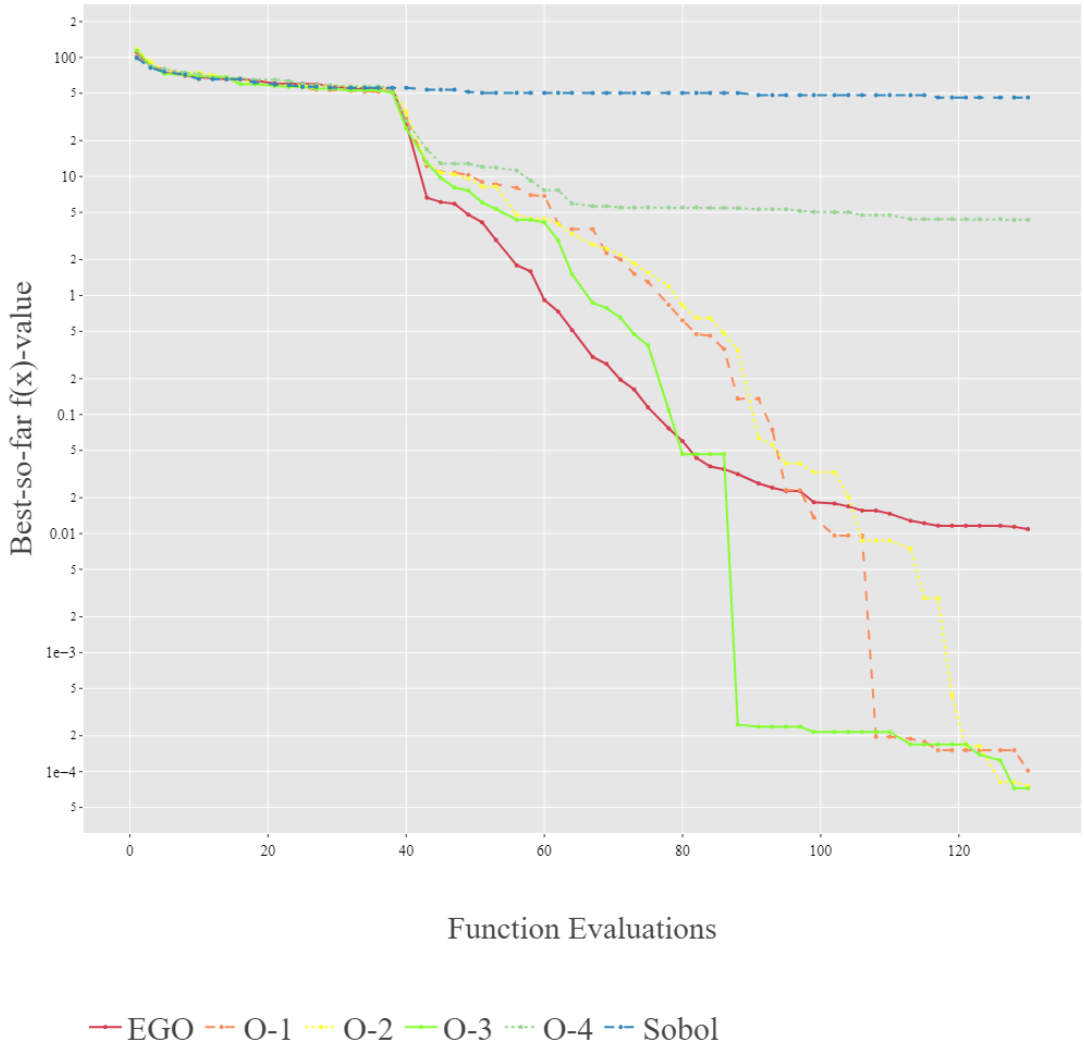

Figure A.5: Results for the 13-dimensional sphere function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

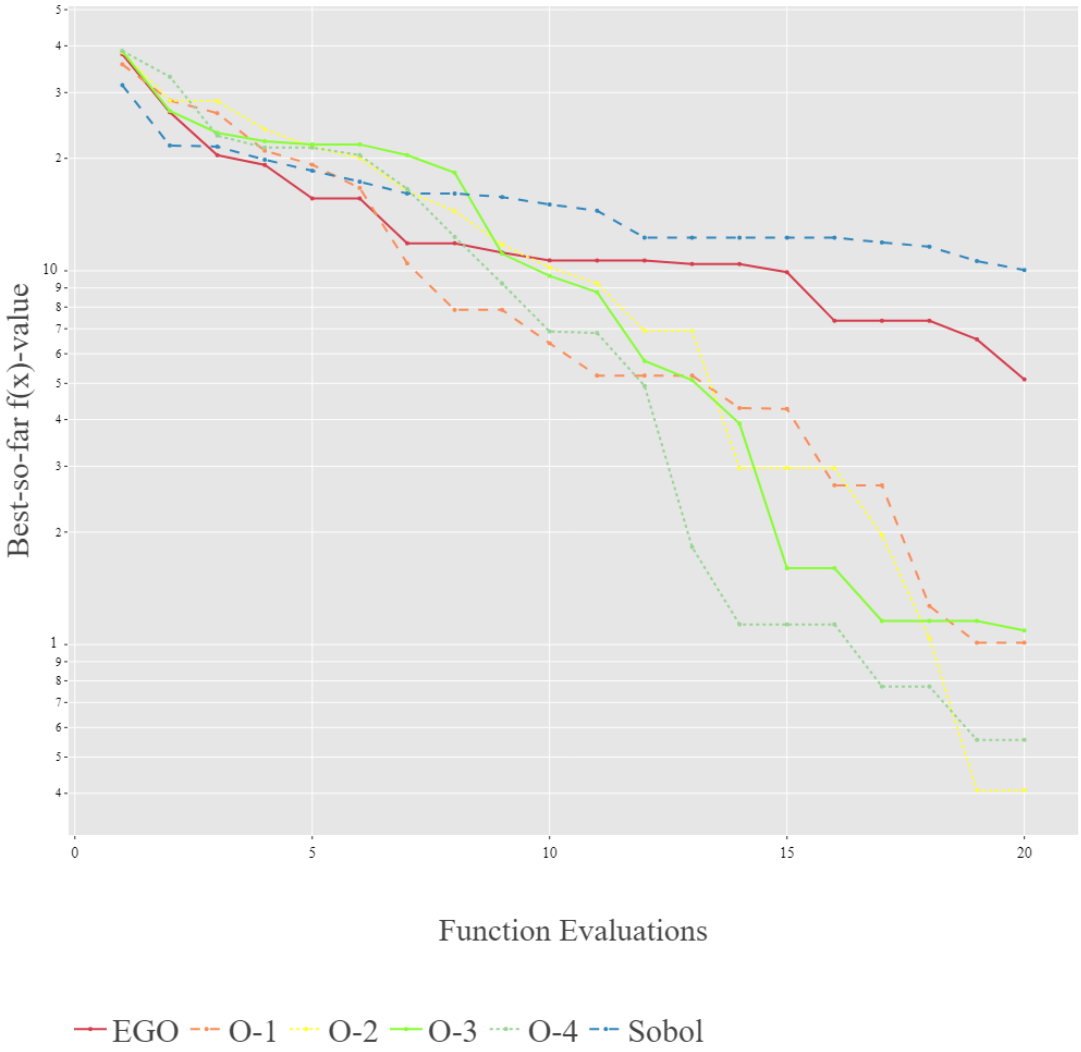

Figure A.6: Results for the 2-dimensional Rastrigin function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 6 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

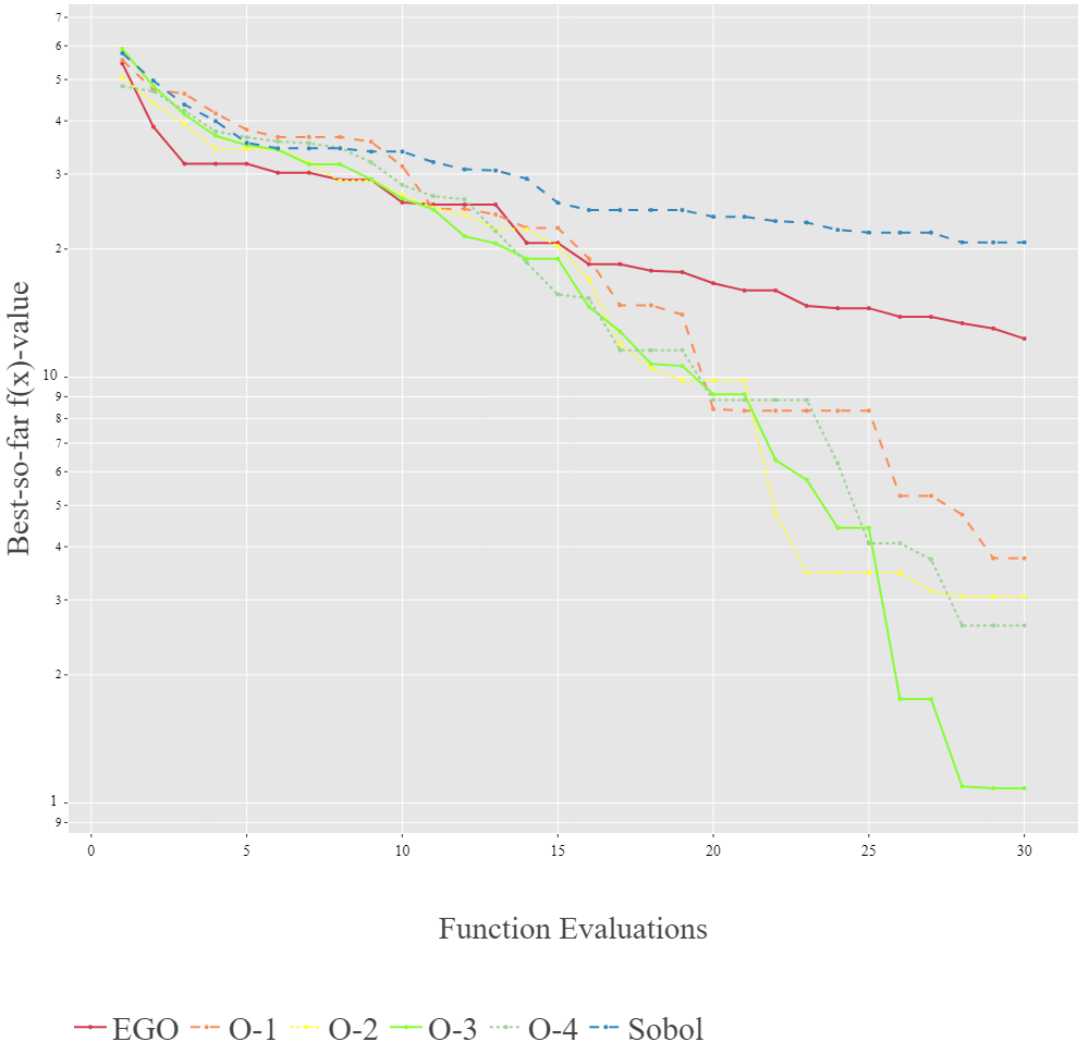

Figure A.7: Results for the 3-dimensional Rastrigin function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 9 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

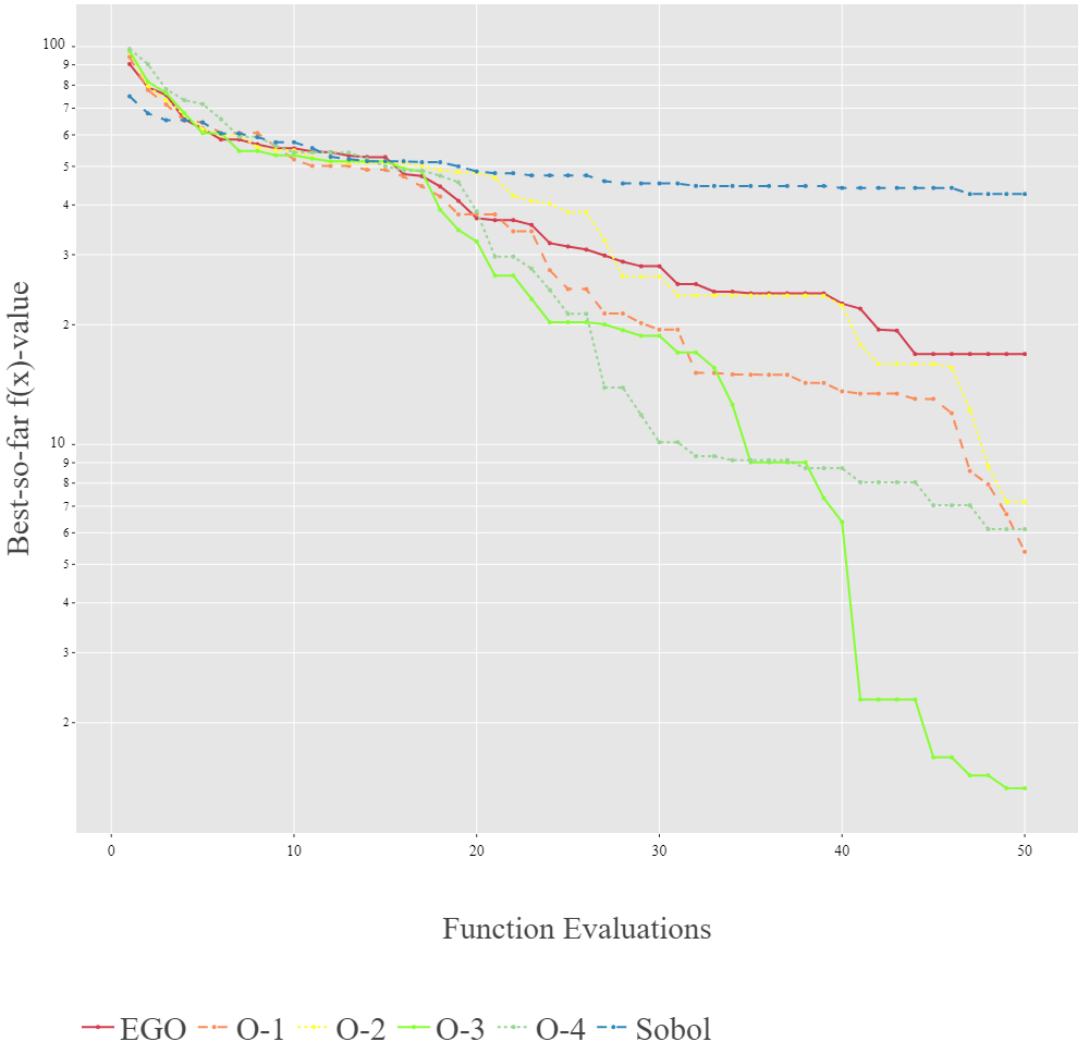

Figure A.8: Results for the 5-dimensional Rastrigin function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 15 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

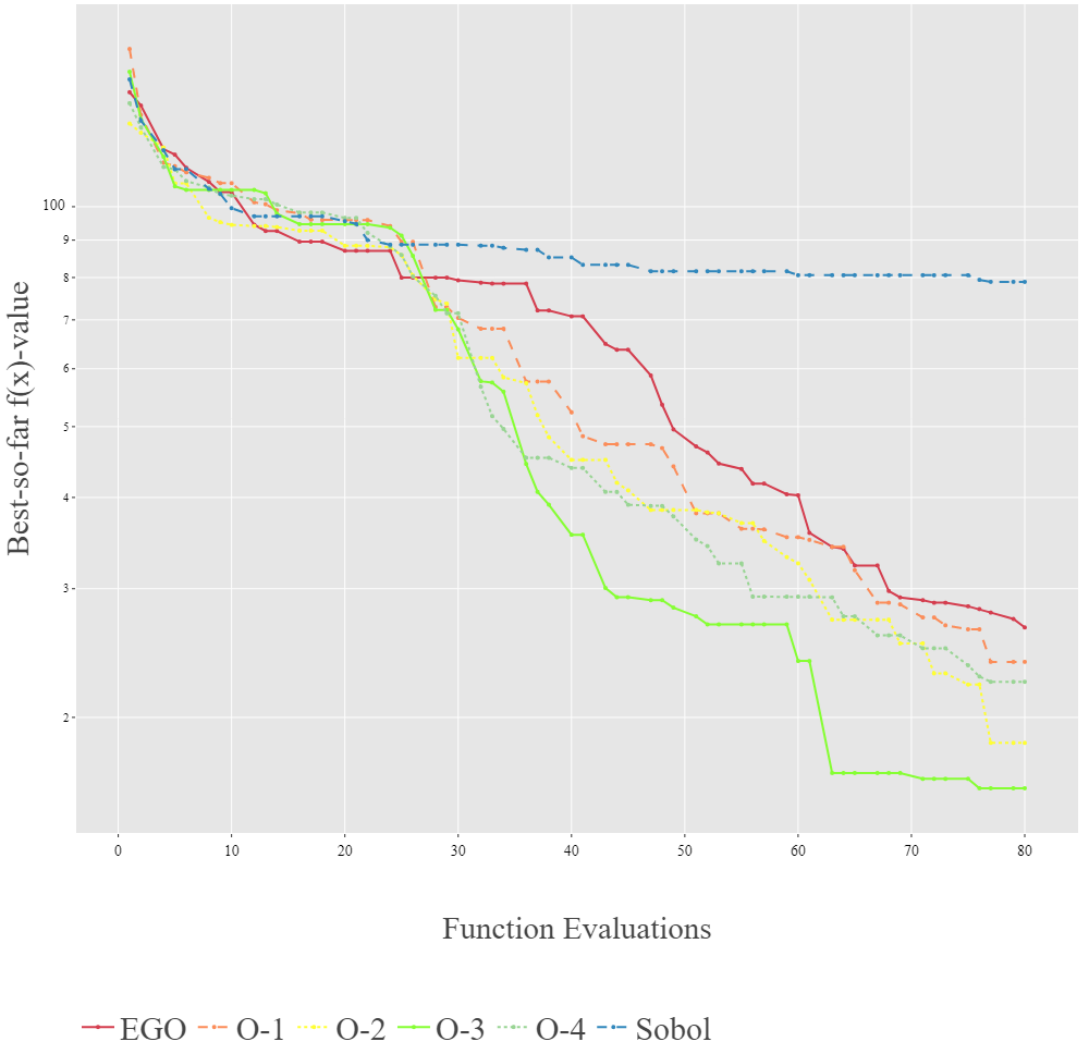

Figure A.9: Results for the 8-dimensional Rastrigin function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 24 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

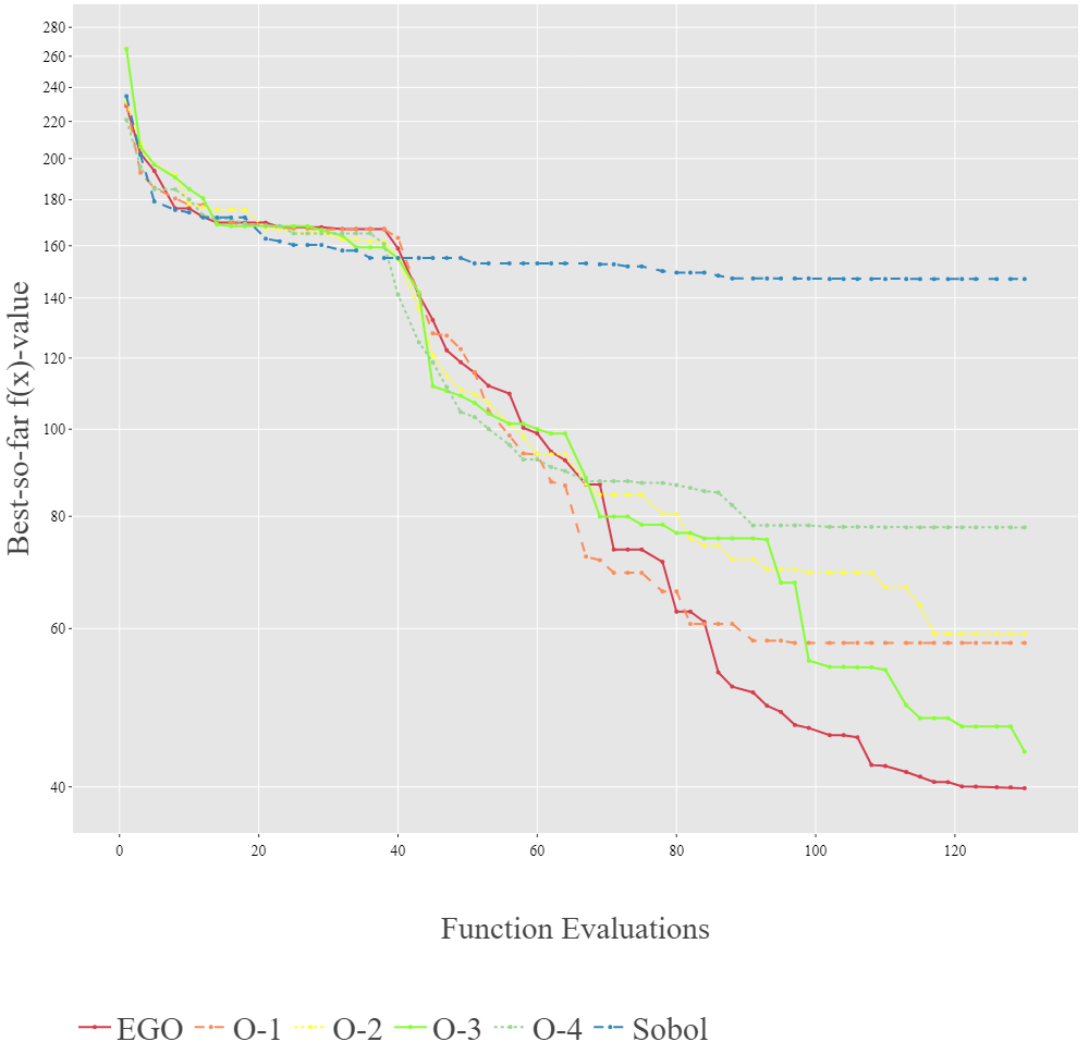

Figure A.10: Results for the 13-dimensional Rastrigin function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

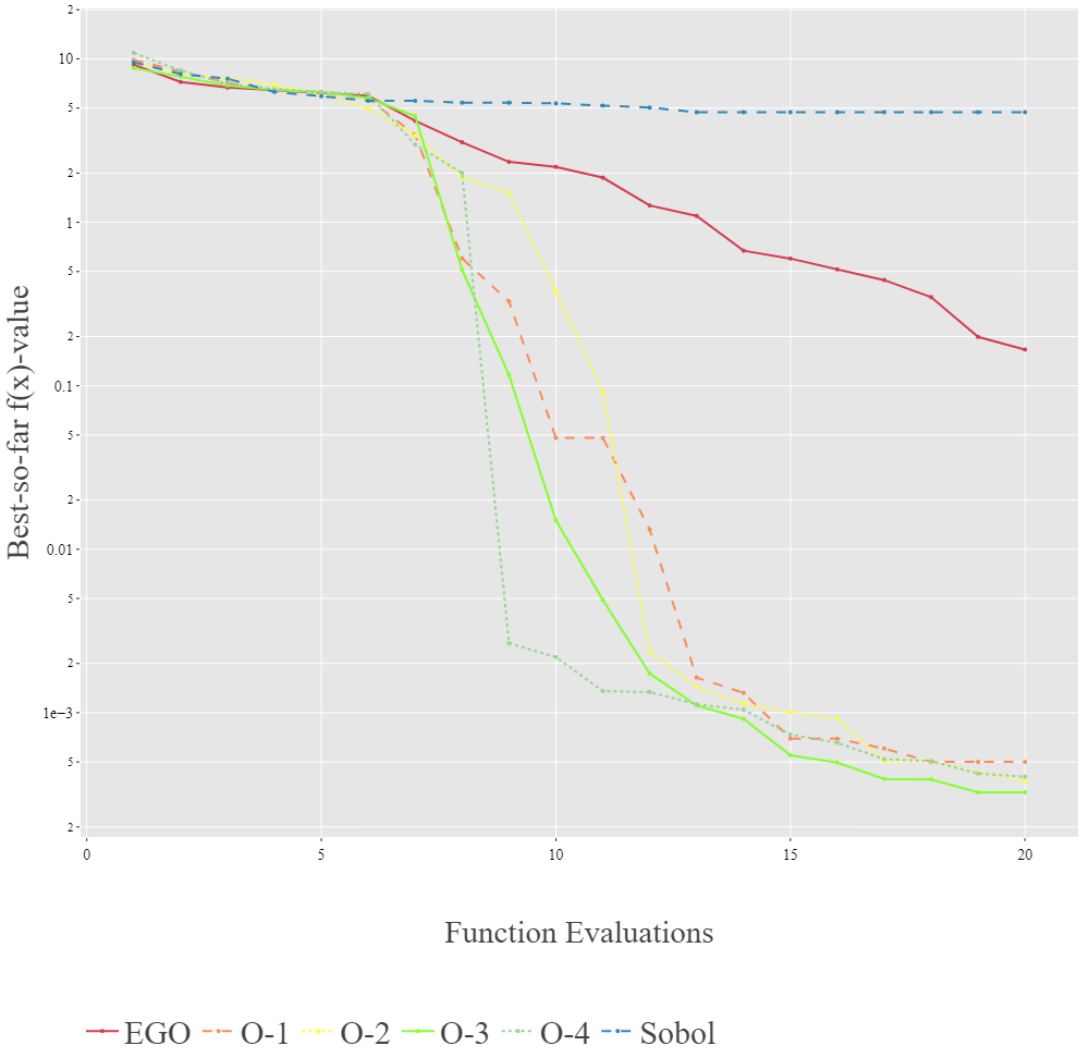

Figure A.11: Results for the 2-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 6 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

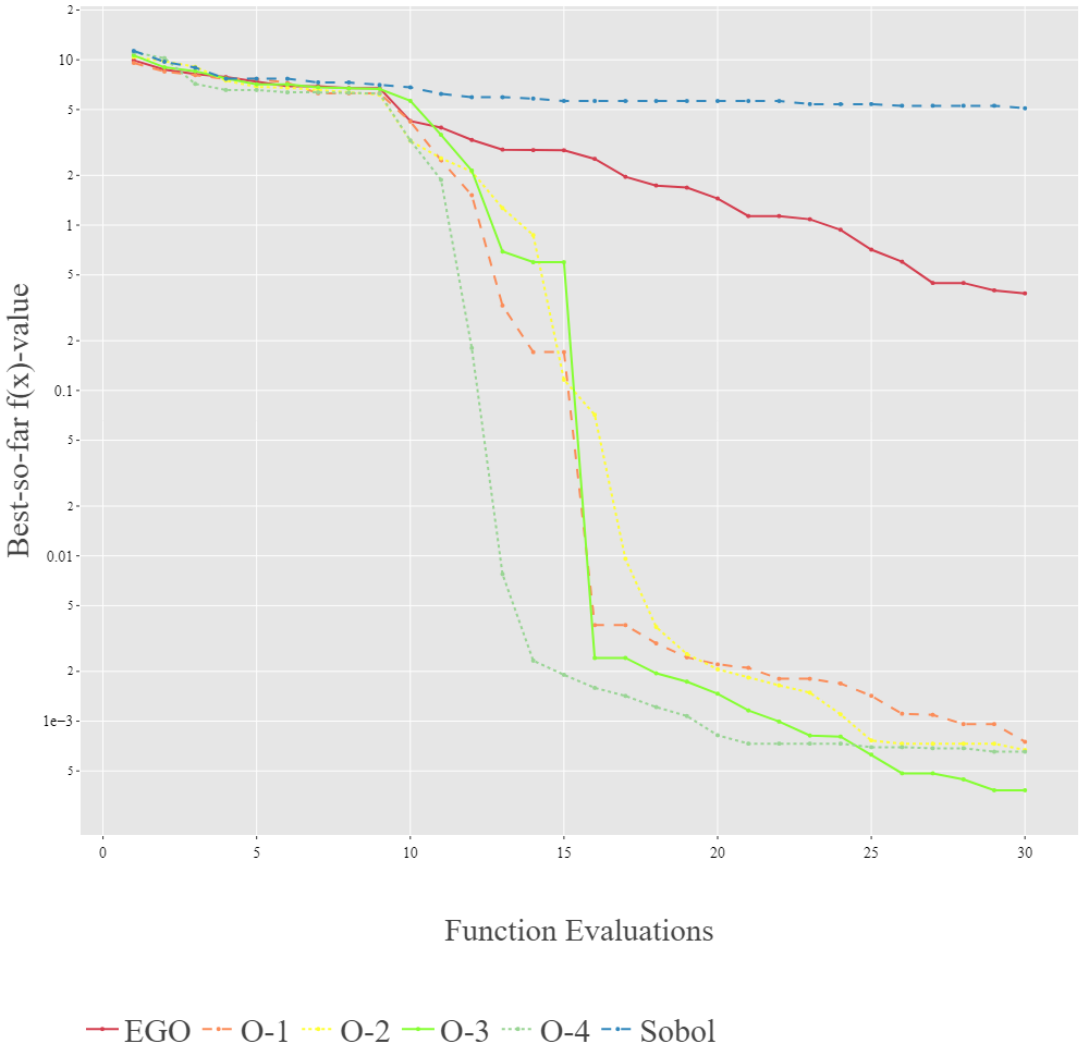

Figure A.12: Results for the 3-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 9 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

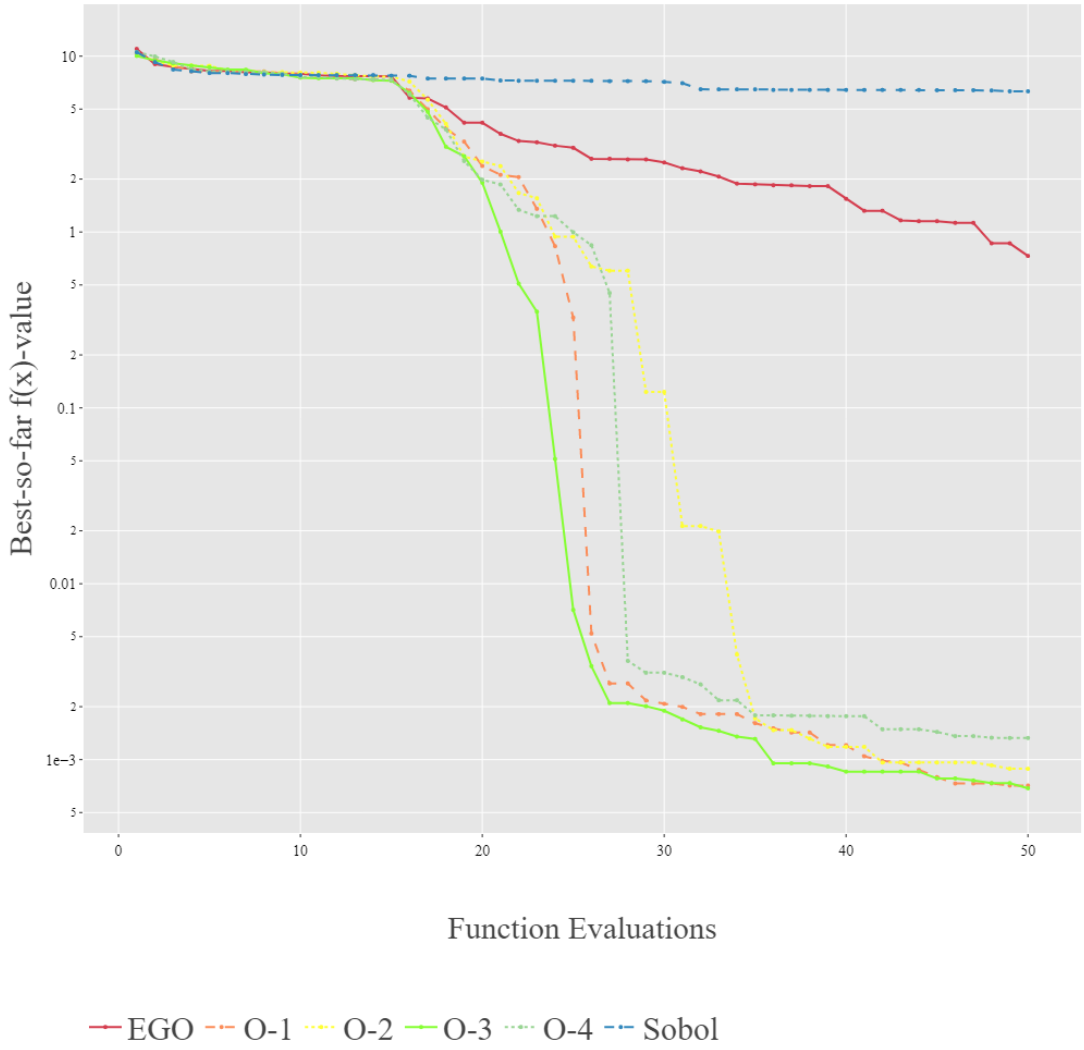

Figure A.13: Results for the 5-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 15 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

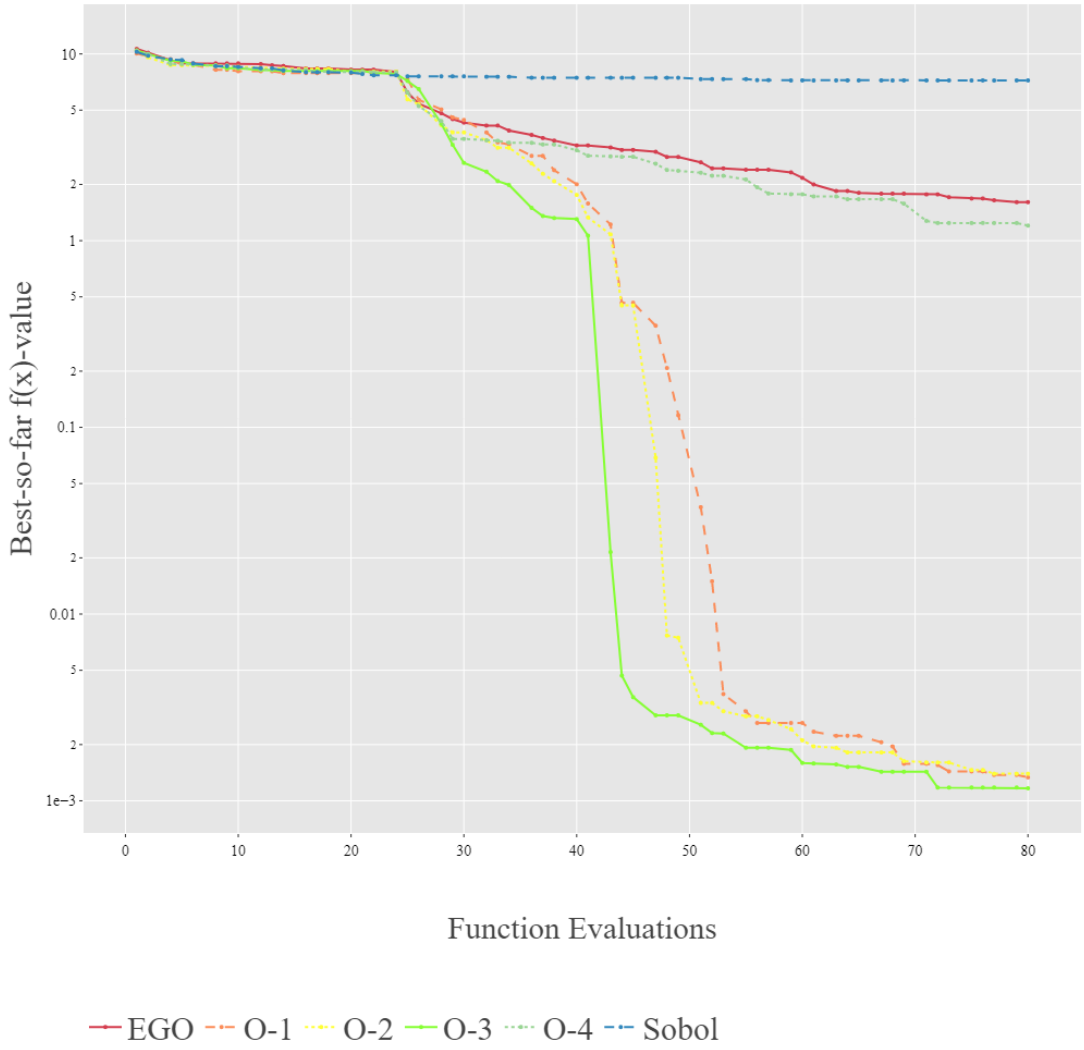

Figure A.14: Results for the 8-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 24 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

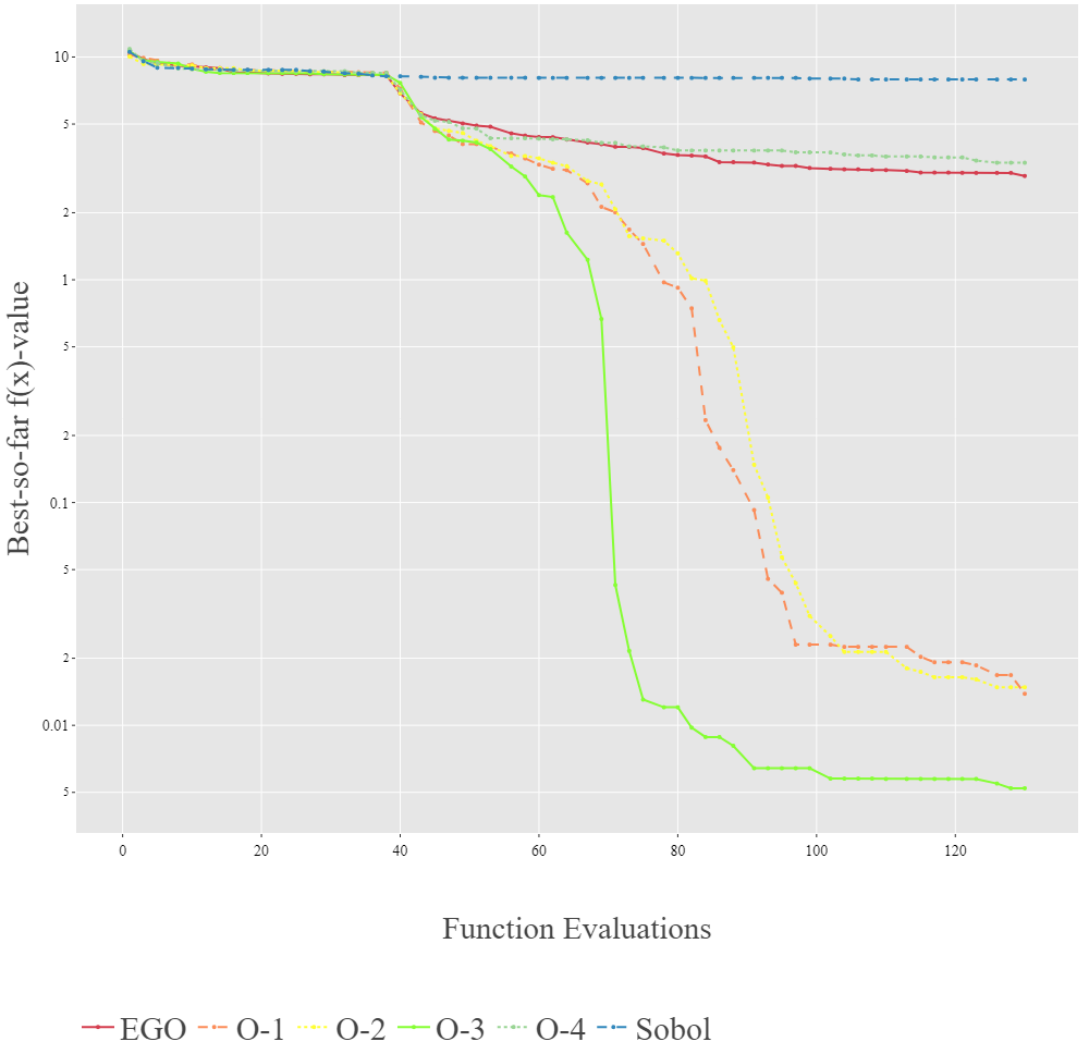

Figure A.15: Results for the 13-dimensional Ackley function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

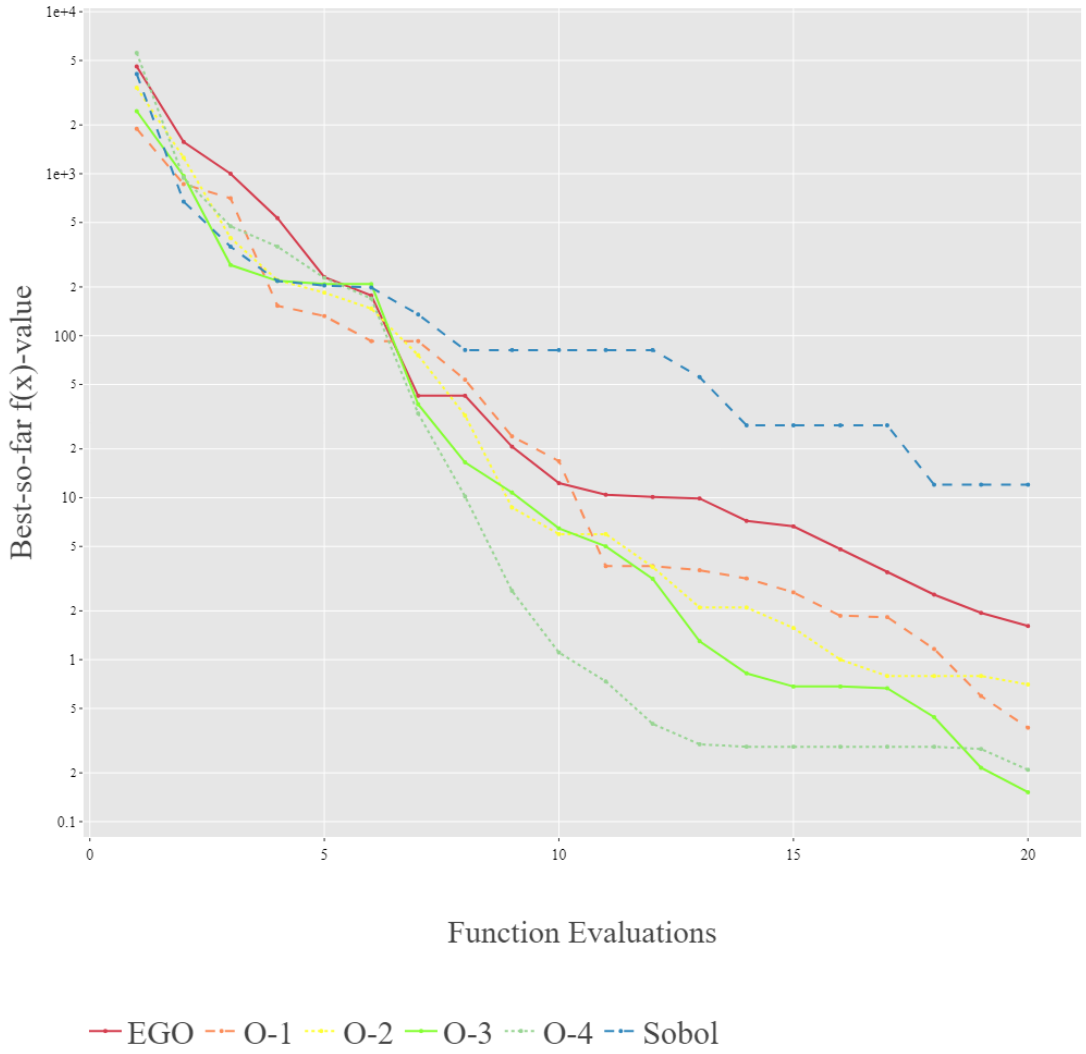

Figure A.16: Results for the 2-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 6 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

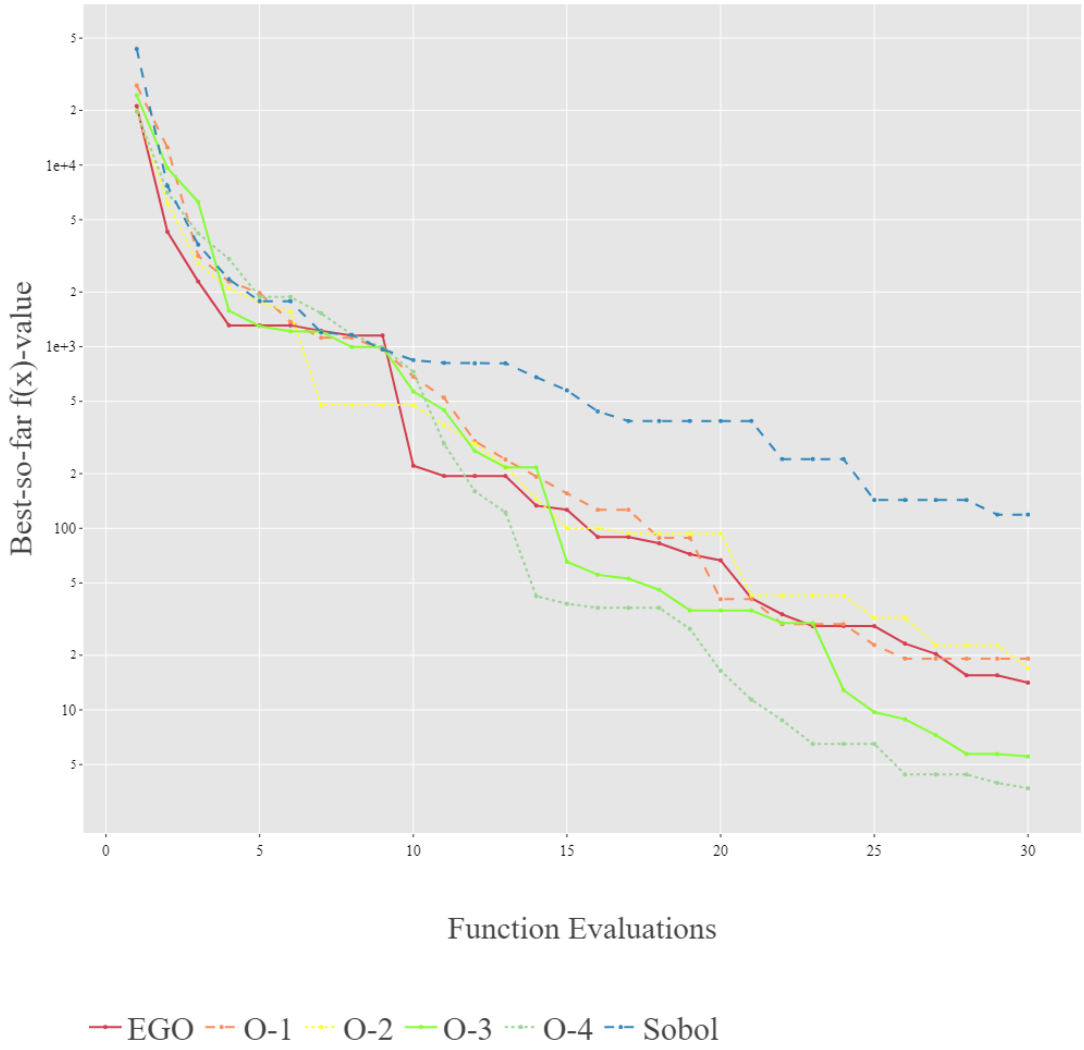

Figure A.17: Results for the 3-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 9 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

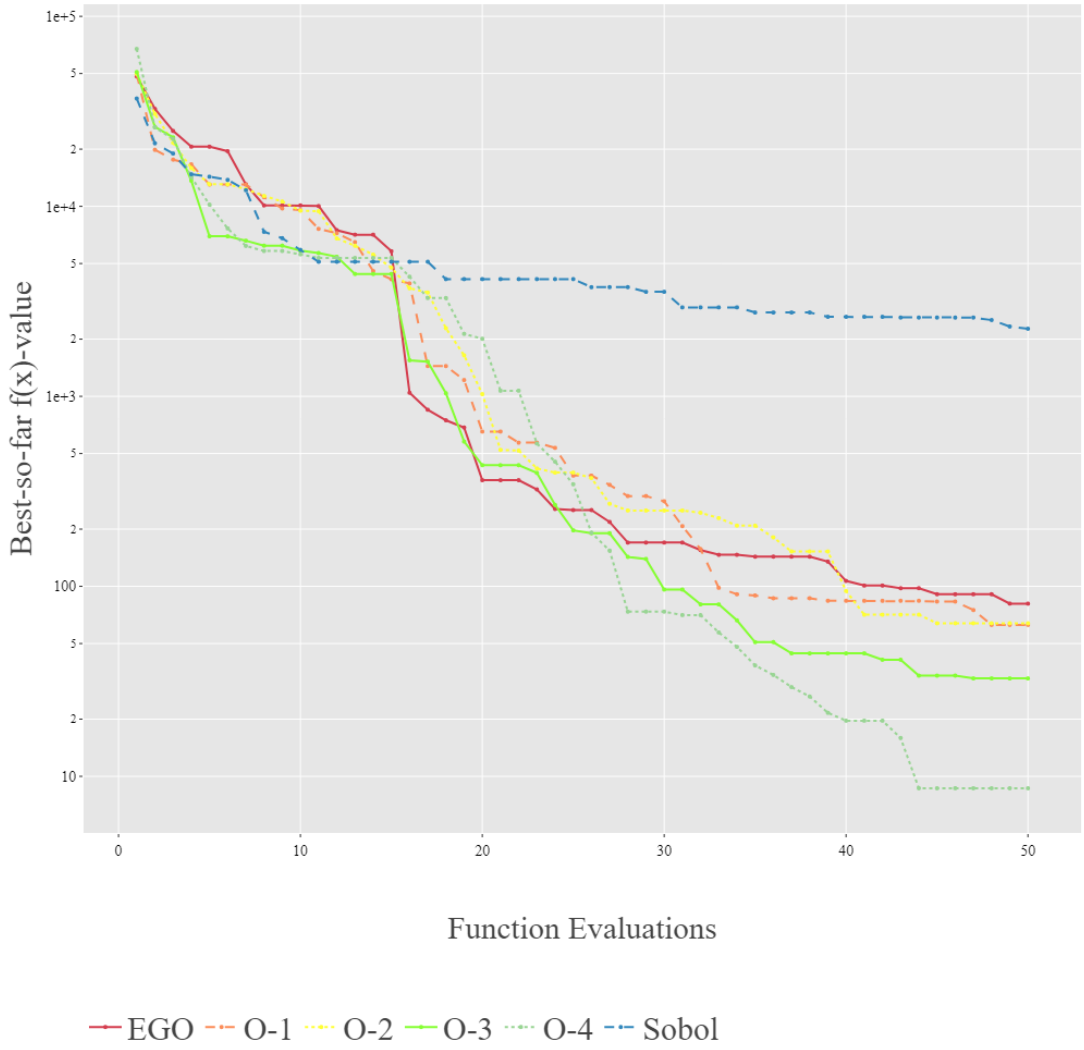

Figure A.18: Results for the 5-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 15 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

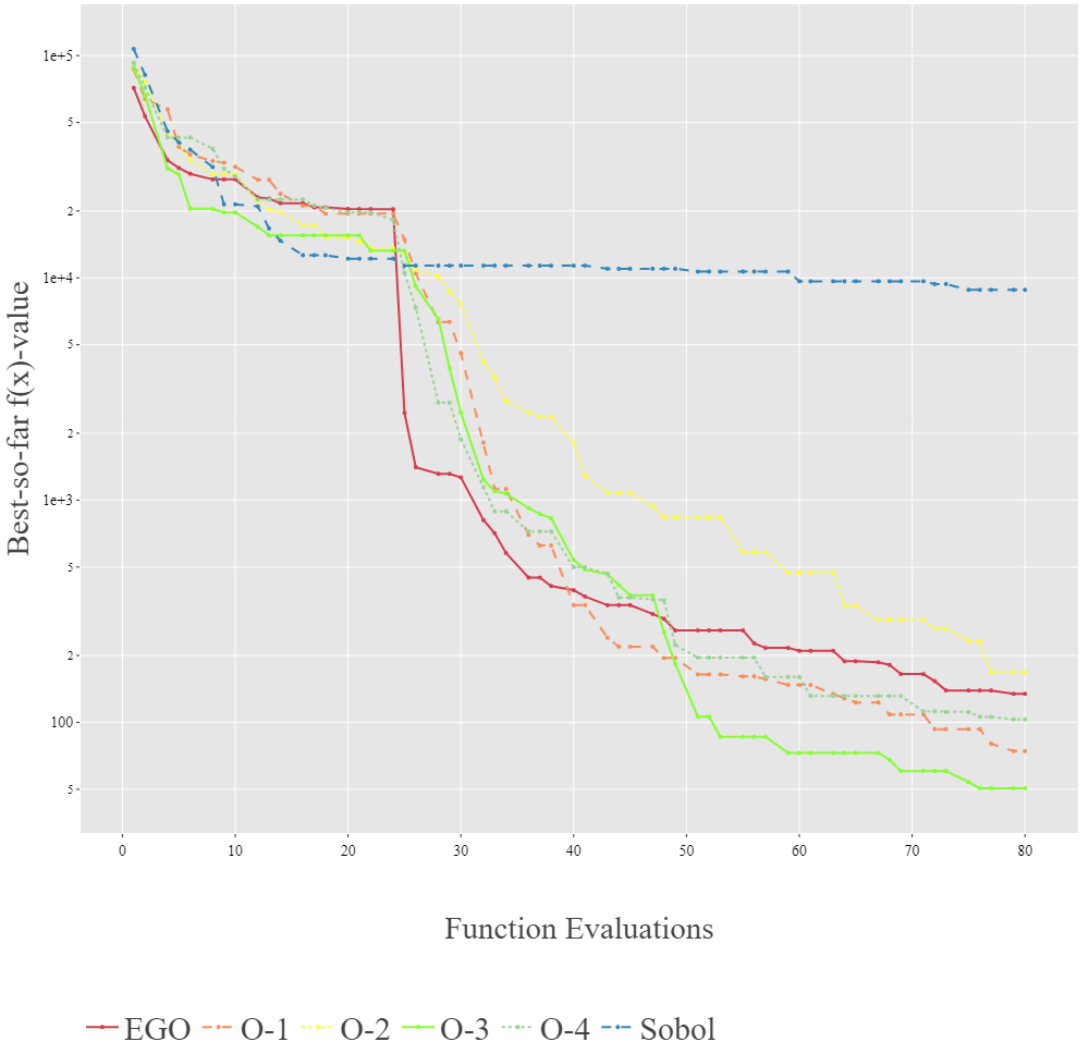

Figure A.19: Results for the 8-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 24 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

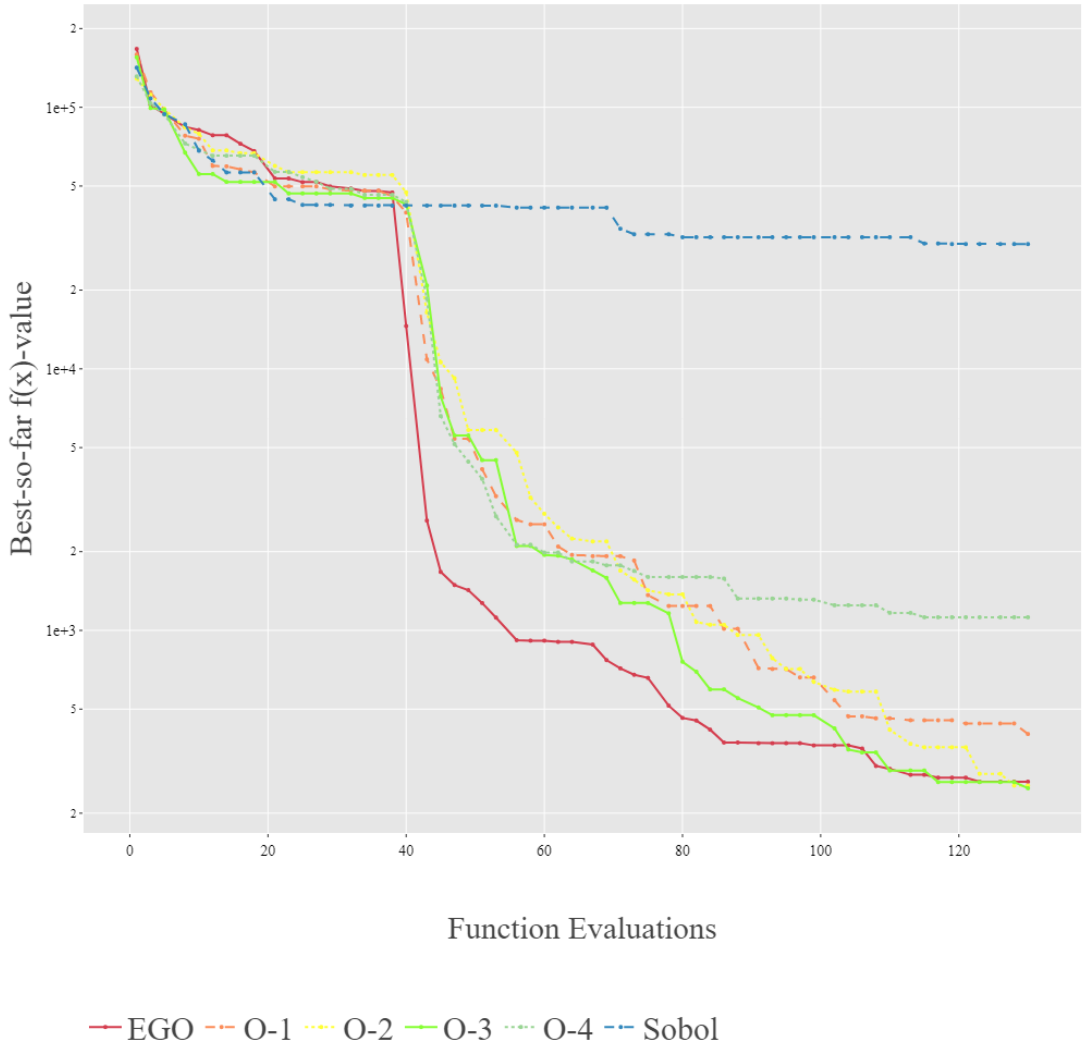

Figure A.20: Results for the 13-dimensional Rosenbrock function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using a logarithmic y-scale. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

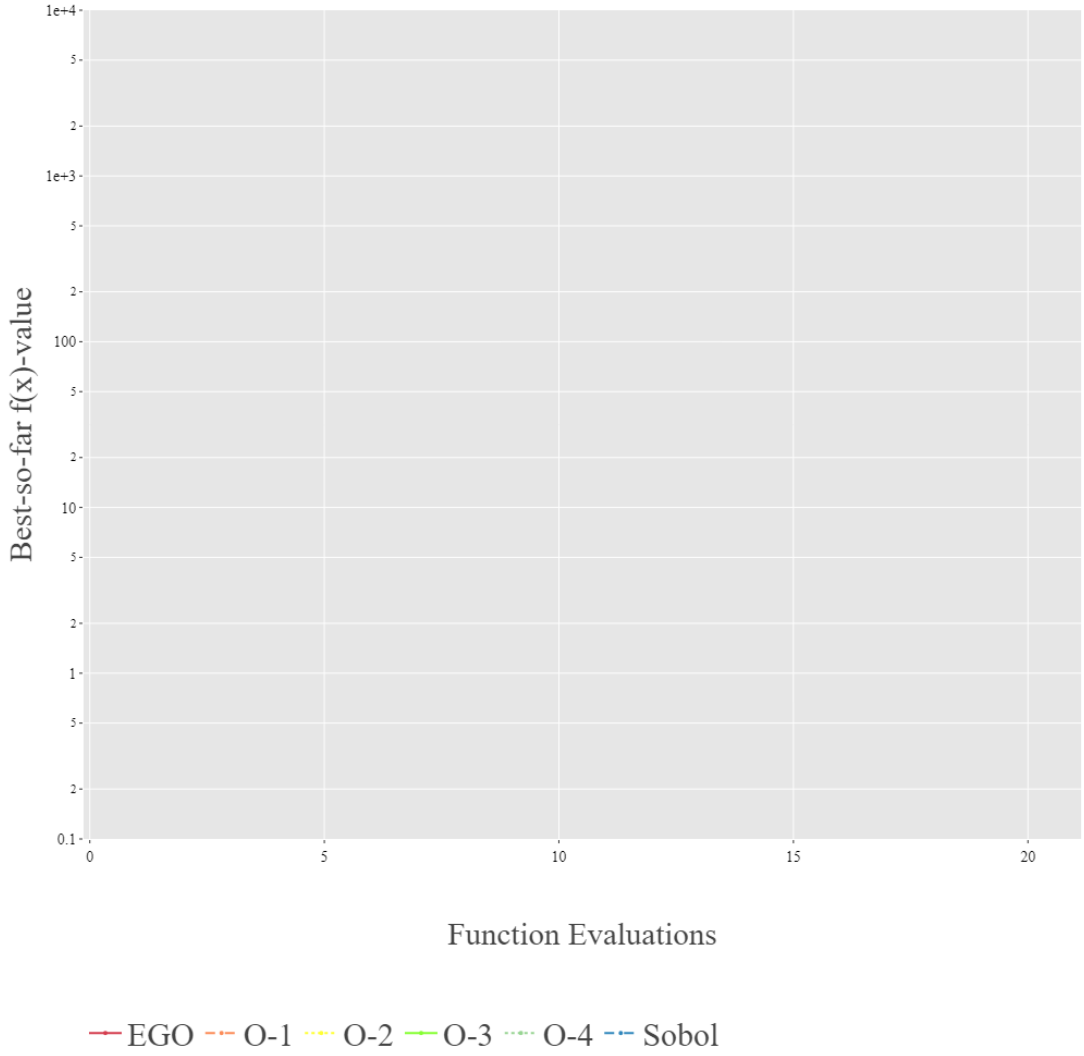

Figure A.21: Results for the 2-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs. The first 6 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

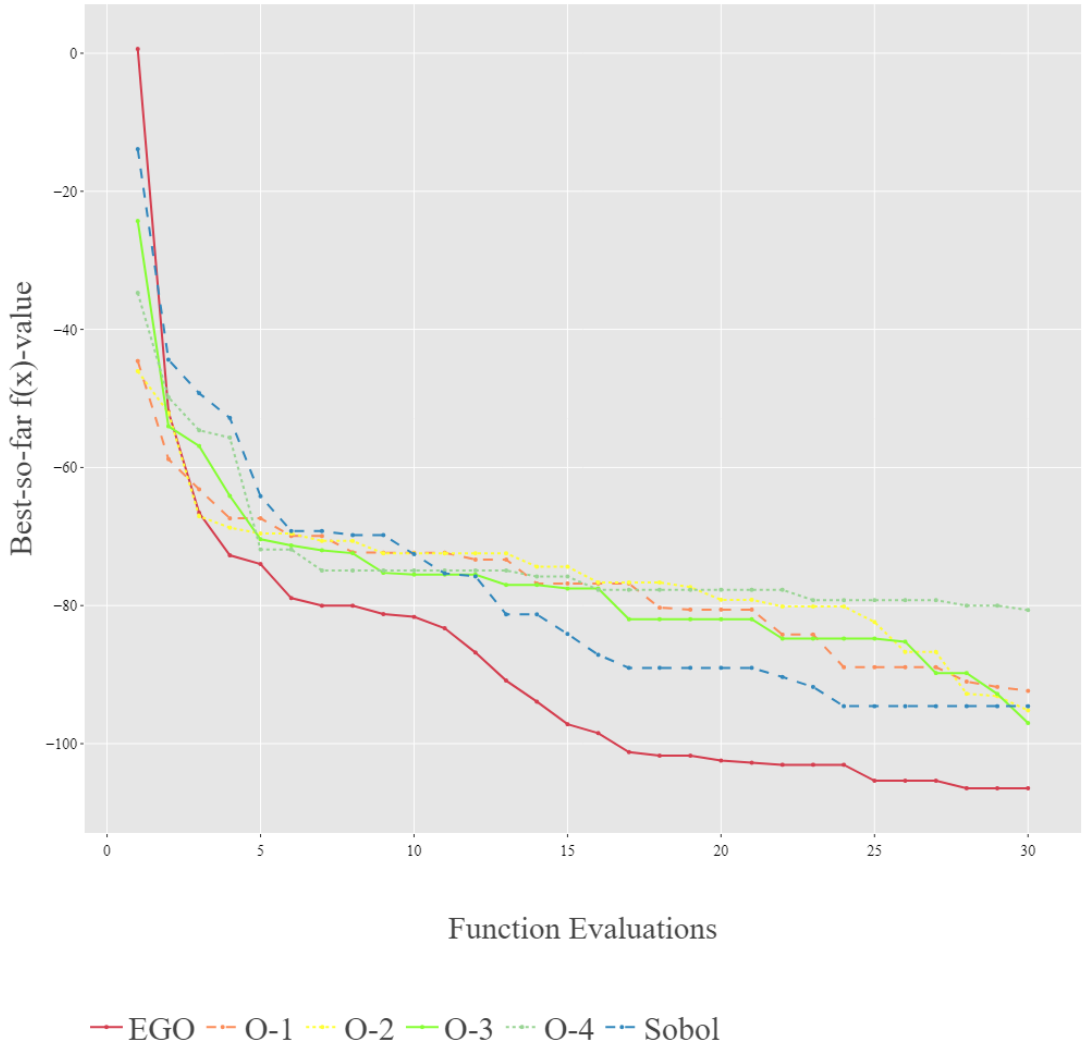

Figure A.22: Results for the 3-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs. The first 9 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

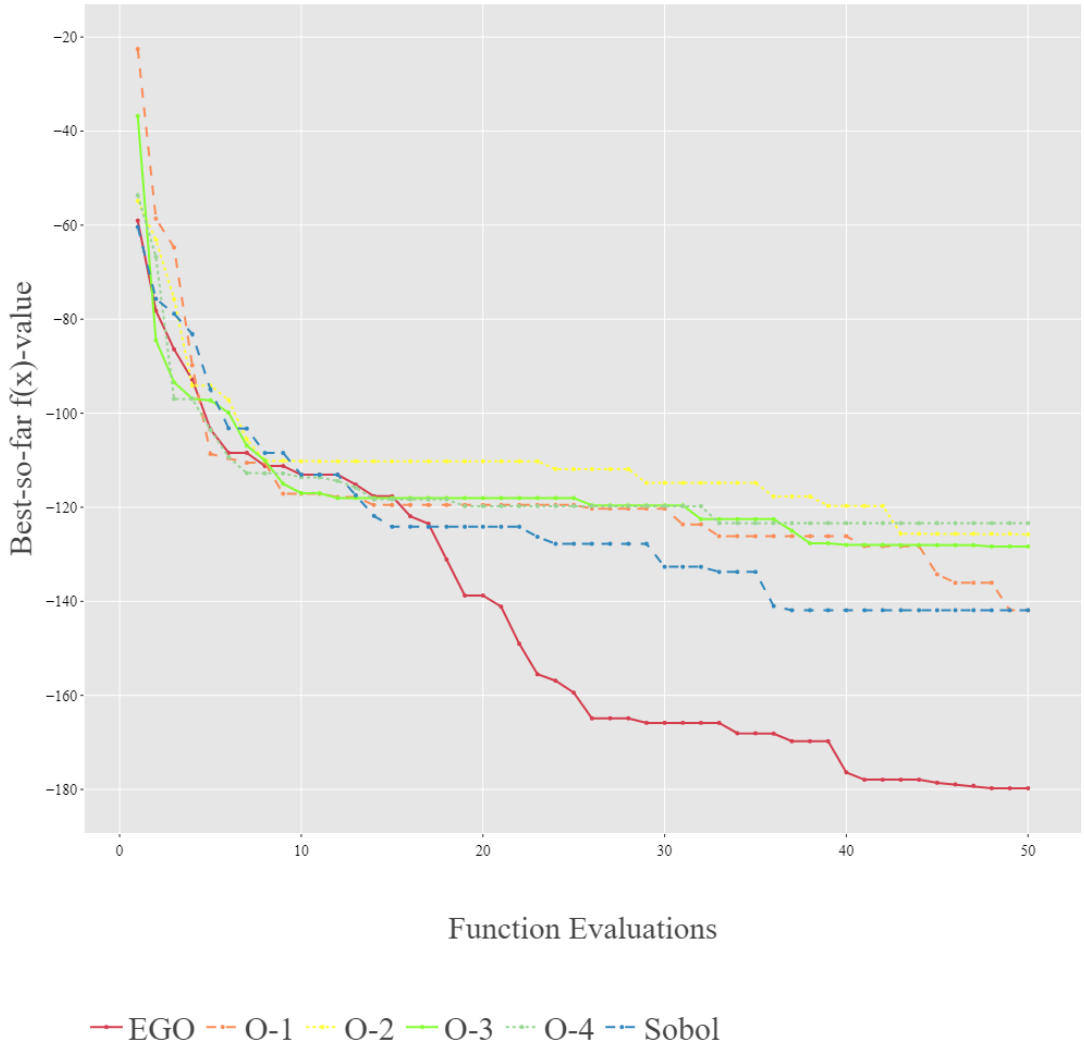

Figure A.23: Results for the 5-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs. The first 15 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

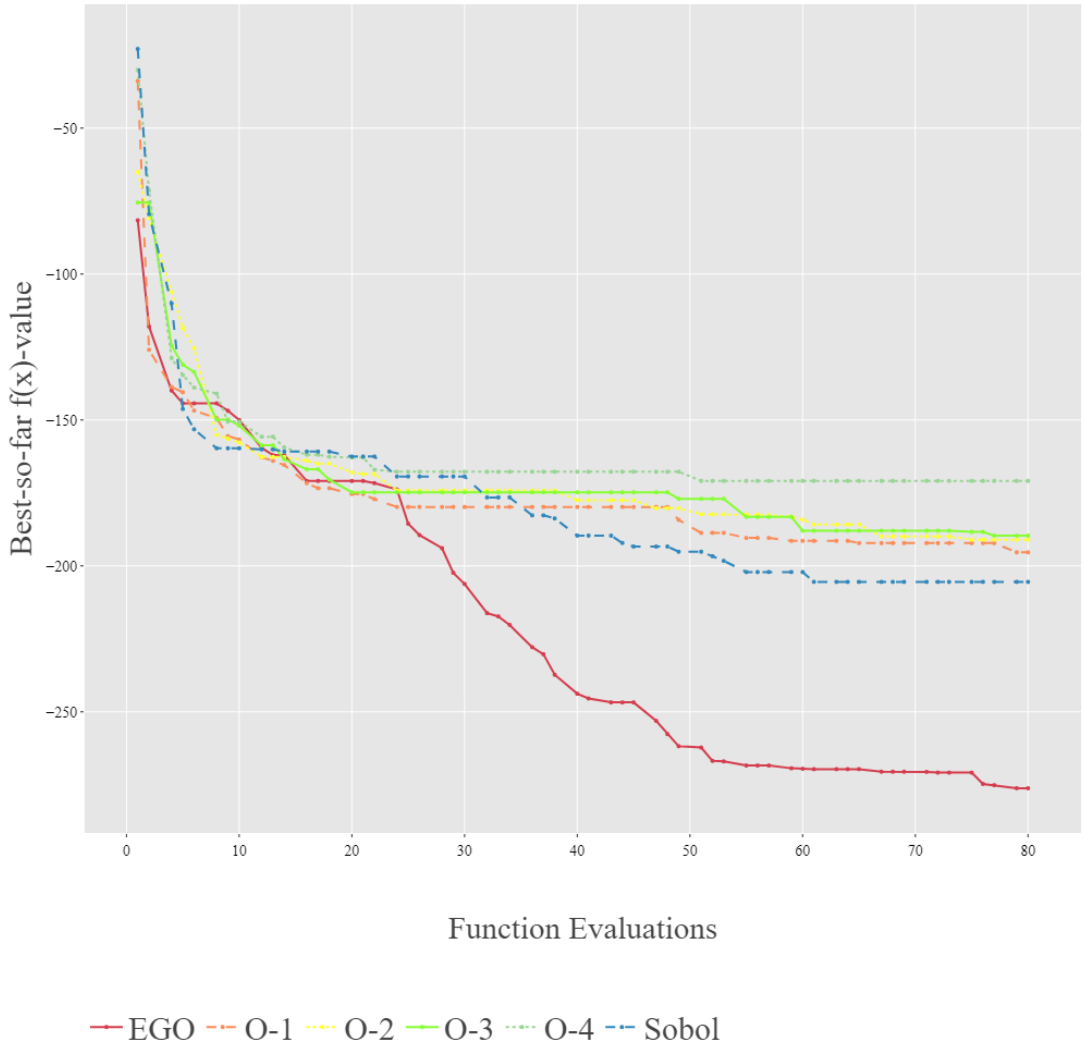

Figure A.24: Results for the 8-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs using. The first 24 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.

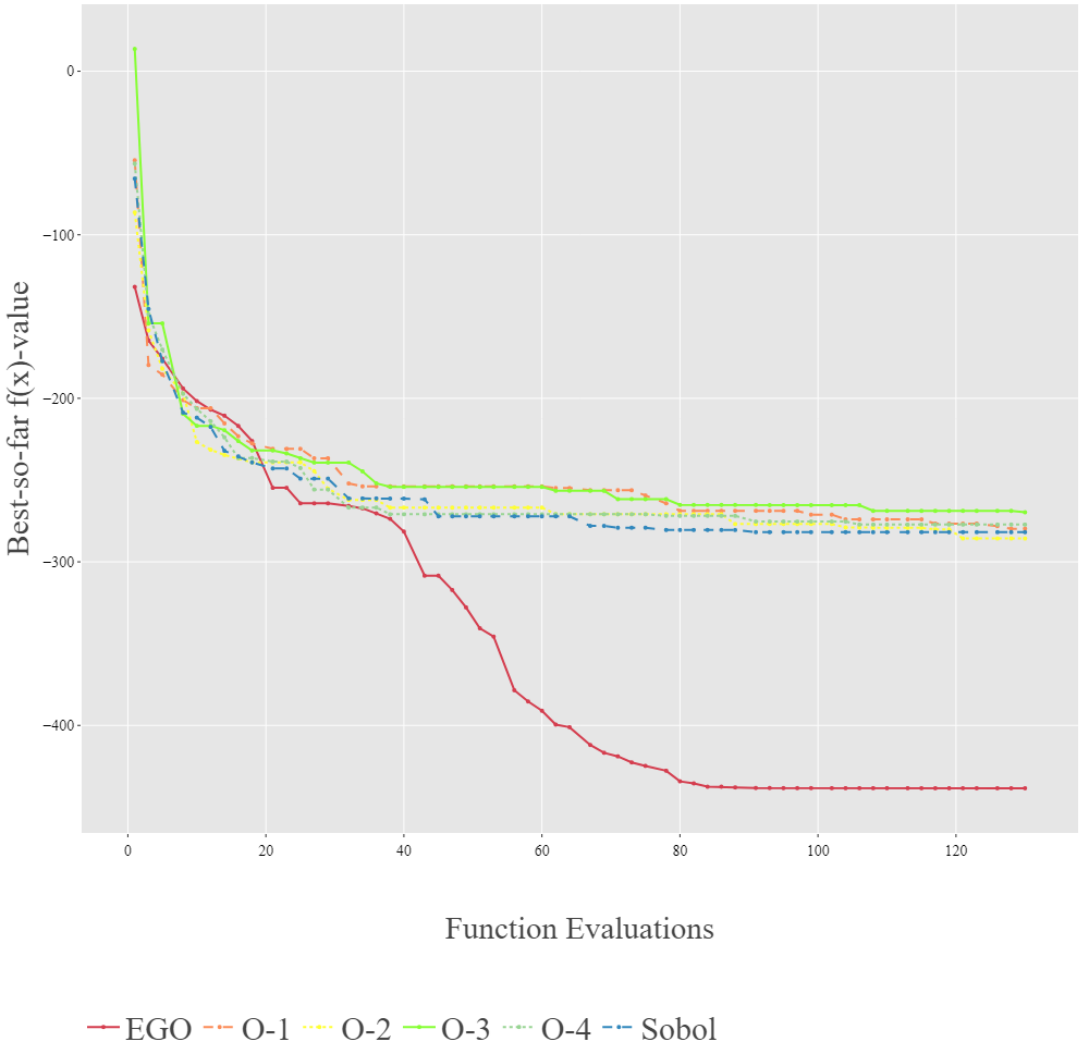

Figure A.25: Results for the 13-dimensional Styblinski-Tang function. Curves indicate the median found optimum after each objective function evaluation over 20 independent runs. The first 39 values are chosen through Sobol sampling in all cases, after which an SMBO algorithm takes over.