Universiteit Leiden
The Netherlands

Data Science and Artificial Intelligence

Comparing the Performance of the Walsh and Multilinear Basis with Regression Surrogate Models for Pseudo-Boolean Optimization

Ella Kennon

Supervisors:
Dr. Thomas Bäck & Dr. Furong Ye

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 01/07/2025

Abstract

This report evaluates four different regression surrogates - polynomial regression, least-angle regression, Lasso regression and LassoLars regression - on both a Walsh and multilinear basis. These models were applied on pseudo-Boolean functions provided by IOHprofiler, where the performance of a model's approximation was measured by mean absolute error. After comparing the approximation results, this report found that the Walsh basis always outperforms the multilinear basis. The surrogates were optimized on both the Walsh and multilinear basis, using Hamming-ball efficient hill-climber optimization. After a review of the results, this report concluded that the Lasso and LassoLars Walsh surrogates were the overall best performing models of those tested.

Contents

1 Introduction

Psuedo-Boolean Optimization (PBO) problems represented in polynomial form have a wide range of implications from economics to manufacturing [BH02]. These problems are mappings from a discrete input domain of binary input vectors, $\mathbb{B} \in \{0,1\}^n$, to outputs from the set of real numbers, $\mathbb{R}$. Binary values can be mapped to Boolean values such that $True \rightarrow 1$ represents the presence of a variable and $False \rightarrow 0$ represents the absence. This property is what gives it the name pseudo-Boolean. They form a representation of the absence or presence of multiple variables, where each variable's presence or absence influences the variables they interact with, leading to complex behaviors that vary by interaction.

An exhaustively search of all inputs in PBO problems is computationally expensive, as they suffer from the curse of dimensionality. A PBO problem composed of 25 variables that is limited to two variable interactions has a total of 326 learnable coefficients, one for each possible variable interaction [LVFM19]. Without limiting the number of variable interactions, the same problem would have coefficients $2^{25} = 33,554,432$. This makes learning the most influential variable interactions crucial for approximating such optimization problems when computation time is limited.

Current State-Of-The-Art (SOTA) models for approximating pseudo-Boolean functions for optimization are Gaussian process regression based on Kriging's approach, the radial basis function network, and the Bayesian approach [LVFM19]. All of these models use the surrogate modeling framework to learn the function behavior for approximation. They do so by sampling training data with known outputs and fitting the model to the data.

Recent research has emerged on the use of Walsh functions as a basis in surrogate models to decompose pseudo-Boolean functions with promising results [UML21]. The reason for the emergence of this research is that using Walsh functions to calculate factors, such as fitness values, the fitness distribution, and the best fitness, for a given Hamming distance, has a relatively fast computation time [VDL+18]. This makes it more efficient for optimization compared to other techniques. The Walsh decomposition can also be used to find crossover points and subproblems, both of which are important when approximating combinatorial domains seen in pseudo-Boolean functions. Additionally, the Walsh decomposition naturally forms an orthogonal basis that corresponds to each set of variables interaction, defined by a maximum order. Walsh coefficients are also sparse, meaning that there are only a few significant nonzero coefficients, which makes them well suited for modeling pseudo-Boolean functions, as they are also sparse.

Since the other SOTA models use the surrogate modeling framework, by integrating Walsh functions, they can be compared with standard SOTA models as a baseline. Since the current research on Walsh surrogate models has been limited, this report aims to expand on how the Walsh functions can be used for better approximations and which method works best in the context of pseudo-Boolean functions. This will be done by evaluating the accuracy of various regression surrogate models, described later, with a multilinear basis compared to a Walsh basis. The goal of this is to determine which regression surrogate works best with the Walsh basis and how the accuracy of the model is affected by the basis used. From there, the models will be optimized and their results compared.

1.1 Thesis overview

This report is a bachelor's thesis done under the supervision of Dr. Thomas Bäck and Dr. Furong Ye at the Leiden Institute for Advanced Computer Science. It investigates the performance of various regression models on both the Walsh basis and the multilinear basis for approximating and optimizing pseudo-Boolean functions. The code used in this report is linked in the Appendix.

Section 2 will go further into the formal definitions of pseudo-Boolean and Walsh functions in addition to other relevant terminology and equations. Section 3 will review the relevant work on current SOTA models for PBO. The surrogate models used will be explored and explained in Section 4. Section 5 will describe the optimizer that is used on all the models. Section 6 will explain the experimental setup and design for testing the models. Section 7 will review the results of the experiments. The final Section 8, will conclude the report with an overview of what was done, the limitations, and possible future work.

2 Definitions

2.1 Pseudo-Boolean Optimization

Pseudo-Boolean optimization is the mapping of a discrete search space to real numbers. As such, a pseudo-Boolean function is of the form:

$$f : \mathbb{B}^n \to \mathbb{R} \tag{1}$$

Where $\mathbb{B}$ is the set of binary values $\{0, 1\}$, $\mathbb{R}$ is the set of real numbers, and n is the dimensionality of problem [BH02]. The goal of pseudo-Boolean optimization is to find the binary input vector that maximizes or minimizes a given pseudo-Boolean function f.

Pseudo-Boolean functions can be represented as a polynomial function with a multilinear basis by the following equation:

$$f(x_1, ..., x_n) = \sum_{S \subseteq V} c_S \prod_{j \in S} x_j \tag{2}$$

Where $x = (x_1, ..., x_n) \in \mathbb{B}^n$ forms the binary vector, $V = \{1, 2, ..., n\}$ is the set of variable indices, S is a subset of indices of V, and c_S is the coefficient corresponding to the interaction of the variables indexed in S. For a function smaller than n, it is possible to evaluate every combination of binary values. However, as more variables are introduced, the scale of pseudo-Boolean functions grows exponentially, as the number of possible combinations is 2^n where n is the degree of the function. Each combination has a corresponding coefficient that needs to be approximated. This quickly becomes computationally expensive as more function evaluations are needed.

2.2 Surrogate Modeling

Due to the high complexity in the amount of variable interactions in pseudo-Boolean functions, it is computationally very expensive to optimize these problems using the function itself. Rather, a surrogate model can be used to evaluate potential solutions in a much more inexpensive manner

[Zae18]. They work by learning from sampled data points of the function to make accurate predictions with less computation time. In the case of PBO, the surrogate learns from binary input data and corresponding outputs to find the most significant variable interactions. The model keeps the variable interactions that are deemed significant, while the less significant interactions can be discarded [LVFM19]. Since the amount of variable interactions is limited when approximating pseudo-Boolean functions, the extraction of significant limited variable interactions to recreate the original expensive function proves accurate and requires fewer function evaluations.

Common models for learning these functions are regression models, radial basis function networks, support vector machines, and Kriging's Gaussian process [Zae18]. These models, specifically regression models, Gaussian process models, and the radial basis function network, are considered state of the art as surrogate models for pseudo-Boolean functions [BH02].

Once the surrogate model has been built and trained, it can be used for optimization. Since the surrogate model requires significantly fewer function evaluations, optimization occurs much more quickly and cheaper than the original evaluation function. For these reasons, surrogate models in pseudo-Boolean functions have great potential to improve the efficiency of optimization at a low cost.

2.3 Walsh Functions

Walsh functions were created by and named after Joseph Leonard Walsh [Wal23]. A Walsh function is an orthonormal basis of a continuous or discontinuous function that can be used to decompose functions containing real or complex numbers. A Walsh function φ_l for any integer represented in binary l, is defined for any binary string $x = (x_1, x_2, ..., x_n)$ with n variables by:

$$\varphi_l(x) = (-1)^{\sum_{i=1}^{n} l_i x_i} \tag{3}$$

Where l_i and x_i represent the i-th bit in the binary vectors l and x. In other words, a Walsh function with binary input x is a vector of values $\{-1, 1\}$ where each column corresponds to a subset of variable interactions l.

Each subset of variable interactions is retrieved from the full set of possible interactions. The set of interactions is limited by the maximum order variable k, where the full set of variable interactions has $k = n$. For each interaction, -1 is raised to the i-th bit-dot product of l_i and x_i. Each binary input x and interaction l produces one value, -1 or $+1$. To find the Walsh basis for a single input x, each interaction l must be evaluated with the input x as seen in equation 3. The resulting basis for input x is a vector composed of $\{-1, 1\}$ where each column corresponds to an interaction defined by l.

Consider, for example, a sampled input $x = [0, 1, 0]$ for $n = 3$. The input sampled x states that in this sample $x_1 = 0$, $x_2 = 1$ and $x_3 = 0$, where each x_i is a variable. To find the equivalent Walsh function when $l = [0, 0, 0]$:

$$\varphi_{[0,0,0]}([0, 1, 0]) = (-1)^{\sum_{i=1}^{n} l_i x_i} = (-1)^{(0\cdot 0)+(0\cdot 1)+(0\cdot 0)} = 1 \tag{4}$$

Thus, the Walsh function for these values is 1. In this case, l is representative of the empty interaction set for when $x = [0, 1, 0]$. The complete Walsh basis for this x can be found for each possible interaction l by repeating this process. The results for this are given in the table below:

l	[0,0,0]	[0,0,1]	[0,1,0]	[0,1,1]	[1,0,0]	[1,0,1]	[1,1,0]	[1,1,1]
x=[0,1,0]	1	1	-1	-1	1	1	-1	-1

This table is the Walsh basis vector for a singular feature x, where each column is a different set of variables that interact, also known as a subfunction [CWS14]. When creating a Walsh basis for pseudo-Boolean functions, the aim is to learn the corresponding Walsh coefficients for each interaction l that minimizes an error metric.

When considering multiple inputs beyond just x, a full matrix is produced in which each row corresponds to the Walsh basis of an input x and each column corresponds to a subset of variable interactions l. The full Walsh matrix with no limit on variable interactions would be 2^n rows by 2^n columns [Say18]. Although there are many ways to generate Walsh functions, this report uses the Hadamard matrix. Since the Walsh matrix is a variation of the Hadamard matrix, a Hadamard matrix of degree n can be reordered to produce the corresponding Walsh matrix of the same degree. This process is explained in more detail in Section 2.6.

2.4 Walsh Basis

Walsh functions are relevant in PBO as they can be used to preform the Walsh decomposition of the functions [Wal23]. The Walsh decomposition is used to approximate functions with reduced computation time. Pseudo-Boolean functions can be represented with a Walsh basis rather than a multilinear basis, defined by equation 2. A pseudo-Boolean function, f, with a Walsh basis, $\hat{f}$, is defined by [UML21]:

$$\hat{f}(x) = \sum_{l=1}^{2^n} \hat{\alpha}_l \cdot \varphi_l \tag{5}$$

Where l is a binary vector that represents a subset of variable interactions from the set of 2^n possible interactions and $\hat{\alpha}_l$ the corresponding learned coefficient for interaction l. Equation 5 generates the full Walsh basis for all variable interactions. This requires many function evaluations to approximate, making it necessary to limit the number of interactions. The equation can be rewritten to limit the number of variable interactions such that for every binary $x \in \{0,1\}^n$[LVFM19]:

$$\hat{f}_k(x) = \sum_{l \text{ where } o(\varphi_l) \leq k} \hat{\alpha}_l \cdot \varphi_l \tag{6}$$

Where o is the number of binary values equal to 1 in l, known as the order of l. The variable k is maximum number of variable interactions allowed. This means that the order of l, $o(l)$, must be less than or equal to k. Thus, the Walsh function with the max order $k = 2$ can be represented as:

$$\hat{f}_2(x) = \alpha_\emptyset + \sum_{i=1}^{n} \alpha_i(-1)^{x_i} + \sum_{i<j\in N} \alpha_{ij}(-1)^{x_i+x_j} \tag{7}$$

Each term α is a coefficient for a Walsh function. The term $\alpha_\emptyset$ is a constant where the variable interaction subset empty. The constant is equivalent to the average fitness function value of $\hat{f}$ [UML21].

The coefficient for the interaction subset l, $\hat{w}_l$, can be found using the discrete Walsh-Hadamard transformation. This will be explored in Section 2.6. Combining the resulting transform coefficients with the Walsh basis produces the pseudo-Boolean function generated by the equation 6. This process is known as the Walsh decomposition or Walsh transform of pseudo-Boolean functions.

2.5 Hadamard Matrix

As mentioned previously, the Walsh basis matrix is a type of discrete Hadamard matrix of the same degree that has been reordered to sequency ordering [Say18]. A Hadamard matrix is composed of values $\{-1, +1\}$ with orthogonal rows. A Hadamard matrix, when multiplied by its transpose, results in the identity matrix of the same degree. Thus, a Hadamard matrix can be used to find the corresponding Walsh matrix of the same degree. A Hadamard matrix of degree n can be generated through the recursive relationship [LP19]:

$$H_1 = \begin{bmatrix} 1 \end{bmatrix} \tag{8}$$

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{9}$$

$$H_{2^n} = \begin{bmatrix} H_{2^{n-1}} & H_{2^{n-1}} \\ H_{2^{n-1}} & -H_{2^{n-1}} \end{bmatrix} \tag{10}$$

The rows of the resulting Hadamard matrix, when arranged in sequency order, are the Walsh functions [Say18]. The reordering of the Hadamard matrix to sequency order is the main step in the discrete Walsh-Hadamard transform.

2.6 Discrete Walsh-Hadamard Transformation

The Discrete Walsh-Hadamard Transformation (DWHT) is the Walsh decomposition of pseudo-Boolean functions used to calculate the Walsh coefficients [UML21].

The transform requires the Hadamard matrix generated by the recursive relationship in equation 10. The rows are then arranged in sequency ordering. The sequency of a row is the number of sign changes in the row halved. Sequency ordering is obtained by first converting the row numbering to it's binary value, starting at row 0. Then convert the row's binary value into Gray code and reverse the result. In the final step, the values are converted back to decimal form [LP19]. In the case of H_4, where $n = 2$, the transformation occurs as follows:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{11}$$

Row	0	1	2	3
Binary	00	01	10	11
Gray Code	00	01	11	10
Reverse	00	10	11	01
Decimal	0	2	3	1

The end decimal value is the new order of the rows:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{array}{l} \text{row } 0 \\ \text{row } 2 \\ \text{row } 3 \\ \text{row } 1 \end{array} \tag{12}$$

Each row of the reordered matrix, H_4, is a Walsh function of degree 2. However, the transformation is not necessarily complete. In addition to arranging the Hadamard matrix in sequency order, it needs to be normalized to find the Walsh coefficients. Normalization is often done by multiplying the sequency-ordered Hadamard matrix by $\frac{1}{\sqrt{2^n}}$ [UML21]:

$$W_{2^n} = \frac{1}{2^n}[H_{2^n}] \tag{13}$$

Where H_{2^n} is in sequency order already. The final matrix W_{2^n} is known as the DWHT transform matrix. Continuing with the example above with $n = 2$, the DWHT transform matrix of H_4 is:

$$W_4 = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{14}$$

To find all the coefficients using the DWHT transform matrix, all outputs of the function must be known and stored in a vector F [LVFM19]:

$$F(x_1, x_2, ..., x_n) = \begin{bmatrix} f(0,0,...,0,0) \\ f(0,0,...,0,1) \\ f(0,0,...,1,0) \\ f(0,0,...,1,1) \\ ... \\ f(1,1,...,1,1) \end{bmatrix} \tag{15}$$

The complete equation to get the set of Walsh coefficients, α, is defined as follows [UML21]:

$$\hat{\alpha} = \begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{1\}} \\ \alpha_{\{2\}} \\ \alpha_{\{1,2\}} \\ ... \\ \alpha_{\{1,...,n\}} \end{bmatrix} = \frac{1}{2^n} H_{2^n} \cdot \begin{bmatrix} f(0,0,...,0,0) \\ f(0,0,...,0,1) \\ f(0,0,...,1,0) \\ f(0,0,...,1,1) \\ ... \\ f(1,1,...,1,1) \end{bmatrix} \tag{16}$$

Equation 16 requires all outputs to be known to find the set of all interaction coefficients α. This is not useful for approximation; however, the interaction coefficients can be learned through training a surrogate model on a subset of interactions. These approximate coefficients, $\hat{\alpha}$, are used with the Walsh basis to generate predictions [LVFM19].

The Walsh coefficients, in addition to being used to represent pseudo-Boolean functions on a Walsh basis, also demonstrate the strength of the interactions. A larger coefficient means that the corresponding interaction has a stronger effect than an interaction with a smaller coefficient

[LVFM19]. The coefficients are therefore representative of the contribution a subset of variable interactions has on the fitness output. This is very useful for approximation as it is known what interactions are the most important to the function and which can be disregarded.

3 Related Work

In recent years, Walsh functions have emerged in pseudo-Boolean optimization models. Specifically, in surrogate models that approximate the pseudo-Boolean functions for optimization. This section will go over these recent models and how they performed.

Florian Leprêtre and coauthors investigated the use of Walsh functions in a surrogate-assisted optimization framework [LVFM19]. They termed their model the Walsh Surrogate-assisted Optimization algorithm (WSaO). Leprêtre and co-authors compared their Walsh surrogate-assisted optimization algorithm (WSaO) with other SOTA techniques, including Bayesian optimization of combinatorial structures (BOCS) and Gaussian process regression (GPR). They found that WSaO worked best in the highest dimension tested, n=100.

Additional work by Sébastien Verel and collegues found that Walsh decomposition as a surrogate model had a similar accuracy to the Kriging approach when variable interactions were k=0 [VDL+18]. However, at k=1 and k=2, the Walsh surrogate was able to reach zero mean absolute error in fewer function evaluations. While they found the kriging approach had better accuracy on linear models (k=0) and required fewer evaluations, the Walsh surrogate converged faster.

Kevin Swingler also introduced a model for pseudo-Boolean optimization using Walsh functions to decompose pseudo-Boolean functions [Swi20]. He introduces mixed-order hyper networks (MOHNS) that utilize Walsh decompositions alongside a hypergraph representation and linear parameter estimation to estimate the fitness. MOHNS use the Walsh decomposition to find relationships between the variables and uses that information to sample fitness evaluations. Swingler concludes that MOHNs require significantly fewer fitness evaluations compared to the possible evaluations, thus building an accurate model with less sampling.

As mentioned in Leprêtre's paper, another SOTA technique for approximating these functions is BOCS. A pseudo-Boolean function has a combinatorial domain, as each evaluation is a combination of binary values assigned to the function variables. Previous Bayesian models were poorly suited for pseudo-Boolean optimization because they suffered greatly from the curse of dimensionality [GDMB+24]. Ricardo Baptista and Matthias Poloczek introduce a BOCS algorithm that is able to deal with the high dimensionality of pseudo-Boolean functions [BP18]. Their BOCS algorithm utilizes a statistical model with sparse Bayesian linear regression and a version of Thompson sampling as an acquisition function. They test both semidefinite program (SDP) and simulated annealing (SA) as optimizers for the acquisition function. They concluded that their BOCS model outperforms other SOTA techniques of machine learning and combinatorial optimization.

Other common SOTA techniques include Kriging's approach, also known as Gaussian process regression (GPR), and radial basis function network (RBFN). Martin Zaefferer and co-authors tested GPR and RBFN as surrogate models with efficient global optimization (EGO) [ZSF+14]. They found that the EGO to be the best optimization method and Hamming distance to be the most suited distance measure. The EGO with GPR outperformed the EGO with RBFN, however, both EGO algorithms outperformed the other models tested.

4 Regression Models

Regression models are a type of supervised learning that aims to predict outputs from input data composed of one or more independent variables [Syk93]. When there is only one linear interactions, meaning each varible does not interact with other variables, a simple regression model is used for learning. This is equivalent to having $k = 1$ variable interactions in the case of pseudo-Boolean functions.

The regressions models this report tests are least angle regression, Lasso regression and Lasso-LARS regression. Each of these models will be tested on different sizes of training data and as a baseline, each Walsh basis regression model will be compared to it's multilinear counterpart. The aim of this is to see which regression works best for the Walsh basis and whether it outperforms the standard implementation.

4.1 Simple Linear Regression

A simple regression is equivalent to linear regression where the model attempts to approximate a linear function:

$$\hat{y} = \beta_0 + \beta_1 x + \epsilon \tag{17}$$

Where $\hat{y}$ is the predicted output at an input x. The input x is the independent variable, β_1 is the coefficient of input x and β_0 is the intercept. The ϵ is known as the noise term or the estimated error of each prediction. This can also be rewritten with a mean and variance function [Wei05]:

$$m(x) = \beta_0 + \beta_1 x \tag{18}$$

$$var(x) = \sigma^2 \tag{19}$$

The variance represents the amount of error between the expected output of the mean function $m(x_i)$ and the true output y_i, so while the variance $\sigma^2 > 0$, for each x_i there is a corresponding error term ϵ_i such that:

$$y_i = m(x_i) + \epsilon_i \tag{20}$$

In order to learn the best parameters for prediction, linear regression uses ordinary least squares [PVG+11]. Ordinary least squares works by trying to find estimators β_0 and β_1 that minimizes the loss function [Wei05]:

$$RSS(\beta_0, \beta_1) = \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i]^2 \tag{21}$$

Where $RSS(\beta_0, \beta_1)$ is the residual sum of squares (RSS) of (β_0, β_1). The residuals reflect the vertical error between the predicted $\hat{y_i}$ and true y_i. These estimators can be found by:

$$\hat{\beta_1} = \frac{SXY}{SXX} = r_{xy}\frac{SD_y}{SD_x} = r_{xy}\left(\frac{SYY}{SXX}\right)^{1/2} \tag{22}$$

$$\hat{\beta_0} = \overline{y} - \hat{\beta_1}\overline{x} \tag{23}$$

Where $\overline{x}$ is the sample average of x and $\overline{y}$ is the sample average of y. Additionally, SXX is the sum of squares of xs, SYY is the sum of squares of ys and SXY is the sum of cross products:

$$SXX = \sum(x_l - \overline{x})^2 = \sum(x_l - \overline{x})x_l \tag{24}$$

$$SYY = \sum(y_l - \overline{y})^2 = \sum(y_l - \overline{y})y_l \tag{25}$$

$$SXY = \sum(x_l - \overline{x})(y_l - \overline{y}) = \sum(x_l - \overline{x})y_l \tag{26}$$

Finally, SD_y and SD_x are the sample standard deviation of the ys and xs respectively. The r_{xy} is the sample correlation with s_{xy} as the sample covariance:

$$s_{xy} = \frac{SXY}{n-1} \tag{27}$$

$$r_{xy} = \frac{s_{xy}}{SD_x SD_y} \tag{28}$$

4.2 Polynomial Features

When there is more than one independent variable, multiple regression is used [Syk93]. A multiple regression model with k independent variables and n observations can be written [Ost12]:

$$\begin{cases} y_1 = \beta_0 + \beta_1 x_{11} + ... + \beta_k x_{1k} + \epsilon_1 \\ y_2 = \beta_0 + \beta_1 x_{21} + ... + \beta_k x_{2k} + \epsilon_2 \\ ... \\ y_n = \beta_0 + \beta_1 x_{n1} + ... + \beta_k x_{nk} + \epsilon_n \end{cases} \tag{29}$$

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + \epsilon_i, \text{ for i=1,2,...,n} \tag{30}$$

This equation can be simplified by rewriting the terms as vectors:

$$Y = X\beta + \epsilon \tag{31}$$

Where Y is the output vector, X is the inputs vector, β is the vector of learnable parameters and ϵ the vector of error terms. As opposed to simple linear regression, where only the intercept and slope, $\beta = \{\beta_0, \beta_1\}$, are learned, multiple regression learns multiple coefficients $\beta = \{\beta_0, \beta_1, ..., \beta_k\}$ corresponding to different k variable interactions. However, both simple and multiple regression have error terms ϵ corresponding to an input and it's expected output.

The multiple regression model works by transforming the input data into polynomial terms such that linear regression can be performed [PVG+11]. For example, given an input vector X_{input} with fourth features $[x_1, x_2, x_3, x_4]$ containing 8 observations, when transformed into polynomial features of degree 2, meaning only two variable interactions at a time, X_{input} is transformed into X_{poly}:

$$X_{input} = \begin{bmatrix} 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0101 \\ 1010 \\ 0110 \\ 0111 \end{bmatrix} = \begin{bmatrix} \emptyset \\ x_4 \\ x_3 \\ x_3 \cdot x_4 \\ x_2 \cdot x_4 \\ x_1 \cdot x_3 \\ x_2 \cdot x_3 \\ x_2 \cdot x_3 \cdot x_4 \end{bmatrix} \tag{32}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_1^2 & x_2^2 & x_3^2 \\ x_4^2 & x_1 \cdot x_2 & x_1 \cdot x_3 & x_1 \cdot x_4 & x_2 \cdot x_3 & x_2 \cdot x_4 & x_3 \cdot x_4 \end{bmatrix} \tag{33}$$

The original 4 features were transformed into 14 features which is composed of the original 4 features and 10 new interaction features. Since $x_i^2 = x_i$ when working with binary variables, the self-interactions can be discarded, making the final polynomial features set:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_1 \cdot x_2 & x_1 \cdot x_3 & x_1 \cdot x_4 & x_2 \cdot x_3 & x_2 \cdot x_4 & x_3 \cdot x_4 \end{bmatrix} \tag{34}$$

Thus for each of the 8 observations in X_{input}, the polynomial feature set is applied to these observations resulting in the following matrix:

$$X_{input} \rightarrow X_{poly} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{35}$$

The final X_{poly} is transformed into one row of features where linear regression models can be used. This means multiple regression minimizes a loss function, sometimes the same as in equation 21 used in linear regression, but a parameter β for each feature in the vector 34.

The use of scikit-learns polynomial features is in all of the base regression models. However it is not needed for the Walsh basis regression models. Rather than use the transformed raw input data as the features which results in a multilinear basis, the Walsh models use the Walsh basis of the input data as features. This means that the multilinear basis models pass 35 as the x train and the Walsh basis models pass 14 as the x train.

4.3 Least Angle Regression

One of the regression model this report tests is Least-Angle Regression (Lars), also called forward stepwise regression. Lars is able to find the set of solutions in m steps, where m is the number of covariates.

The Lars model is composed of the active set, $\mathcal{A}$, a vector of signs, $s_{\mathcal{A}}$, and a vector of predictions, $\hat{\mu}$, a correlation vector for each prediction, $c(\hat{\mu})$, the predictors vector X and the response vector y.

At initialization, the Lars model sets its components as follows [EHJT04]:

$$\begin{aligned}&\mathcal{A} = \emptyset \\ &s_j = sign(c_j)\ for\ j \in \mathcal{A} \\ &\hat{\mu} = 0\end{aligned} \tag{36}$$

The vector of predictions $\hat{\mu}$ contains the predicted coefficients starting at zero and is used to calculate the vector of correlations $c(\hat{\mu})$ where [ZZ14]:

$$c(\hat{\mu}) = X^T(y - \hat{\mu}) \tag{37}$$

Such that the correlation c_j is proportional to x_j and the current residual vector.

After initialization, the first step is to select the predictor in X that has the absolute highest correlation with the response y. Suppose that the predictor with the highest correlation is x_{j_1}. Lars then updates the active set with x_{j_1} with $\mathcal{A} = \mathcal{A} \cup \{j_1\}$ [ZZ14]. Furthermore, the prediction vector $\hat{\mu}$ is updated in the direction of the selected predictor x_{j_1} with the following equation [EHJT04]:

$$\hat{\mu}_1 = \hat{\mu}_0 + \gamma s_{j_1} x_{j_1} \tag{38}$$

Where γ is the step length such that $\gamma \geq 0$. The step length will be the largest possible in the direction of the predictor that can be taken until there is another predictor, say x_{j_2}, with an equal correlation with the residual. This variable is added to the active set.

Now that the active contains more than one predictor, rather than updating the predictions in the direction of the new predictor x_{j_2}, Lars takes into account all variables in the active set. It does so by updating the direction using the equiangular direction vector $u_{\mathcal{A}}$ of the active variables. The equiangular direction refers to the direction in which all variables in the active set are equally correlated [ZZ14]. In other words, each predictor in the active set now contributes equally to the direction change.

Let the predictors in the active set be written as [EHJT04]:

$$X_{\mathcal{A}} = (...s_j x_j...)_{j \in \mathcal{A}} \tag{39}$$

And let:

$$\begin{aligned}\mathcal{G}_{\mathcal{A}} &= X'_{\mathcal{A}} X_{\mathcal{A}} \\ A_{\mathcal{A}} &= (1'_{\mathcal{A}} \mathcal{G}_{\mathcal{A}}^{-1} 1_{\mathcal{A}})^{-1/2}\end{aligned} \tag{40}$$

With $1_{\mathcal{A}}$ as vectors of 1's of equal length of the active set. Then the equiangular vector $u_{\mathcal{A}}$ is:

$$\begin{aligned}u_{\mathcal{A}} &= X_{\mathcal{A}} w_{\mathcal{A}} \\ w_{\mathcal{A}} &= A_{\mathcal{A}} \mathcal{G}_{\mathcal{A}}^{-1} 1_{\mathcal{A}}\end{aligned} \tag{41}$$

The equiangular vector $u_{\mathcal{A}}$ is a unit vector that creates equal angles less than 90°with the columns of $X_{\mathcal{A}}$ and $w_{\mathcal{A}}$ is a weight vector.

$$\begin{aligned}&X'_{\mathcal{A}} u_{\mathcal{A}} = A_{\mathcal{A}} 1_{\mathcal{A}} \\ &||u_{\mathcal{A}}||^2 = 1\end{aligned} \tag{42}$$

Assume that the Lars model has the estimate $\hat{\mu}_{\mathcal{A}}$ and the correlations:

$$\hat{c} = X'(y - \hat{\mu}_{\mathcal{A}}) \tag{43}$$

Then define the largest absolute correlation as:

$$\hat{C} = max_j(|\hat{c}_j|) \tag{44}$$

Now find a in order to update $\hat{\mu}_{\mathcal{A}}$:

$$a \equiv X'u_{\mathcal{A}} \tag{45}$$

Finally, the Lars algorithm updates the predictions:

$$\hat{\mu}_{\mathcal{A}^+} = \hat{\mu}_{\mathcal{A}} + \hat{\gamma}u_{\mathcal{A}} \tag{46}$$

Where the step length γ can be computed as follows:

$$\hat{\gamma} = min^+_{j \notin \mathcal{A}} \left(\frac{\hat{C} - \hat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\hat{C} + \hat{c}_j}{A_{\mathcal{A}} + a_j} \right) \tag{47}$$

These steps repeat until all predictors are added to the active set. In summary, Lars uses stagewise regression to make ϵ small steps in the coefficients in the direction of the most correlated variable to get more accurate predictions [ZZ14].

4.4 Lasso Regression

Lasso regression is a type of ordinary least-squares regression that has been constrained [UGH09]. The way Lasso is constrained is while it minimizes the residual sum of squares it limits the sum of the absolute values of it's regression coefficients. This makes the Lasso model a type of linear model that uses L1 prior to regularization [PVG+11]. As a result, it is highly sparse, which works well with pseudo-Boolean functions, as some interactions have little to no influence on the response and Lasso models allow the regression coefficients to be equal to zero [Han09].

Lasso controls the L_1-norm of its coefficient vector β using a penalized least squares [Han09]:

$$\hat{\beta_L} = argmin_\beta (y - X\beta)^T (y - x\beta) + \alpha ||\beta||_1 \tag{48}$$

With $||\beta||_1$ [ZZ14]:

$$||\beta||_1 = \sum_{i=1}^{p} |\beta_i| \tag{49}$$

Where p is the number of predictors or features and α determines the amount of shrinkage in the coefficient vector. When $\alpha = 0$, the model is equivalent to a regular ordinary least-squares regression since no shrinkage is applied [PVG+11]. When α is large enough, $\hat{\beta}_L$ can shrink to zero [Han09].

4.5 Lasso-LARS Regression

The last type of regression that this report evaluates is Lasso-Lars regression. Since LARS is closely related to Lasso, it has been shown that LARS can make a fast implementation of Lasso [EHJT04]. If $\hat{\beta}$ with $\hat{\mu} = X\hat{\beta}$ is a Lasso solution, then the sign of a nonzero $\hat{\beta}_j$. The sign s_j of $\hat{\beta}_j$ must agree with the sign of the current correlation $\hat{c}_j = x'_j(y - \hat{\mu})$ such that:

$$sign(\hat{\beta}_j) = sign(\hat{c}_j) = s_j \tag{50}$$

This restriction is not included in the standard LARS model.

If a LARS step has been completed, resulting in a new active set $\mathcal{A}$ then its estimate $\hat{\mu}_{\mathcal{A}}$ corresponds to a Lasso solution $\hat{\mu} = X\hat{\beta}$. Then, just as in LARS, the vector of weights is defined:

$$w_{\mathcal{A}} = A_{\mathcal{A}}\mathcal{G}_{\mathcal{A}}^{-1}1_{\mathcal{A}} \tag{51}$$

Now, define a vector $\hat{d}$ with a length equal to the number of predictors. This vector is equal to $s_j w_{\mathcal{A}_j}$ for $j \in \mathcal{A}$ and zero for $j \notin \mathcal{A}$. When the LARS direction is positive, it can be seen that:

$$\begin{aligned} \mu(\gamma) &= X\beta(\gamma) \\ \beta_j(\gamma) &= \hat{\beta}_j + \gamma\hat{d}_j \end{aligned} \tag{52}$$

Again for $j \in \mathcal{A}$. Thus, $\beta_j(\gamma)$ will have a sign change at:

$$\gamma_j = -\hat{\beta}_j/\hat{d}_j \tag{53}$$

With the first sign change happening at:

$$\widetilde{\gamma} = min_{\gamma_j > 0}(\gamma_j) \tag{54}$$

For covarite $x_{\widetilde{j}}$.

It can be said that if $\widetilde{\gamma}$ is less than the estimated $\hat{\gamma}$, then $\beta_J(\gamma)$ can't be a Lasso solution for $\gamma > \widetilde{\gamma}$ as the sign restriction in equation 50 will have been violated. This is because $\beta_j(\gamma)$ has changed sign while $c_j(\gamma)$ hasn't, hence the violation.

To summarize, the integration of Lasso into the LARS model changes the update rule seen in equation 46. That is, if $\widetilde{\gamma} < \hat{\gamma}$, stop the current LARS step at $\gamma = \widetilde{\gamma}$ and remove the index $\widetilde{j}$ of covarite $x_{\widetilde{j}}$ from the next calculation in the equiangular direction. The new update rule for the Lasso-LARS regression is as follows:

$$\begin{aligned} \hat{\mu}_{\mathcal{A}_+} &= \hat{\mu}_{\mathcal{A}} + \widetilde{\gamma}\hat{\mu}_{\mathcal{A}} \\ \mathcal{A}_+ &= \mathcal{A} - \widetilde{j} \end{aligned} \tag{55}$$

This rule replaces the LARS update in equation 46.

5 Hamming-Ball Efficient Hill-Climber Optimization

To optimize the approximated pseudo-Boolean functions, this report uses the same efficient hill-climber (EH) implementation used by Florian Leprêtre and coauthors as it is well suited for binary optimization specifically in the case of Walsh functions [LVFM19]. The EH algorithm used was introduced by Francisco Chicano and coauthors for the purpose of PBO [CWS14].

Chicano and coauthors define the r-ball neighborhood as a set of solutions for binary strings of size n with a Hamming distance of r or less from the current solution. Hamming distance is the difference between two binary strings where for each bit they have in common, one is added to the distance. As such, the Hamming Ball score for an input x for a pseudo-Boolean function $f : \mathbb{B}^n \to \mathbb{R}$ is [CWS14]:

$$S_v(x) = f(x \oplus v) - f(x) \tag{56}$$

For $v, x \in \mathbb{B}^n$ where v is the move or bitflip, and $S_v(x)$ is the respective score. The $\oplus$ is the exclusive OR bitwise operation. All possible scores for altered strings $f(x \oplus v)$ where $|v| \leq r$ are stored in a vector that is updated as the new moves are tested. By storing the scores in this vector, it makes it possible to find improving moves that are in a ball, or reside, in the radius r from the current solution. This continues until either no improvements can be made or the preset budget has run out.

Chicano and coauthors give the following pseudo-code for their Hamming-Ball EH [CWS14]:

Algorithm 1 Hamming-Ball Efficient Hill-Climber

```
$best \leftarrow None$
while budget not spent do
    $x \leftarrow$ Choose random solution
    $S \leftarrow$ Compute the scores of x with regards to v with equation 56
    while $S_v > 0$ for some $v \in M^r$ do
        $t \leftarrow$ Choose an improving move from vector S
        Update the score vector S with x and t
        $x \leftarrow x \oplus t$
    end while
    if $best = None$ or $f(x) > f(best)$ then
        $best \leftarrow x$
    end if
end while
```

The first while loop in line 2 ensures that the algorithm does not exceed the predefined budget. The while loop in line 5 ensures that there are still improving scores.

While this algorithm was shown to perform well on Walsh surrogates when tested by Florian Leprêtre and coauthors, they used both EH and Bayesian optimization for their sparse linear regression surrogate using a mutlilinear basis with varying results of which performed better depending on the problem [LVFM19]. This report will use the Hamming-ball EH for all regression models. While the optimizer should perform well for the Walsh surrogates, it is worth noting that it may be suboptimal for the multilinear surrogates, especially considering the Bayesian optimization is a SOTA optimizer for pseudo-Boolean functions. This is addressed further in the section 8.

6 Experiments

6.1 Benchmark

The models' benchmarks are problem sets provided by IOHprofiler's experimentation problems for PBO [DYH+20]. This report will test both the NK-landscapes problem and the OneMax with w-model epitasis problem, generated by IOHprofiler.

6.1.1 NK-Landscapes Problem

An NK-landscape (NKL) function is a mapping of function states to a fitness measure [Csa18]. In other words, it evaluates the fitness of a set of inputs that form the representation of the state of the function. The function has n components g where n is the amount of variables and g is the set of components such that $g = (g_1, g_2, ..., g_n)$. Each component g_i can be in a variety of states that depend on the fitness of the other components with which it interacts. Each component has k components it interacts with; this is known as the max order of the function. When $k = 0$, each component is considered independently without interactions, at $k = 1$, each component interacts with one other component, making the function linear, and at $k = 2$, the function is quadratic, a polynomial of degree 2. The degree of the polynomial representation of the function increases with k and the higher k means more coefficients to estimate. As a result, fitness is found by adding the fitness contribution of each component and getting the average [DYH+20]. The equation for fitness can be seen below [LVFM19]:

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} g_i(x_i, x_{i_1}, x_{i_2}, ..., x_{i_n}) \tag{57}$$

The NKL functions make for a very fitting set of problems to evaluate the models, as they very clearly demonstrate how variable interactions influence the landscape of the fitness function [Csa18]. This makes them more closely aligned to some real world problems relative to other problem types [LVFM19].

6.1.2 OneMax with W-Model Epitasis

The OneMax with w-model epitasis expands on the baseline OneMax model given by IOHprofiler's PBO set [DYH+20]. The standard OneMax problem has a smooth landscape making it very easy to approximate. It can be represented by the equation [DYH+20]:

$$\{0, 1\} \to [0...n], x \mapsto \sum_{i=1}^{n} x_i \tag{58}$$

The w-model introduces new alterations to the existing OneMax baseline, creating new functions and features. The IOH w-model has 4 different transformations, but this report will focus on the epistasis transformation. The epistasis transformation works by dividing an input x into blocks of size v. Then a permutation e_v is applied to each of these blocks, with e_v defined as such:

$$e_v : \{0, 1\}^v \to \{0, 1\}^v \tag{59}$$

The permutation e_v is chosen so that two neighboring blocks with a Hamming distance of 1 are altered so that they have a Hamming distance of $v - 1$. Each of the new substring blocks is mapped to another string such that:

$$(y_{(i-1)v+1}, ..., y_{iv}) = e_v(x_{(i-1)v+1}, ..., x_{iv}) \tag{60}$$

The resulting landscape is much more rugged than the baseline OneMax problem. This problem is well suited for testing the surrogate models as it's ruggedness makes optimization more difficult, as with each bitflip, there is a large change.

6.2 Models

All of the implemented models will use the Scikit-Learn framework [PVG+11]. The models that will be tested are linear regression with polynomial features, Lasso regression, least angle regression (Lars) and Lasso-Lars regression. The implementation of these models aims to expand on current research on Walsh-based surrogate models for approximating pseudo-Boolean functions. Since much of the current research focuses on one Walsh-based model compared with other SOTA models, this paper expands this by testing multiple regression types to see which performs best in terms of mean absolute error when using a Walsh basis.

The first evaluation of regression surrogate models will compare the use of a Walsh basis instead of the standard basis. The equation for the Walsh basis of all interactions is defined by equation 5 and the equation for the Walsh basis with a limit on variable interactions is given by equation 6. This report uses the second equation 6 as the goal of surrogate modeling is to approximate the model, not to learn all the coefficients. The multilinear basis is typical in surrogate modeling techniques and is given by equation 2.

6.3 Experimental Setup

Each model will be tested on multiple instances of NKL problems, with each problem tested 5 times and averaged over those repetitions. The models will be tested on various dimensions, $n = \{10, 25, 50, 100\}$, and various interactions, $k = \{1, 2, 3\}$. The models performance will be measured by their mean absolute error (MAE) and averaged for each training size interval. The goal of this is to demonstrate how much training data is required for the models to reach zero or small enough MAE. Models that require less training data are considered more efficient than those that require more, assuming they reach the same MAE score.

The Lasso regression model will use an alpha of 1×10^{-5}, the same used in Leprêtre and coauthors Walsh surrogate-assisted optimization [LVFM19]. Lasso-Lars will use the same alpha value. All other models will use the default arguments given by Scikit-Learn [PVG+11].

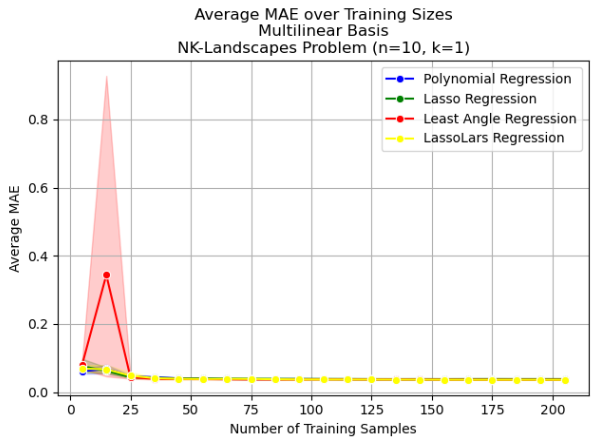
Once the approximation has concluded, each model will be optimized with the efficient hill-climber, introduced in the section 5. The models will attempt to maximize multiple instances of NKL and OneMax problems using the dimensions and interactions set above. Each problem instance at a given n and k will be tested five times in order to obtain the confidence interval, set to 95%.

7 Results

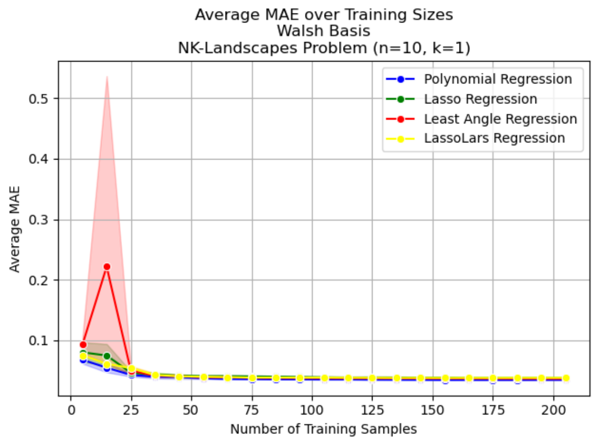
The results have been divided into two main sections, one for approximation and one for optimization. In the approximation section, each graph includes the average mean absolute error (MAE) with a confidence interval of 95% on the y-axis and the size of the training set on the x-axis. The results have also been divided into sections based on the value of n. Some graphs have a secondary version if the original graph is too difficult to visualize. These sections will cover both benchmarks and all regression models.

7.1 Approximation Results

7.1.1 Results for n=10

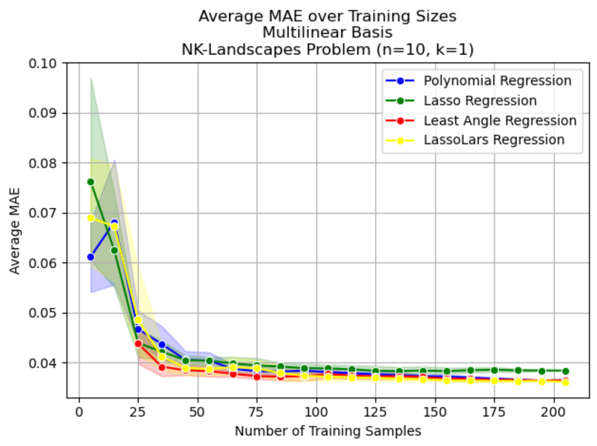


(a) Multilinear Basis Models

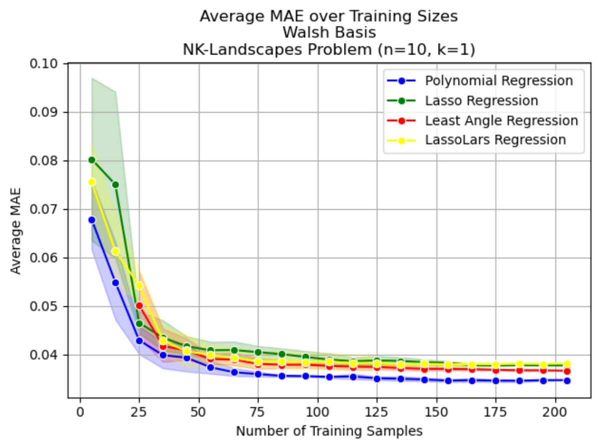


(b) Walsh Basis Models

Figure 1: Mean absolute error over training sizes on the NK-landscapes problem, for n=10 and k=1 with confidence interval (95%).

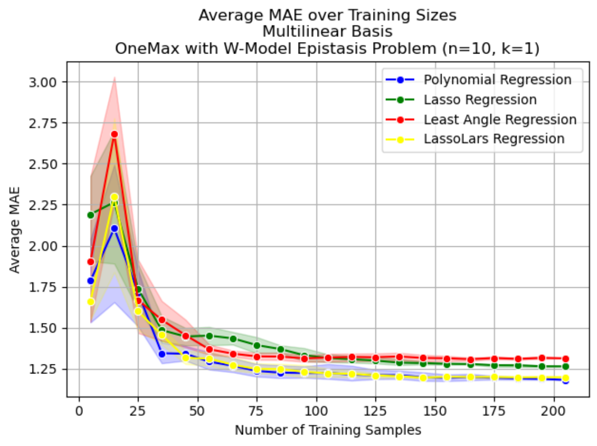


(a) Multilinear Basis Models

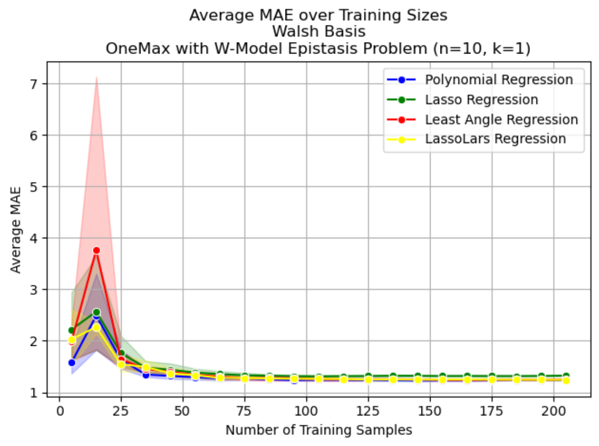


(b) Walsh Basis Models

Figure 2: Mean absolute error over training sizes on the NK-landscapes problem, for n=10 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 1 except both basis start the Lars model at the second sample interval.

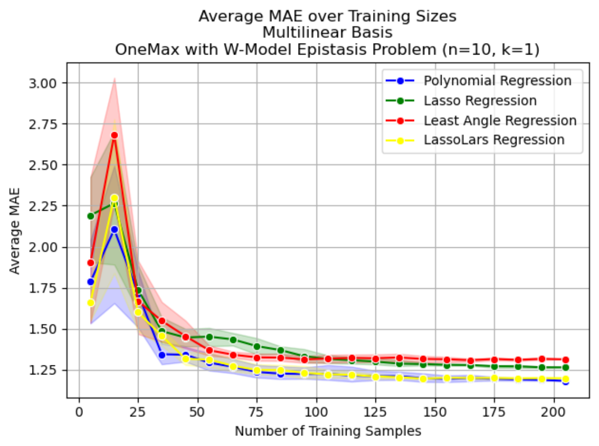


(a) Multilinear Basis Models

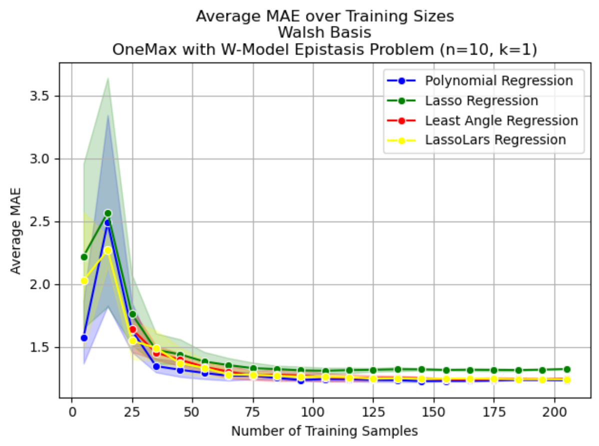


(b) Walsh Basis Models

Figure 3: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=10 and k=1 with confidence interval (95%).

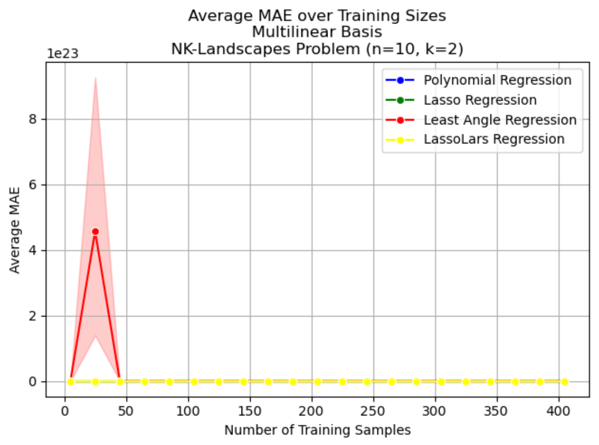


(a) Multilinear Basis Models

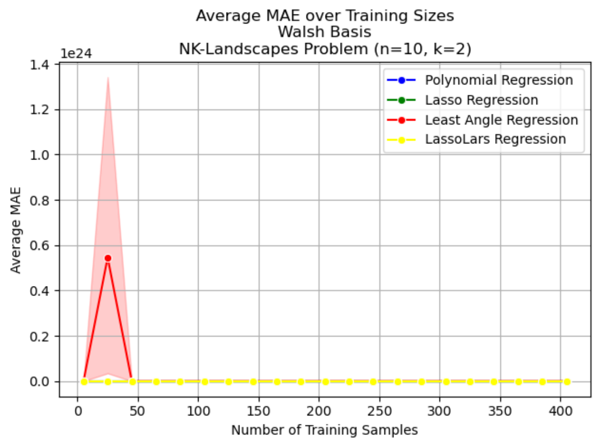


(b) Walsh Basis Models

Figure 4: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=10 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 3 except the Walsh basis starts the Lars model at the second sample interval.

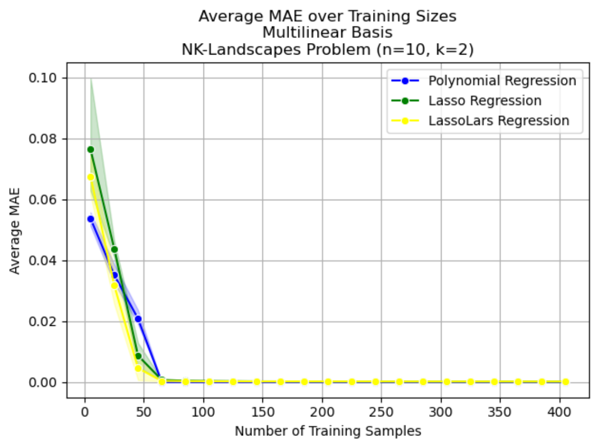


(a) Multilinear Basis Models

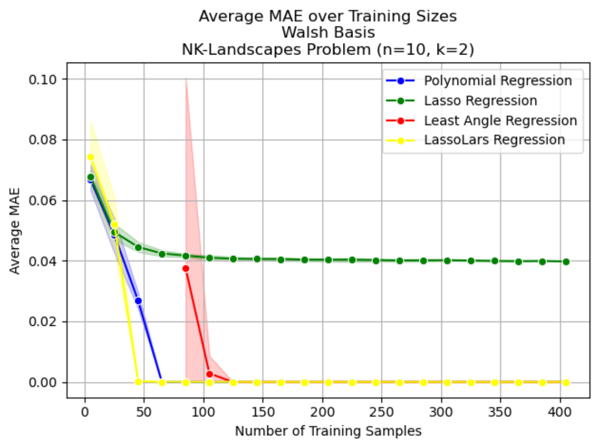


(b) Walsh Basis Models

Figure 5: Mean absolute error over training sizes on the NK-landscapes problem, for n=10 and k=2 with confidence interval (95%).

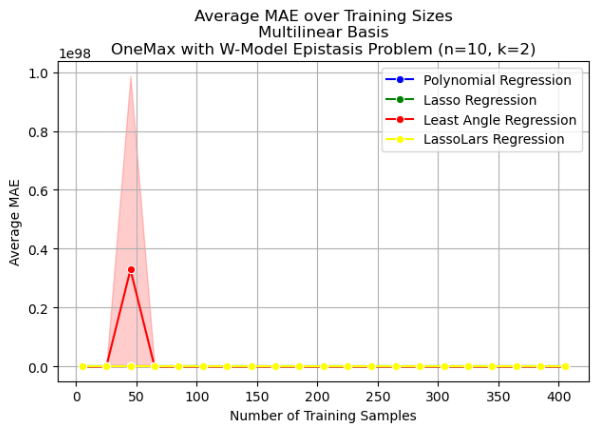


(a) Multilinear Basis Models

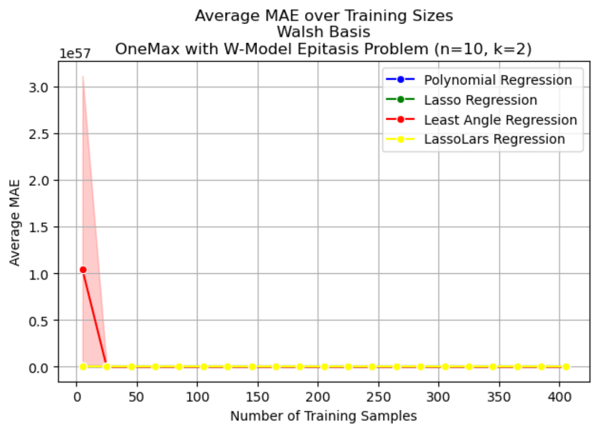


(b) Walsh Basis Models

Figure 6: Mean absolute error over training sizes on the NK-landscapes problem, for n=10 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 5 with the Lars model excluded in the multilinear basis graph and the Lars model starting from the fourth sample interval in the Walsh basis.

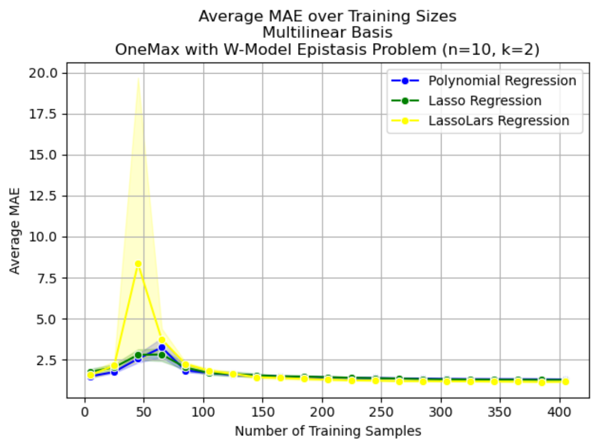


(a) Multilinear Basis Models

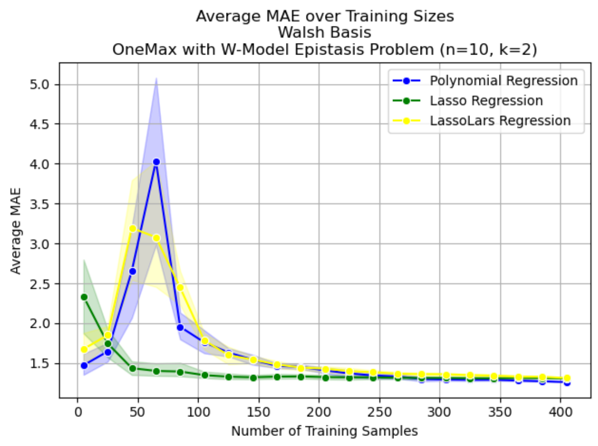


(b) Walsh Basis Models

Figure 7: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=10 and k=2 with confidence interval (95%).



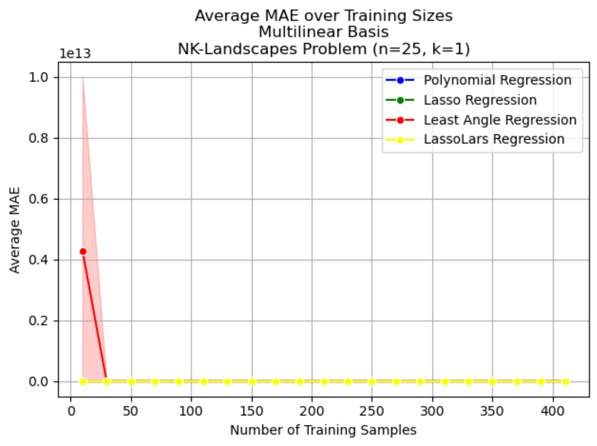
(a) Multilinear Basis Models



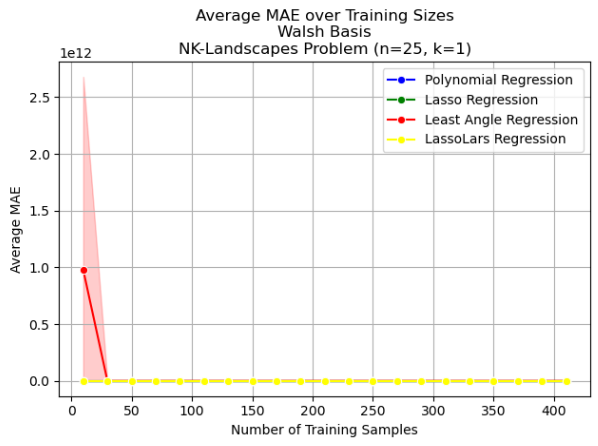
(b) Walsh Basis Models

Figure 8: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=10 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 7 with the Lars model excluded.

7.1.2 Results for n=25

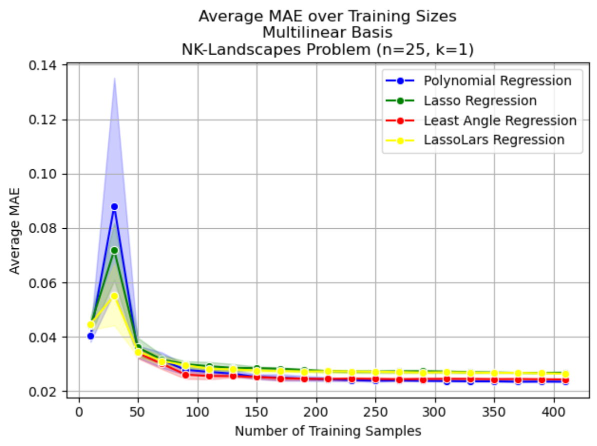


(a) Multilinear Basis Models

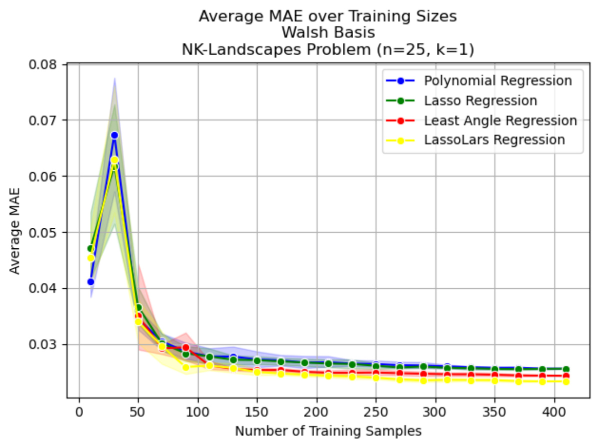


(b) Walsh Basis Models

Figure 9: Mean absolute error over training sizes on the NK-landscapes problem, for n=25 and k=1 with confidence interval (95%).

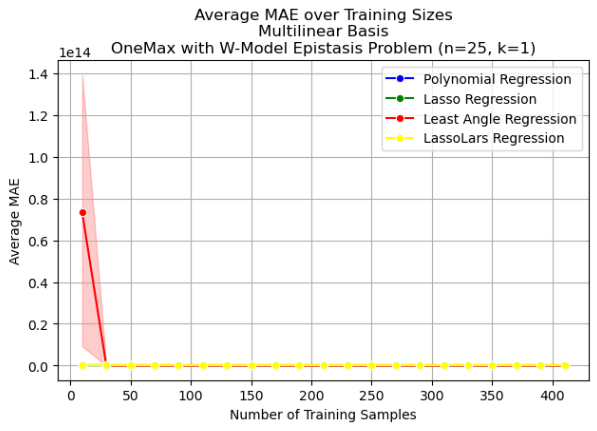


(a) Multilinear Basis Models

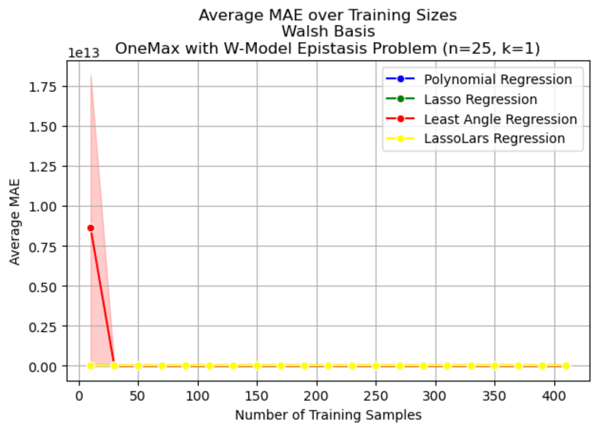


(b) Walsh Basis Models

Figure 10: Mean absolute error over training sizes on the NK-landscapes problem, for n=25 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 9 except the Lars model starts from the second training interval.

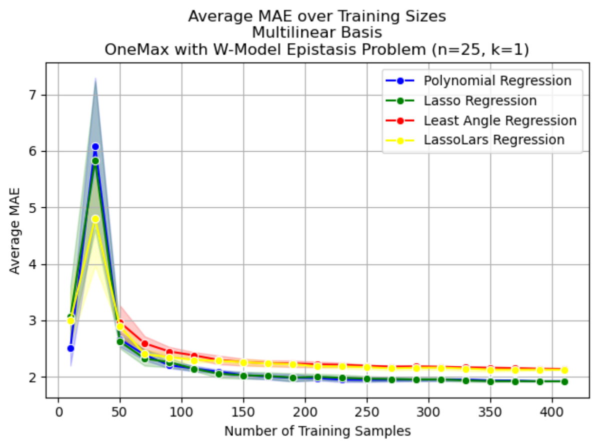


(a) Multilinear Basis Models

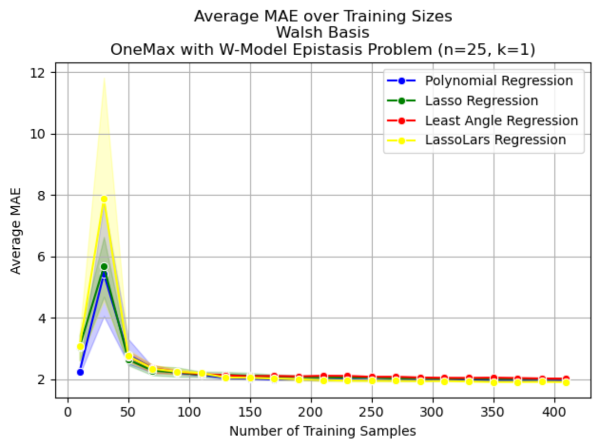


(b) Walsh Basis Models

Figure 11: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=25 and k=1 with confidence interval (95%).

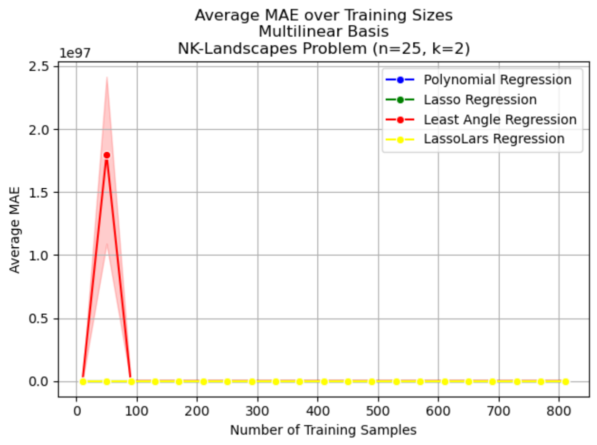


(a) Multilinear Basis Models

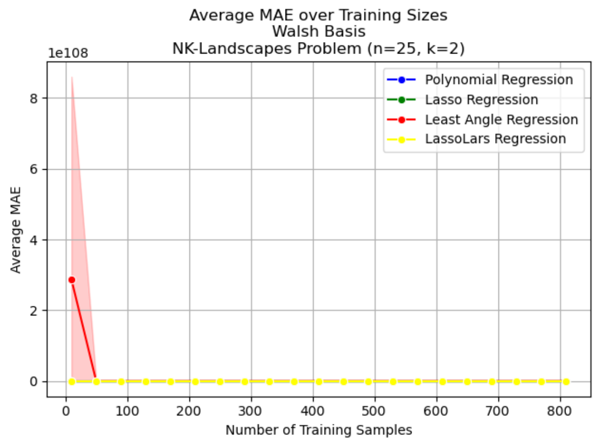


(b) Walsh Basis Model

Figure 12: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=25 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 11 except both basis start the Lars model from the second sample interval.

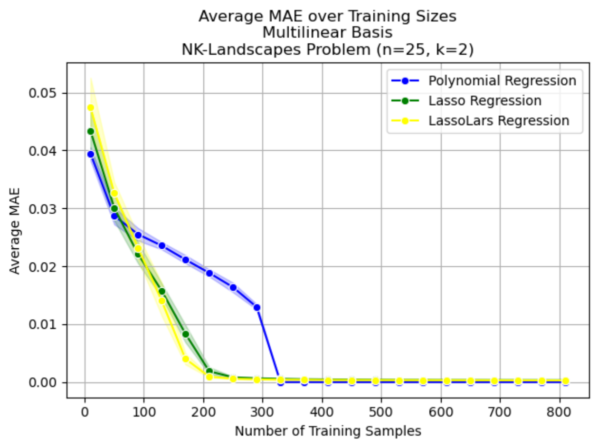


(a) Multilinear Basis Models

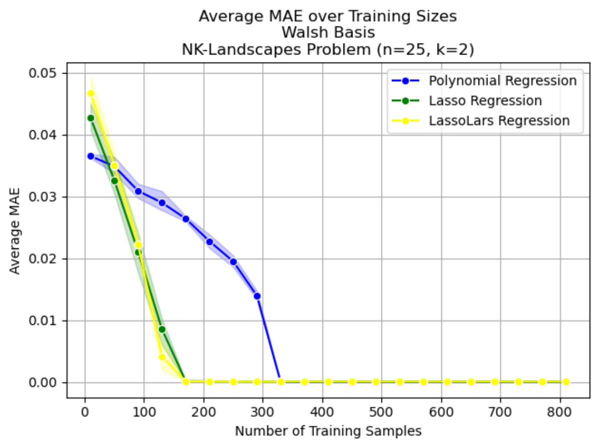


(b) Walsh Basis Models

Figure 13: Mean absolute error over training sizes on the NK-landscapes problem, for n=25 and k=2 with confidence interval (95%).

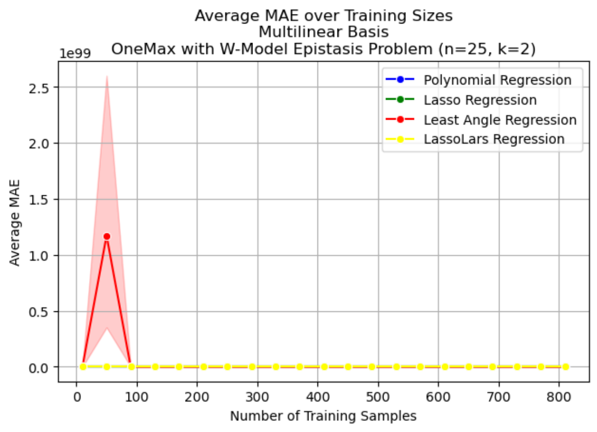


(a) Multilinear Basis Models

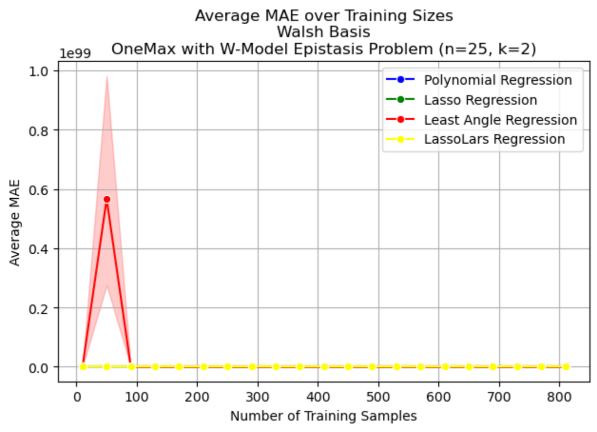


(b) Walsh Basis Models

Figure 14: Mean absolute error over training sizes on the NK-landscapes problem, for n=25 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 13 except they exclude the Lars model.

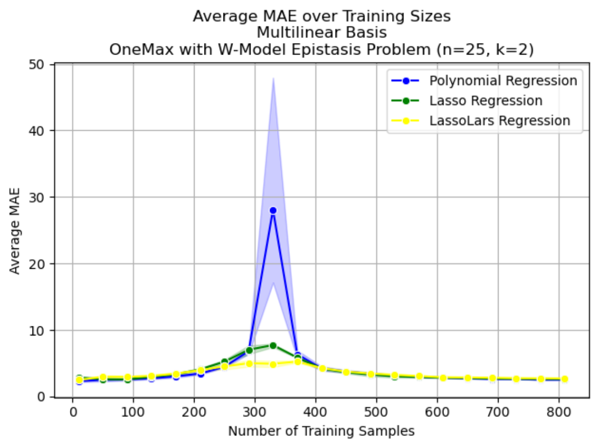


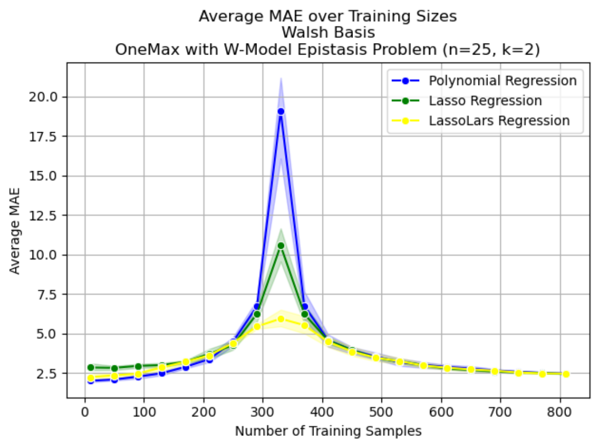
(a) Multilinear Basis Models



(b) Walsh Basis Models

Figure 15: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=25 and k=2 with confidence interval (95%).



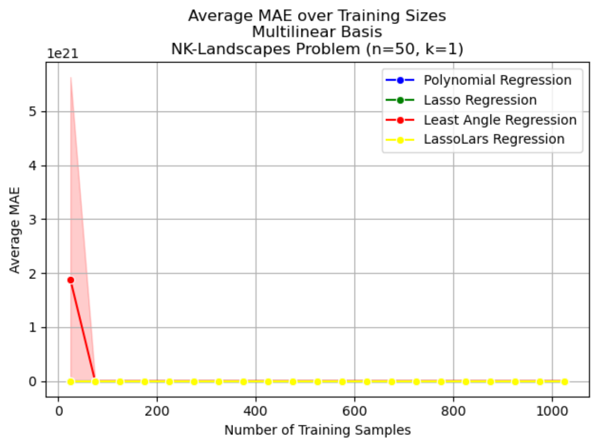


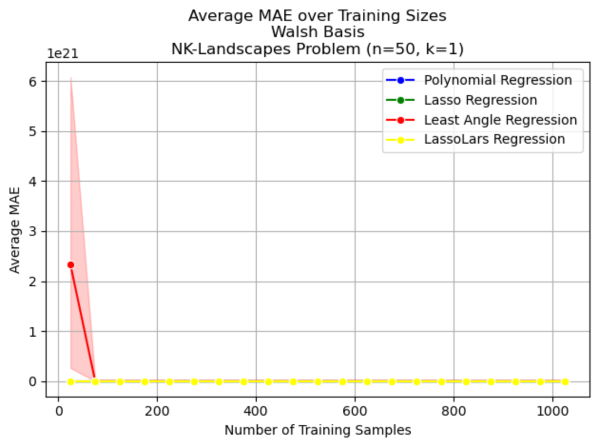
(a) Multilinear Basis Models

(b) Walsh Basis Models

Figure 16: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=25 and k=2 with confidence interval (95%). These graphs are the same as those in figure 15 except they exclude the Lars model.

7.1.3 Results for n=50

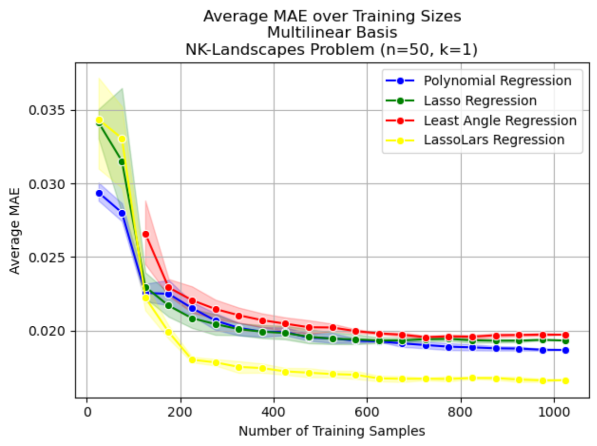




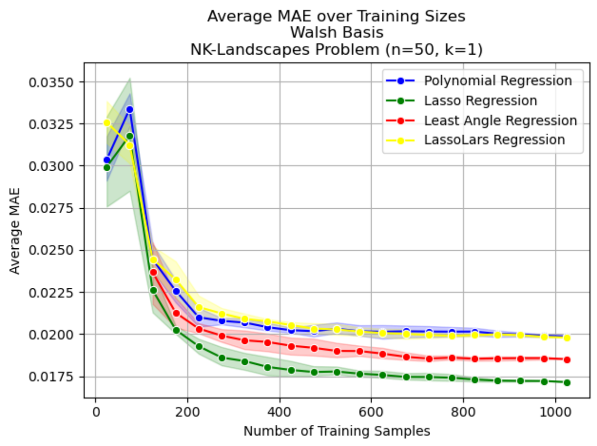
(a) Multilinear Basis Models

(b) Walsh Basis Models

Figure 17: Mean absolute error over training sizes on the NK-landscapes problem, for n=50 and k=1 with confidence interval (95%).

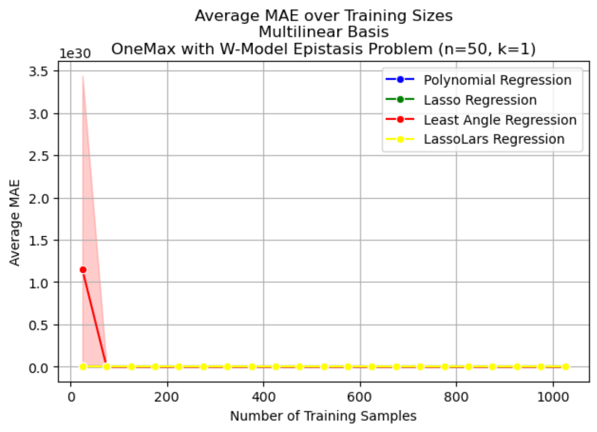


(a) Multilinear Basis Models

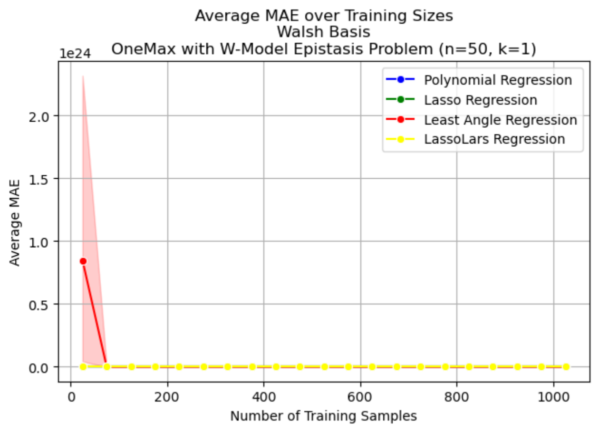


(b) Walsh Basis Models

Figure 18: Mean absolute error over training sizes on the NK-landscapes problem, for n=50 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 17 except the both basis start the Lars model at the second sample interval.

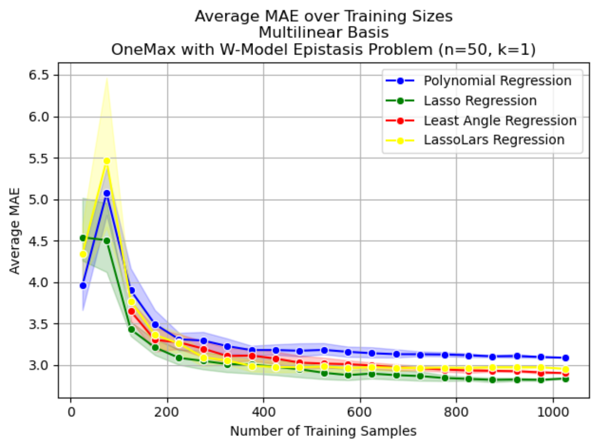


(a) Multilinear Basis Models

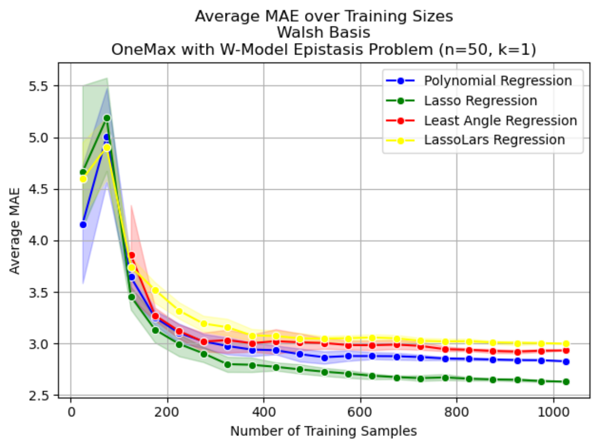


(b) Walsh Basis Models

Figure 19: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=50 and k=1 with confidence interval (95%).

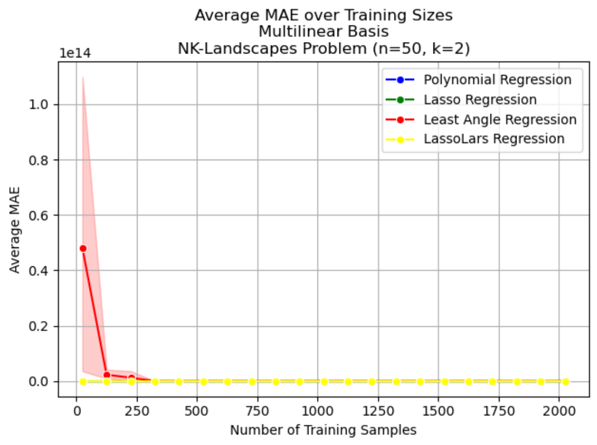


(a) Multilinear Basis Models

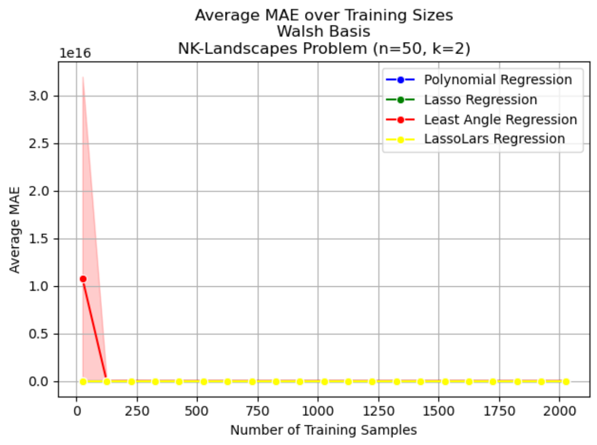


(b) Walsh Basis Models

Figure 20: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=50 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 19 but starting the Lars model at the second sample interval.

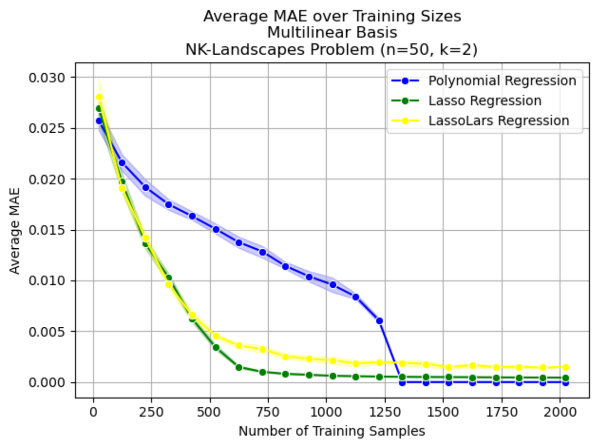


(a) Multilinear Basis Models

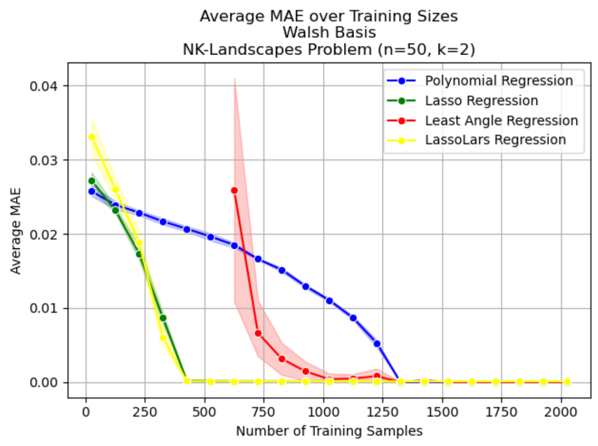


(b) Walsh Basis Models

Figure 21: Mean absolute error over training sizes on the NK-landscapes problem, for n=50 and k=2 with confidence interval (95%).

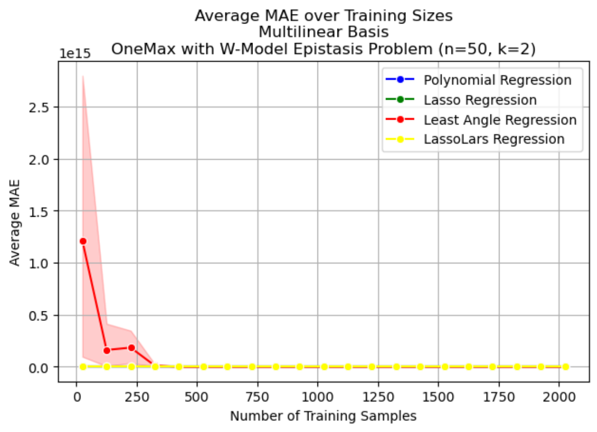


(a) Multilinear Basis Models

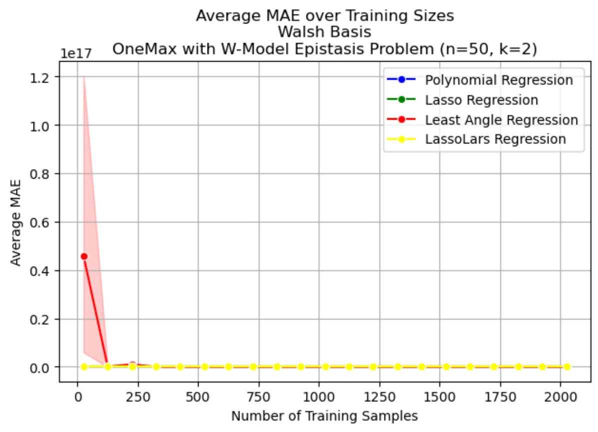


(b) Walsh Basis Models

Figure 22: Mean absolute error over training sizes on the NK-landscapes problem, for n=50 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 21 except the multilinear basis excludes the Lars model and the Walsh basis starts the Lars model from the sixth sample interval.

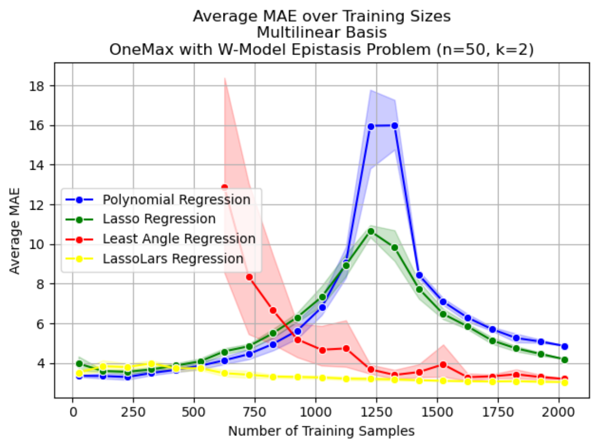


(a) Multilinear Basis Models

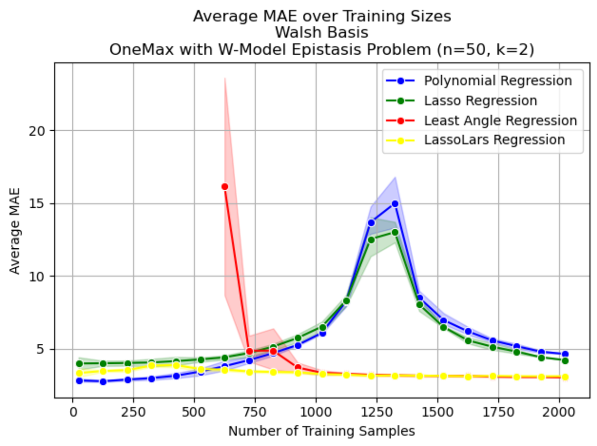


(b) Walsh Basis Models

Figure 23: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=50 and k=2 with confidence interval (95%).



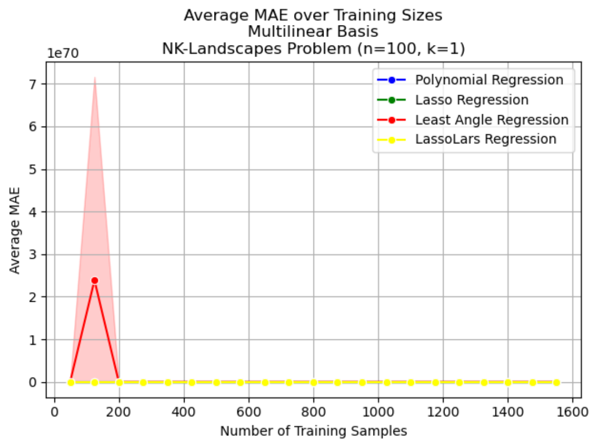
(a) Multilinear Basis Models



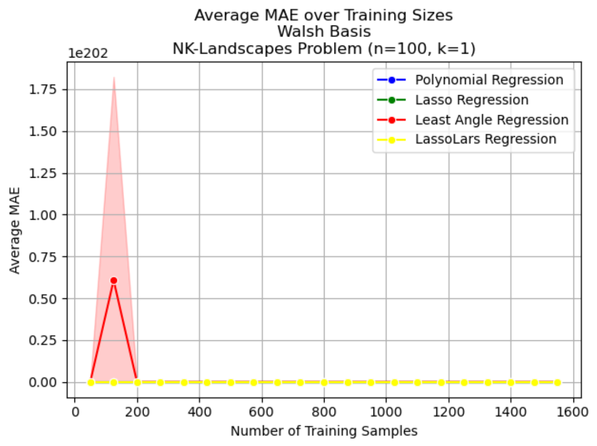
(b) Walsh Basis Models

Figure 24: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=50 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 23 except both basis start the Lars model from the sixth sample interval.

7.1.4 Results for n=100

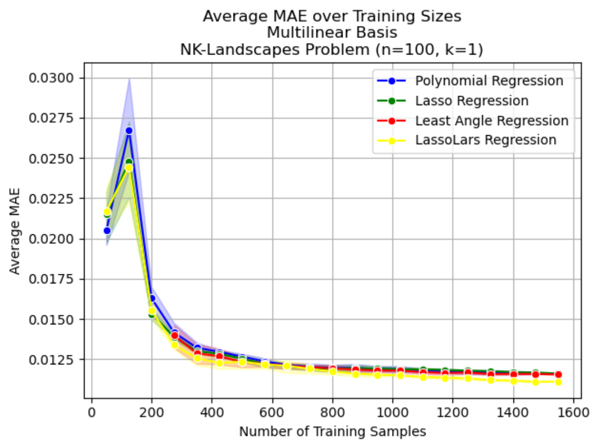


(a) Multilinear Basis Models

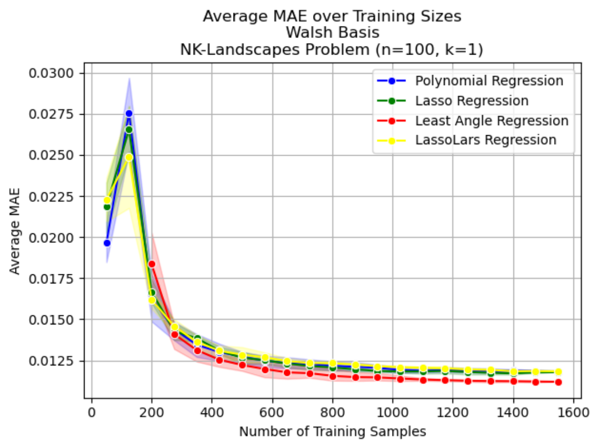


(b) Walsh Basis Models

Figure 25: Mean absolute error over training sizes on the NK-landscapes problem, for n=100 and k=1 with confidence interval (95%).

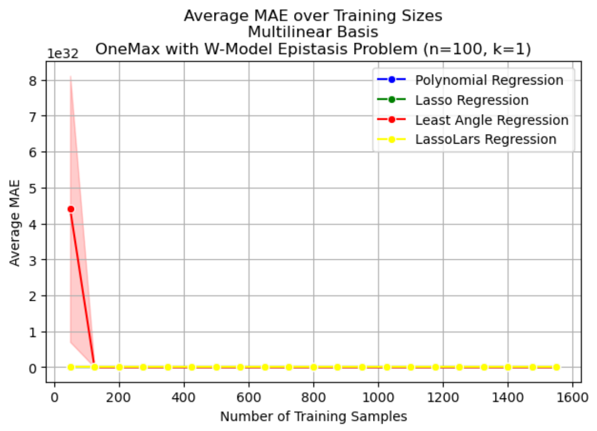


(a) Multilinear Basis Models

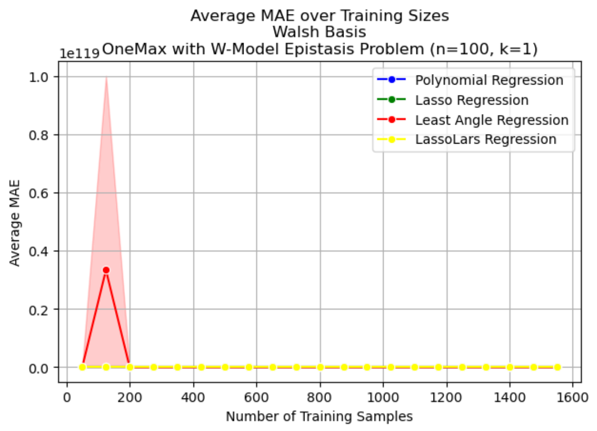


(b) Walsh Basis Models

Figure 26: Mean absolute error over training sizes on the NK-landscapes problem, for n=100 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 25 but in the Lars model starts the the third sample interval for the multilinear basis and the second sample interval for the Walsh basis.

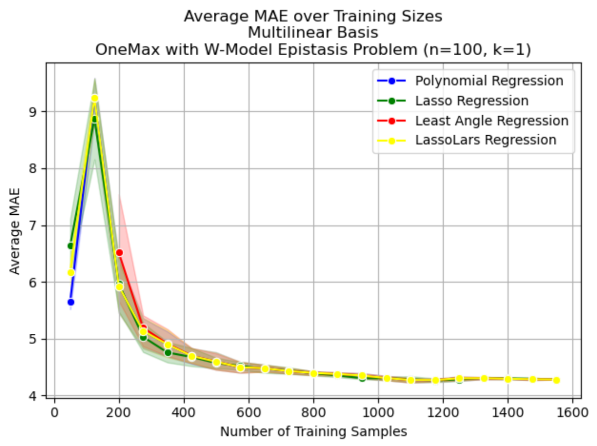


(a) Multilinear Basis Models

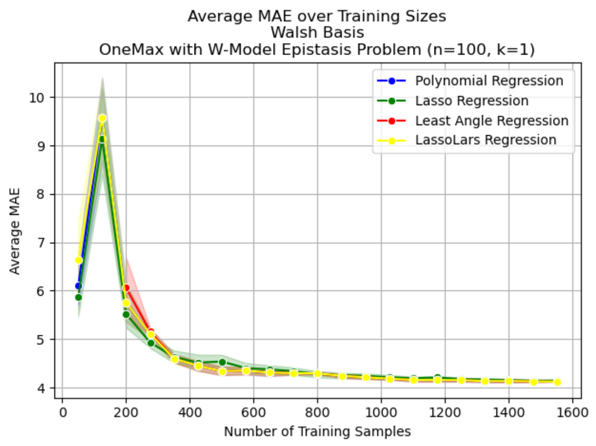


(b) Walsh Basis Models

Figure 27: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=100 and k=1 with confidence interval (95%).

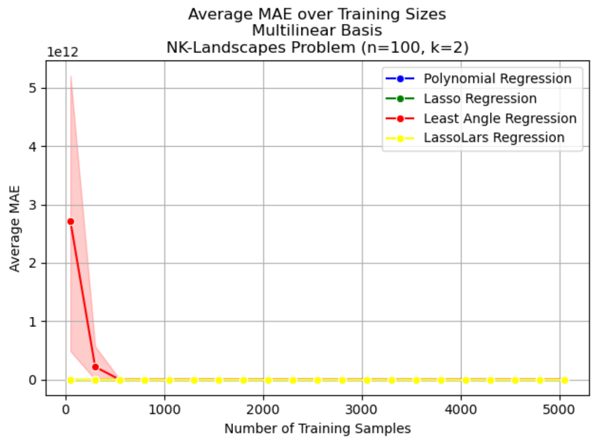


(a) Multilinear Basis Models

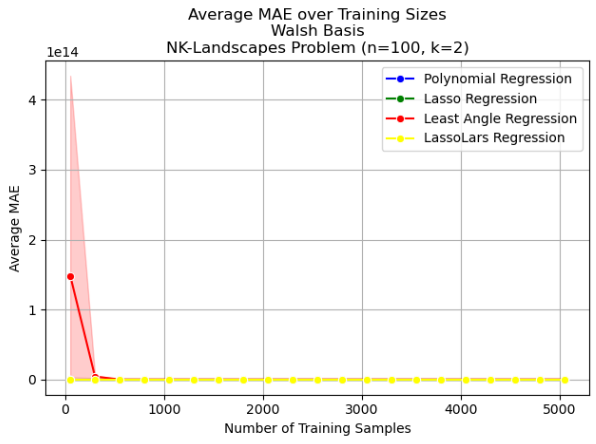


(b) Walsh Basis Models

Figure 28: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=100 and k=1 with confidence interval (95%). These graphs are the same as those in Figure 27 but the Lars model starts at the second sample interval.

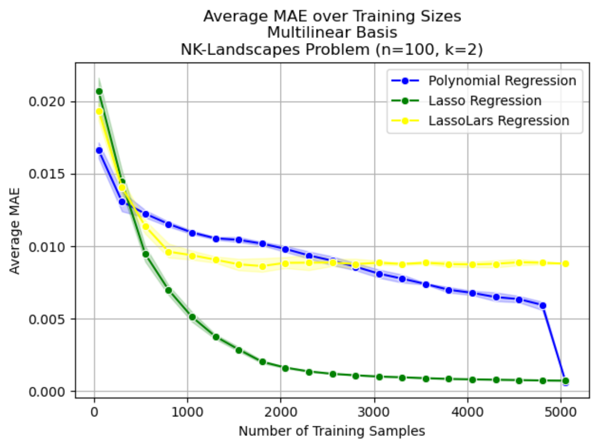


(a) Multilinear Basis Models

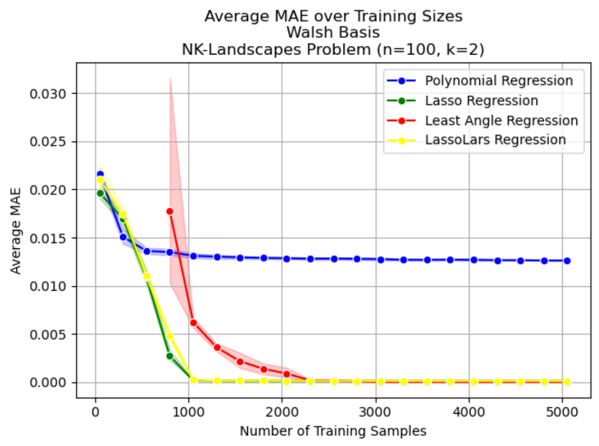


(b) Walsh Basis Models

Figure 29: Mean absolute error over training sizes on the NK-landscapes problem, for n=100 and k=2 with confidence interval (95%).

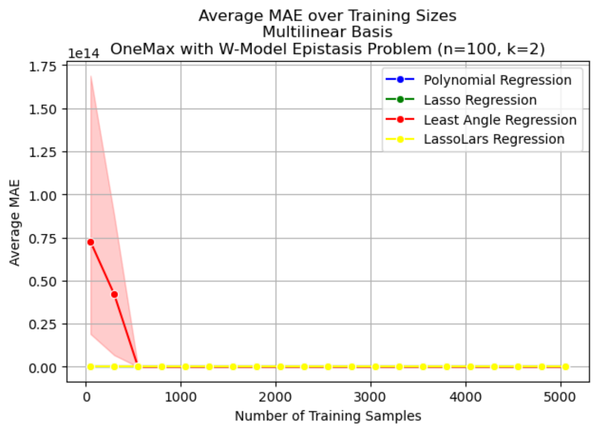


(a) Multilinear Basis Models

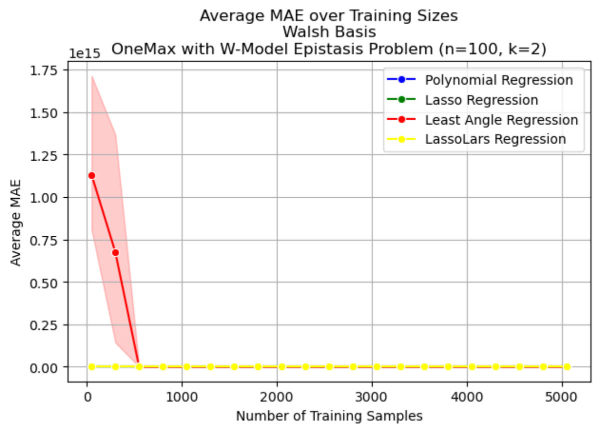


(b) Walsh Basis Models

Figure 30: Mean absolute error over training sizes on the NK-landscapes problem, for n=100 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 29 except the multilinear basis excludes the Lars model and the Walsh basis starts the Lars model at the third sample interval.

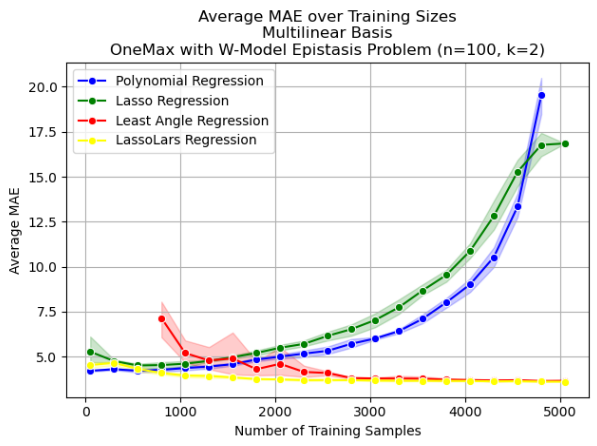


(a) Multilinear Basis Models

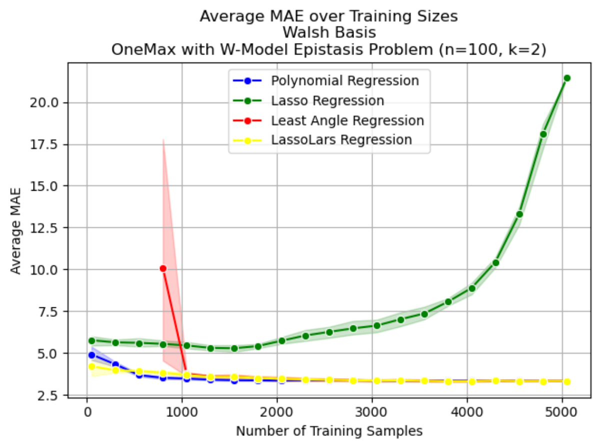


(b) Walsh Basis Models

Figure 31: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=100 and k=2 with confidence interval (95%).



(a) Multilinear Basis Models



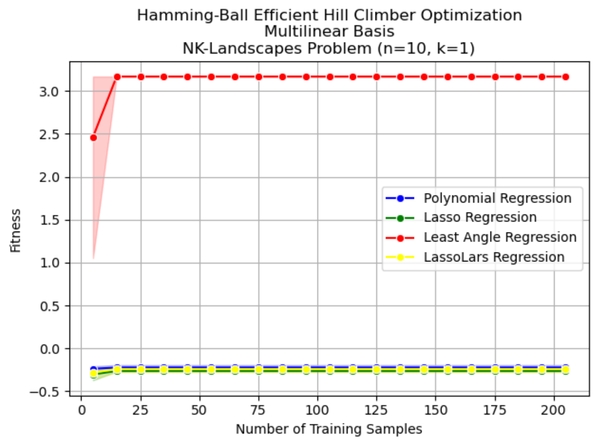
(b) Walsh Basis Models

Figure 32: Mean absolute error over training sizes on the OneMax with w-model epistasis problem, for n=50 and k=2 with confidence interval (95%). These graphs are the same as those in Figure 31 except the multilinear models starts the Lars model from the second sample interval and removes the final sample interval from the polynomial model, and the Walsh basis starts the Lars model from the second sample interval.

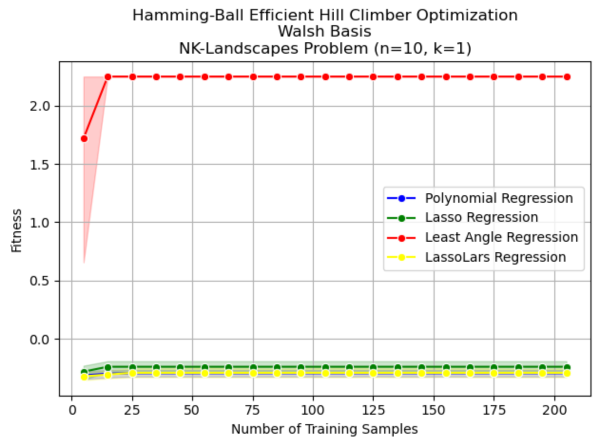
7.2 Optimization

All the surrogate models will now undergo optimization, using the Hamming-ball efficient hill-climber. For most of the graphs, there is a second version excluding Lars as it skews the graph, making it difficult to view the other results.

7.2.1 Results for n=10

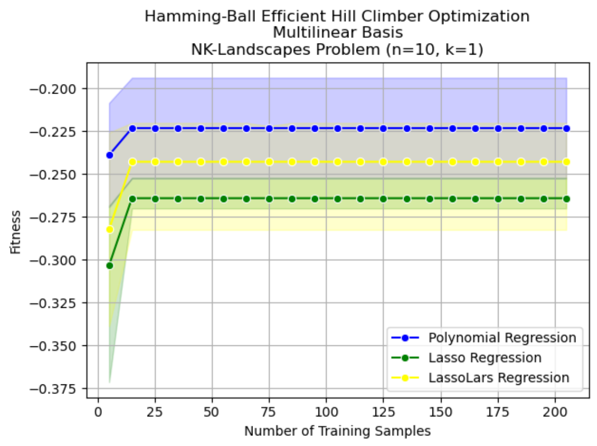


(a) Multilinear Basis Models

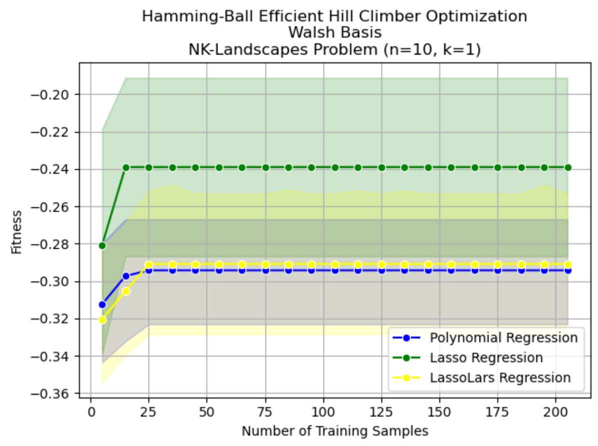


(b) Walsh Basis Models

Figure 33: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=1 with confidence interval (95%).

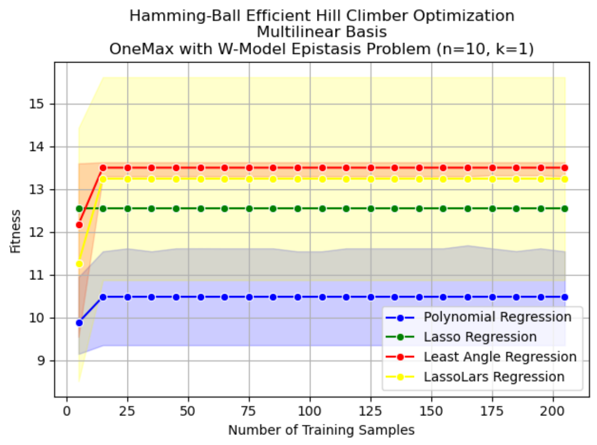


(a) Multilinear Basis Models

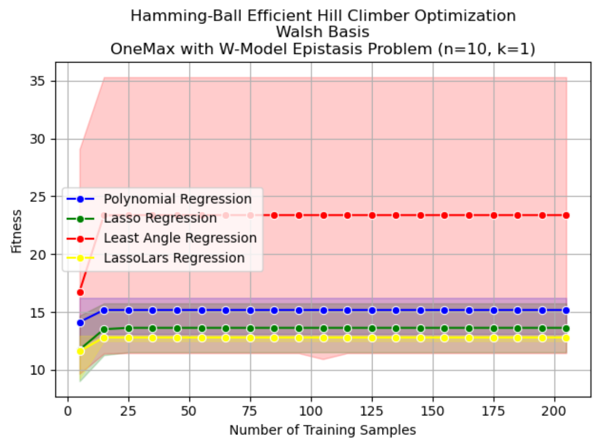


(b) Walsh Basis Models

Figure 34: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=1 with confidence interval (95%). These graphs are the same as Figure 33 excluding the Lars model.

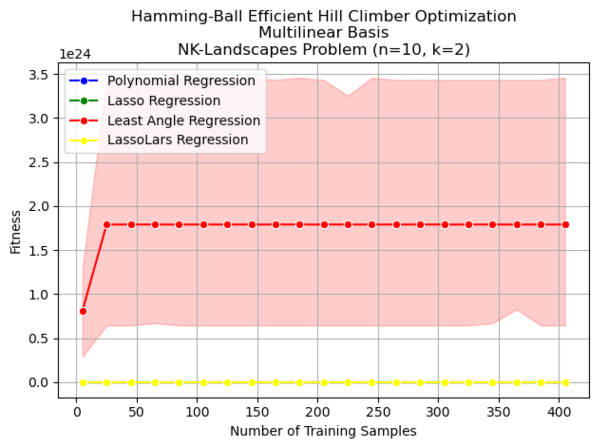


(a) Multilinear Basis Models

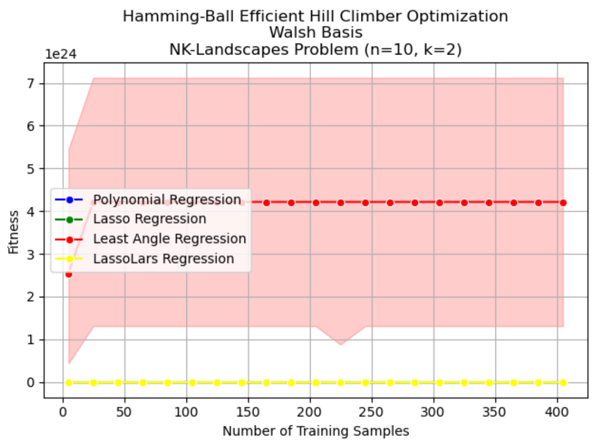


(b) Walsh Basis Models

Figure 35: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=1 with confidence interval (95%).

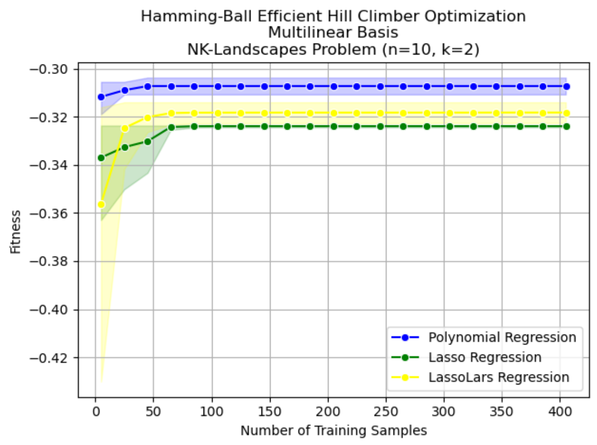


(a) Multilinear Basis Models

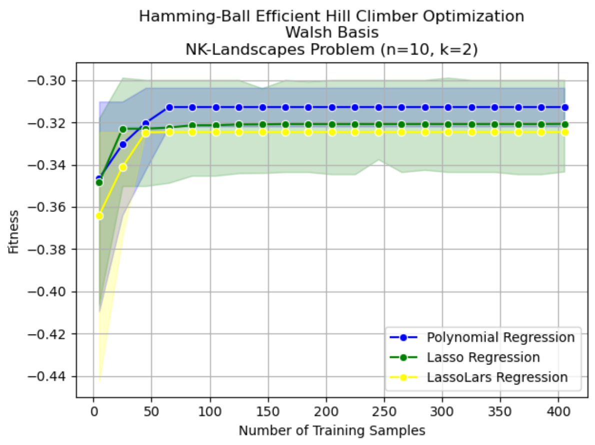


(b) Walsh Basis Models

Figure 36: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=2 with confidence interval (95%).

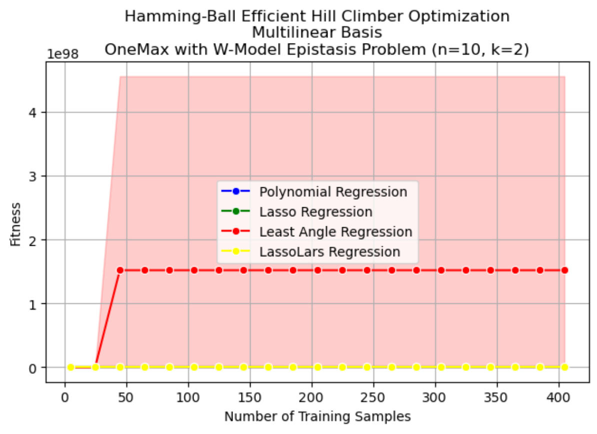


(a) Multilinear Basis Models

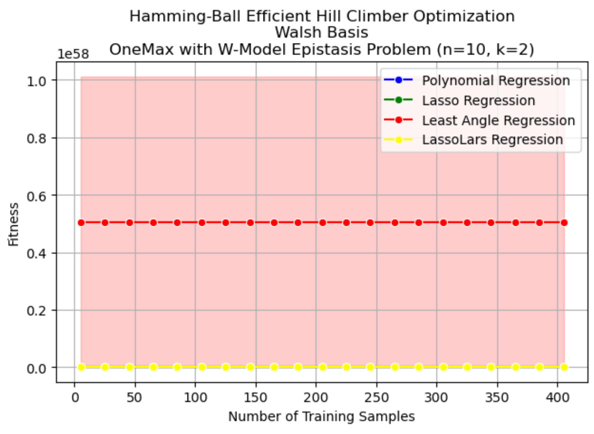


(b) Walsh Basis Models

Figure 37: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=2 with confidence interval (95%). These graphs are the same as Figure 36 excluding the Lars model.

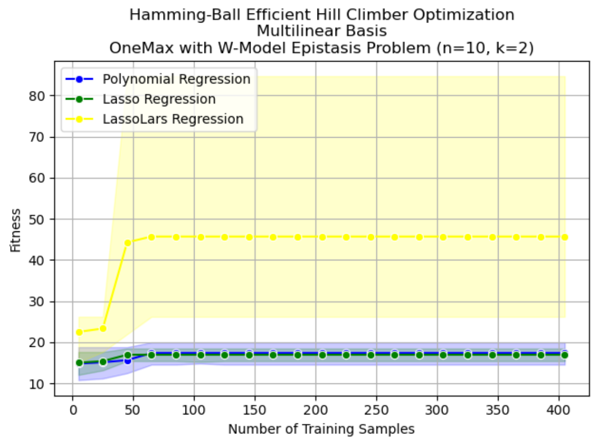


(a) Multilinear Basis Models

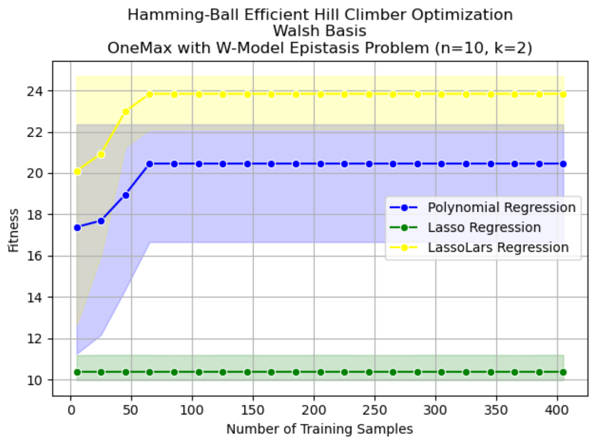


(b) Walsh Basis Models

Figure 38: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=2 with confidence interval (95%).



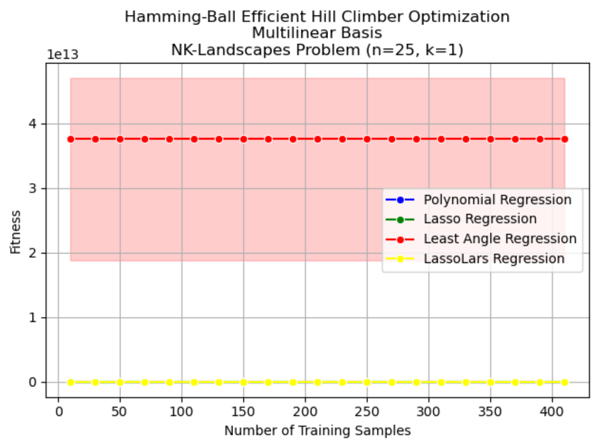
(a) Multilinear Basis Models



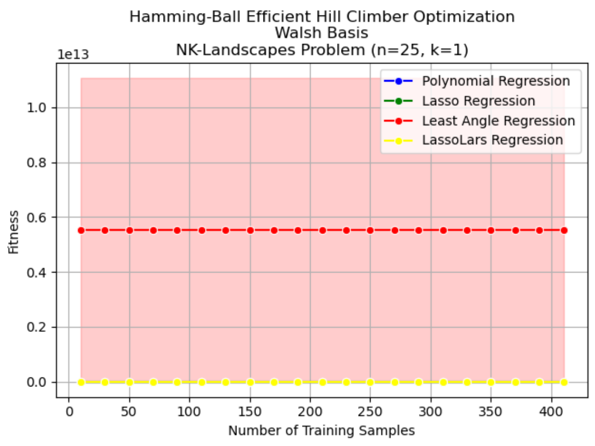
(b) Walsh Basis Models

Figure 39: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=10 and k=2 with confidence interval (95%). These graphs are the same as Figure 38 excluding the Lars model.

7.2.2 Results for n=25

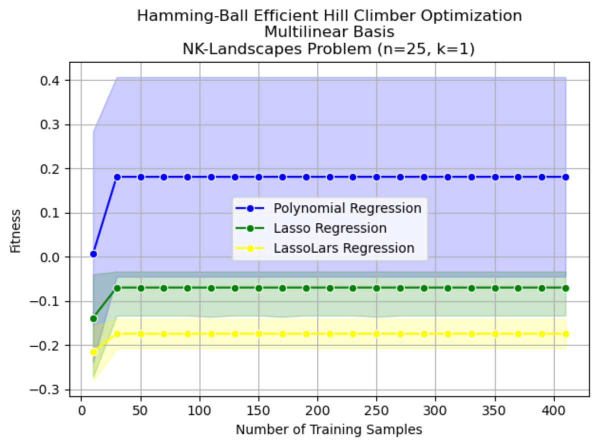


(a) Multilinear Basis Models

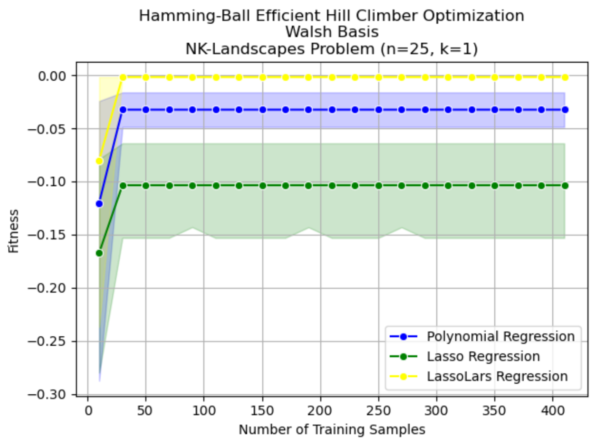


(b) Walsh Basis Models

Figure 40: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=1 with confidence interval (95%).

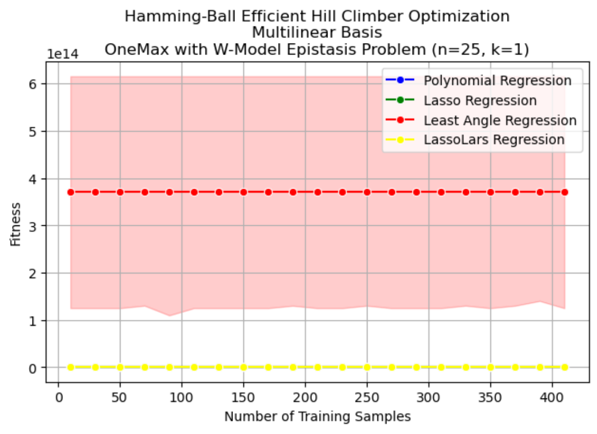


(a) Multilinear Basis Models

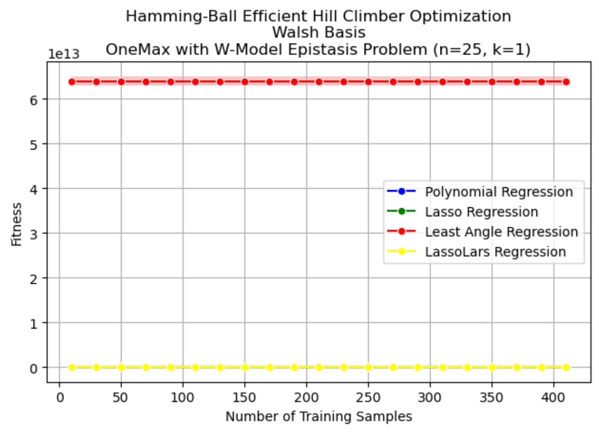


(b) Walsh Basis Models

Figure 41: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=1 with confidence interval (95%). These graphs are the same as Figure 40 excluding the Lars model.

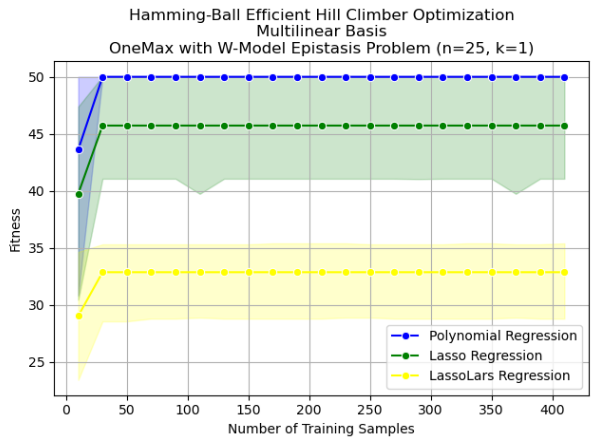


(a) Multilinear Basis Models

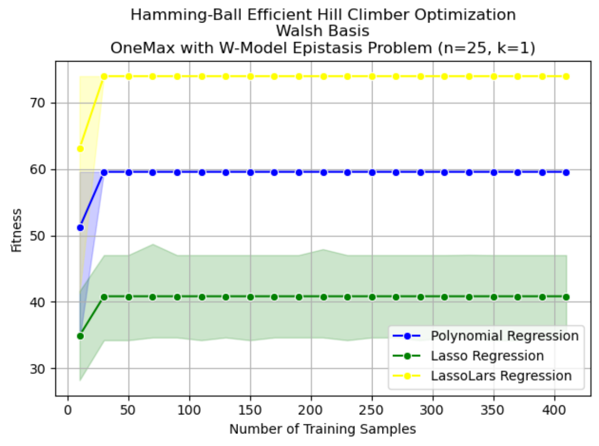


(b) Walsh Basis Models

Figure 42: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=1 with confidence interval (95%).

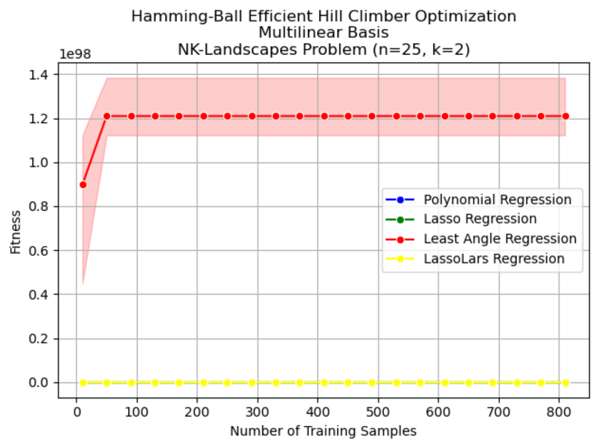


(a) Multilinear Basis Models

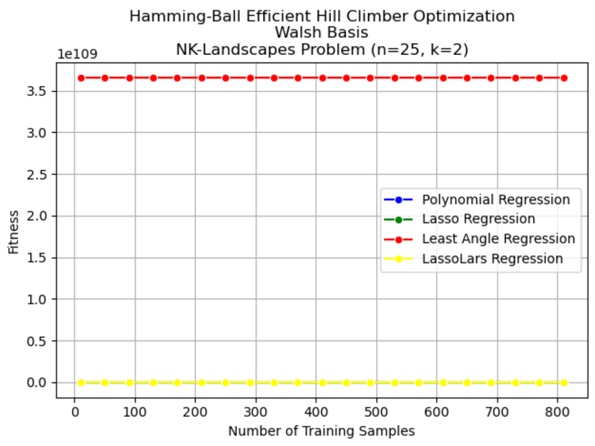


(b) Walsh Basis Models

Figure 43: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=1 with confidence interval (95%). These graphs are the same as Figure 42 excluding the Lars model.

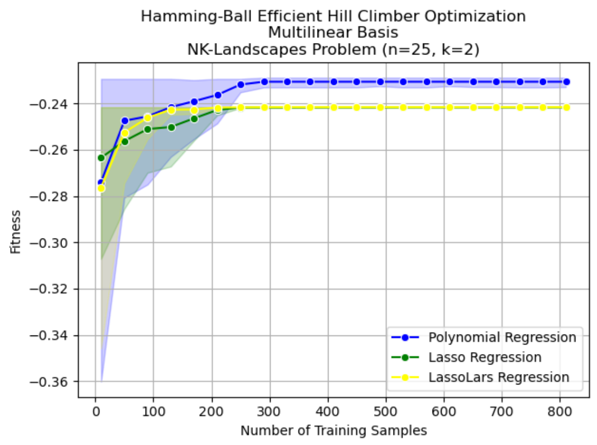


(a) Multilinear Basis Models

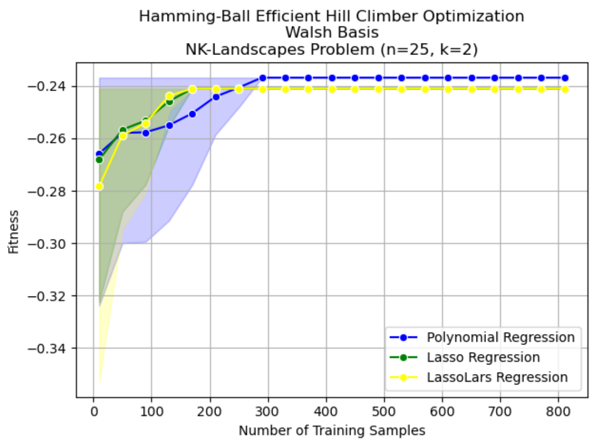


(b) Walsh Basis Models

Figure 44: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=2 with confidence interval (95%).

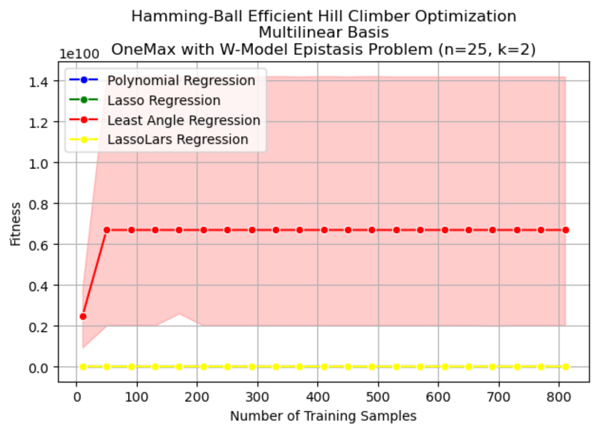


(a) Multilinear Basis Models

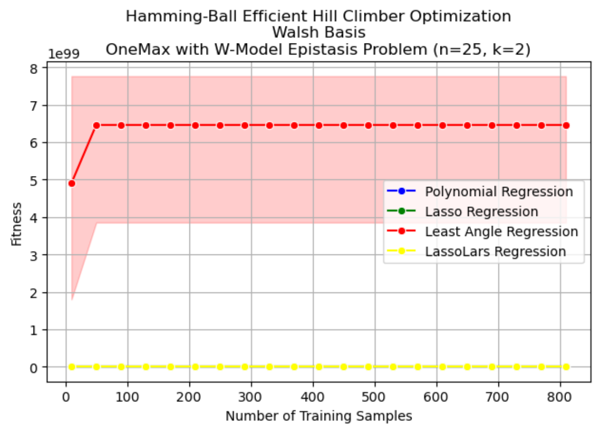


(b) Walsh Basis Models

Figure 45: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=2 with confidence interval (95%). These graphs are the same as Figure 44 excluding the Lars model.

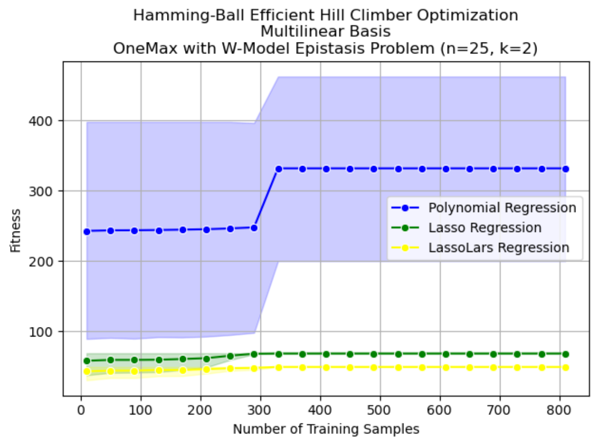


(a) Multilinear Basis Models

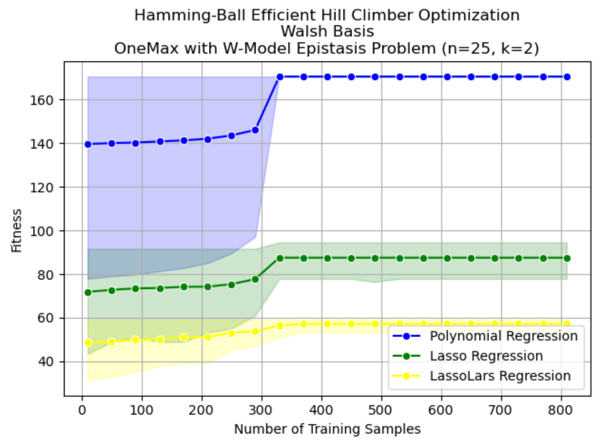


(b) Walsh Basis Models

Figure 46: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=2 with confidence interval (95%).



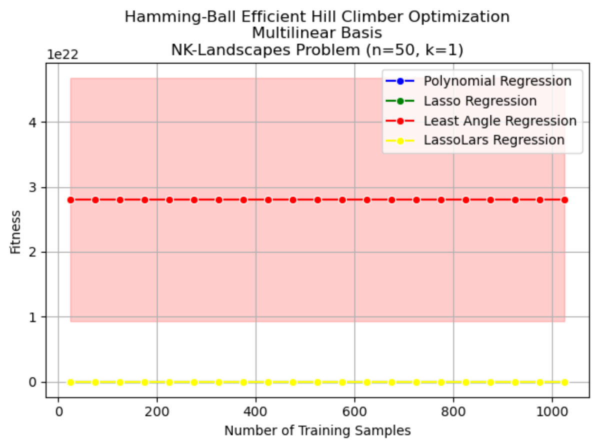
(a) Multilinear Basis Models



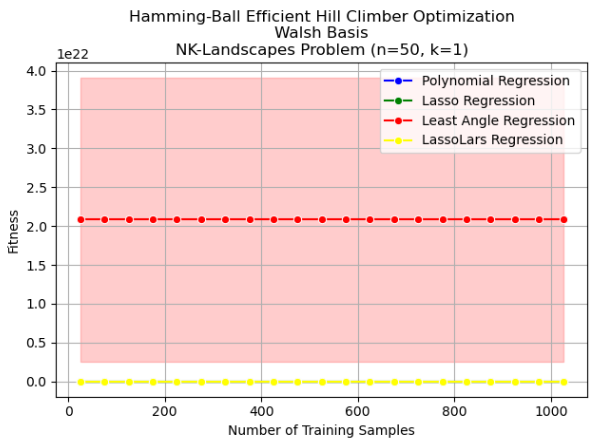
(b) Walsh Basis Models

Figure 47: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=25 and k=2 with confidence interval (95%). These graphs are the same as Figure 46 excluding the Lars model.

7.2.3 Results for n=50

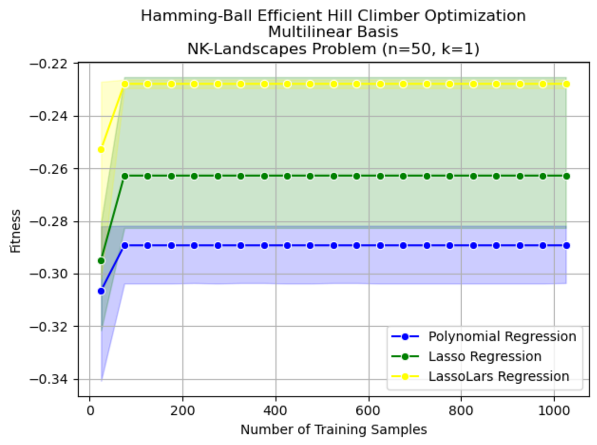


(a) Multilinear Basis Models

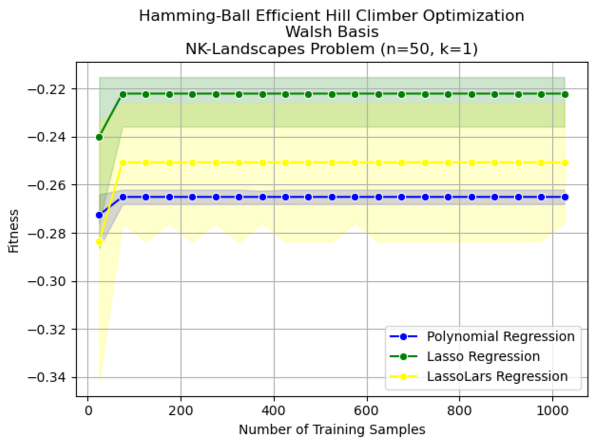


(b) Walsh Basis Models

Figure 48: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=1 with confidence interval (95%).

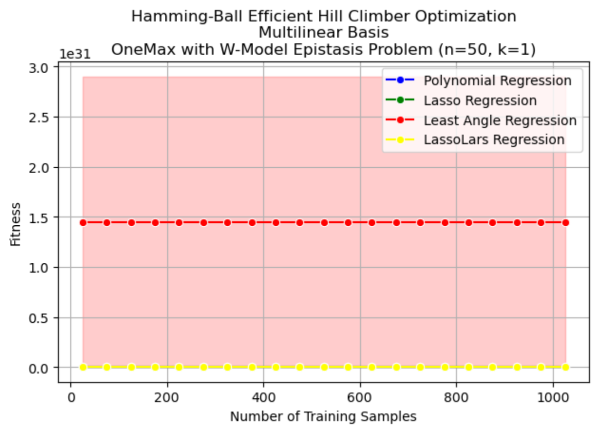


(a) Multilinear Basis Models

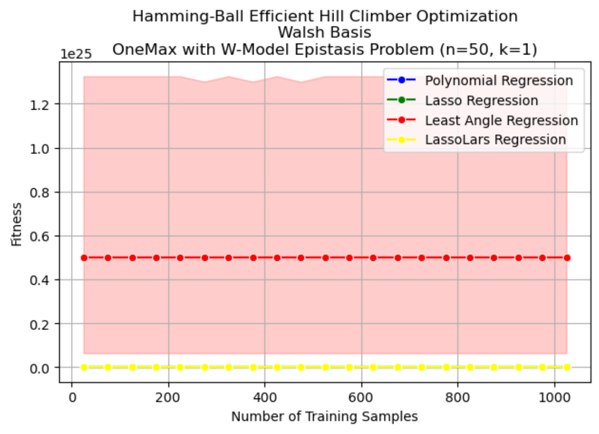


(b) Walsh Basis Models

Figure 49: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=1 with confidence interval (95%). These graphs are the same as Figure 48 excluding the Lars model.

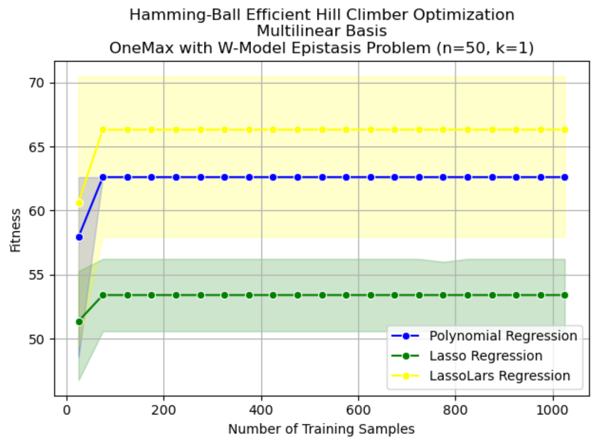


(a) Multilinear Basis Models

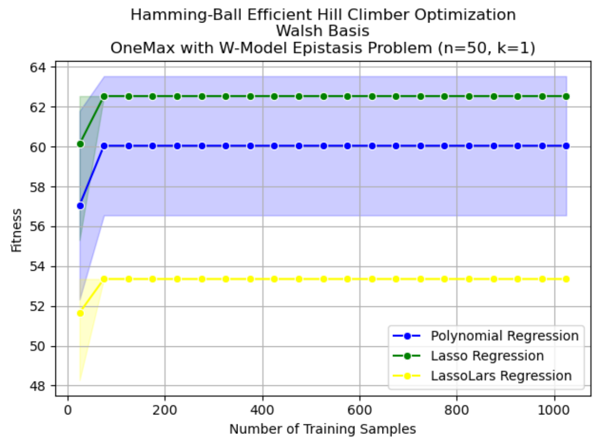


(b) Walsh Basis Models

Figure 50: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=1 with confidence interval (95%).

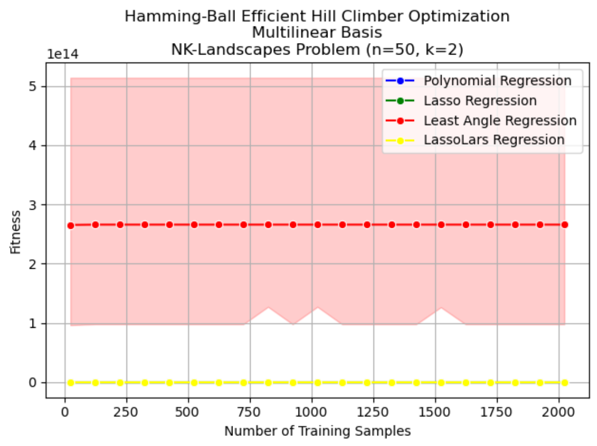


(a) Multilinear Basis Models

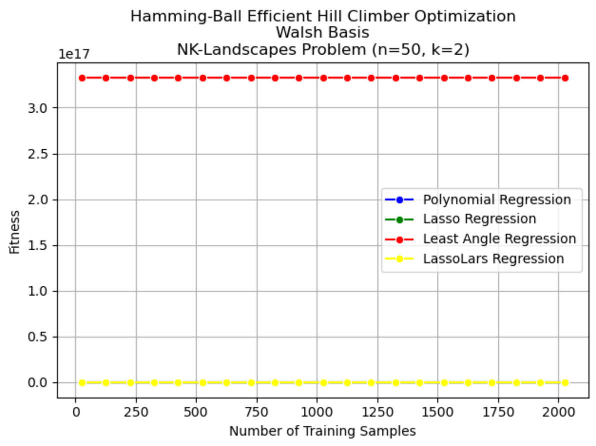


(b) Walsh Basis Models

Figure 51: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=1 with confidence interval (95%). These graphs are the same as Figure 50 excluding the Lars model.

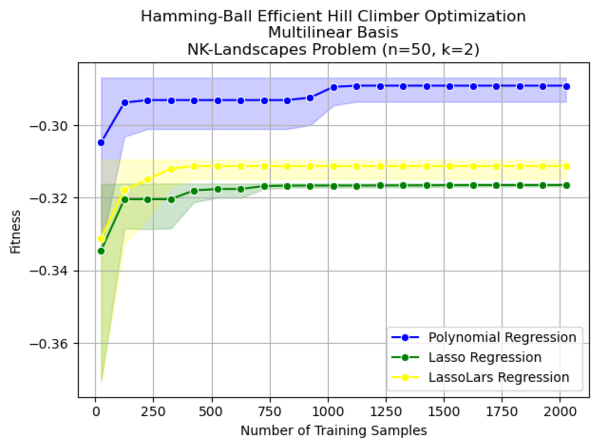


(a) Multilinear Basis Models

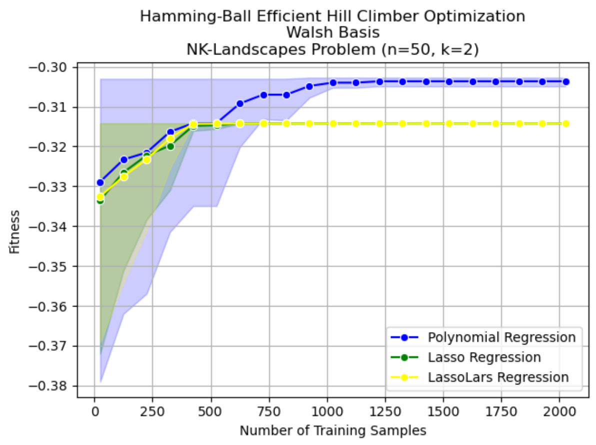


(b) Walsh Basis Models

Figure 52: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=2 with confidence interval (95%).

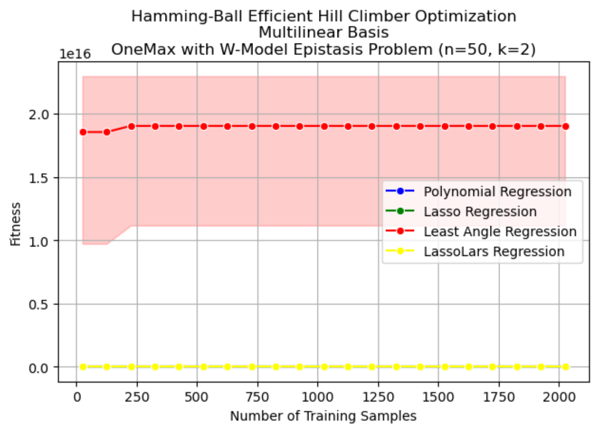


(a) Multilinear Basis Models

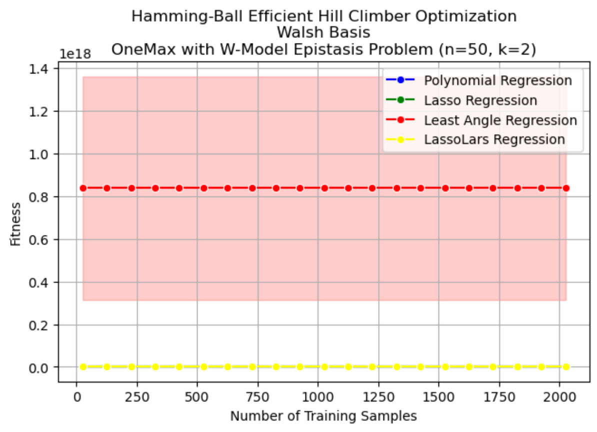


(b) Walsh Basis Models

Figure 53: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=2 with confidence interval (95%). These graphs are the same as Figure 52 excluding the Lars model.

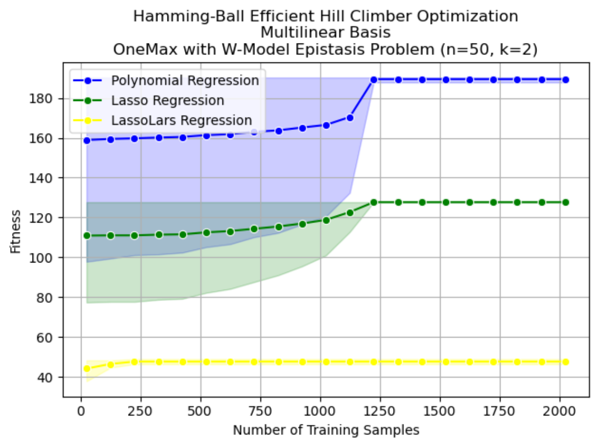


(a) Multilinear Basis Models

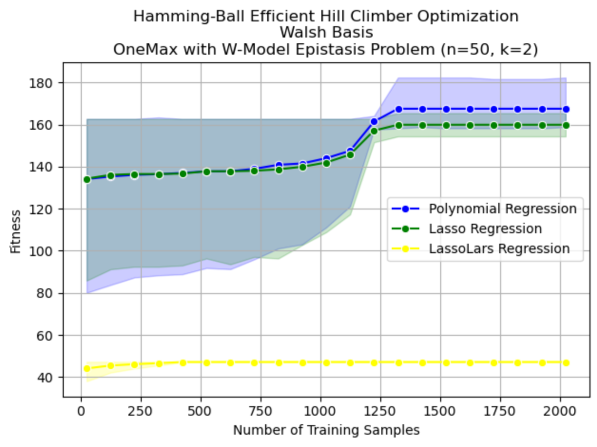


(b) Walsh Basis Models

Figure 54: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=2 with confidence interval (95%).



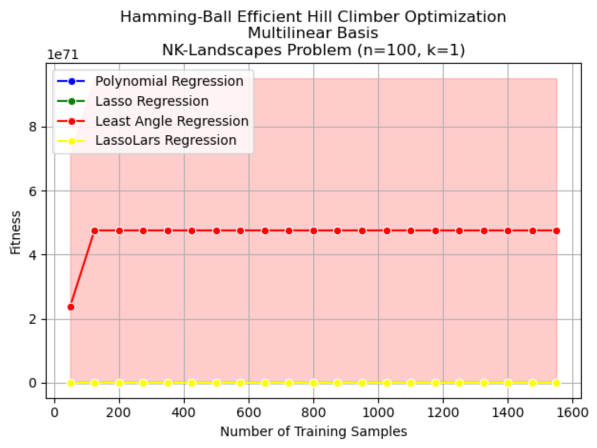
(a) Multilinear Basis Models



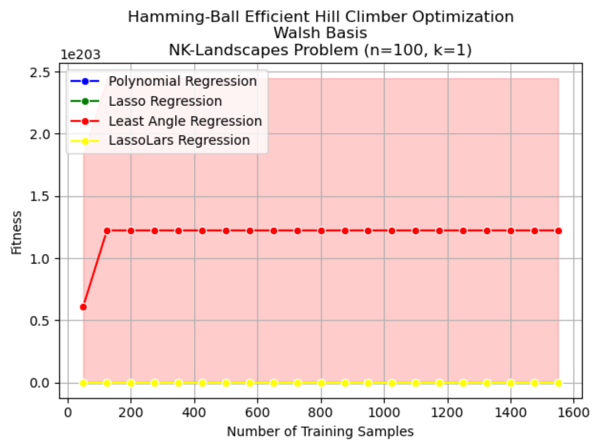
(b) Walsh Basis Models

Figure 55: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=50 and k=2 with confidence interval (95%). These graphs are the same as Figure 54 excluding the Lars model.

7.2.4 Results for n=100

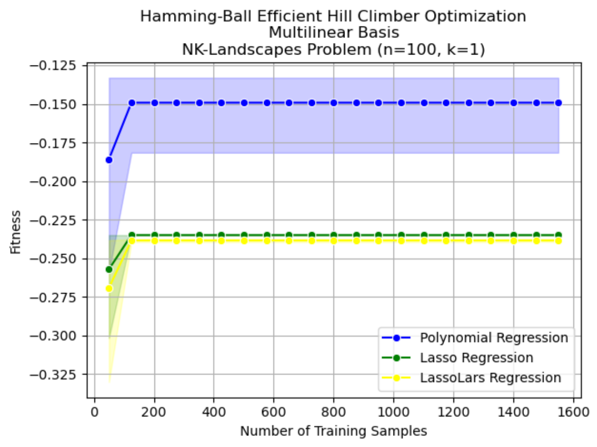


(a) Multilinear Basis Models

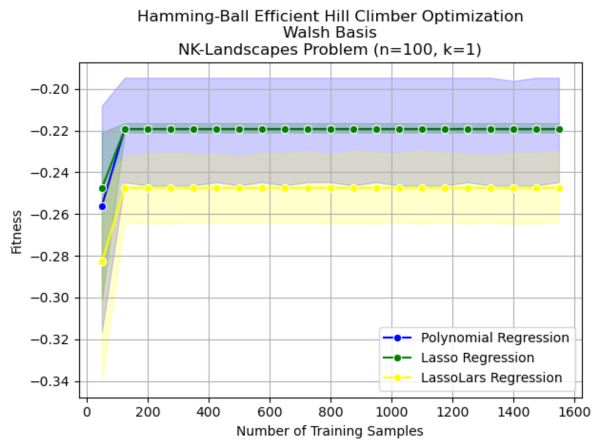


(b) Walsh Basis Models

Figure 56: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=1 with confidence interval (95%).

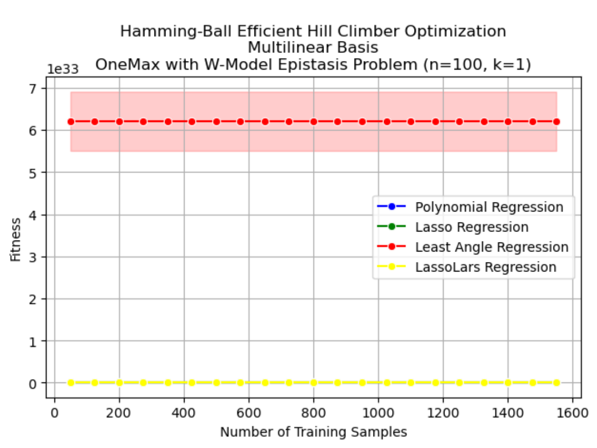


(a) Multilinear Basis Models

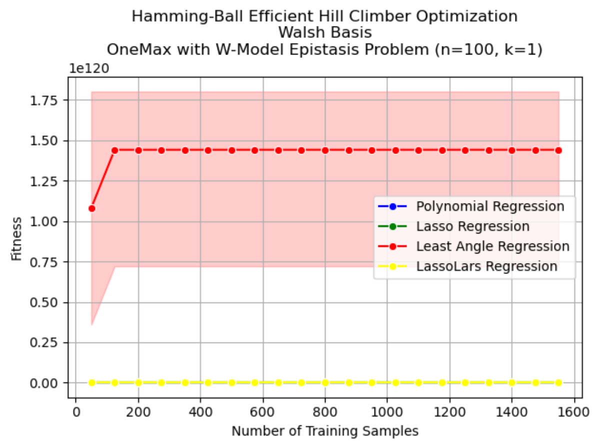


(b) Walsh Basis Models

Figure 57: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=1 with confidence interval (95%). These graphs are the same as Figure 56 excluding the Lars model.

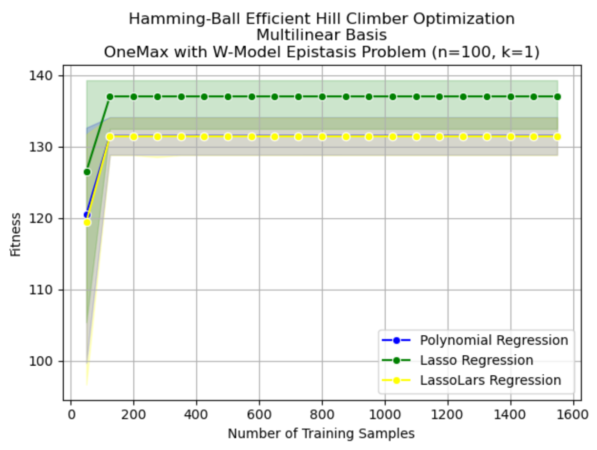


(a) Multilinear Basis Models

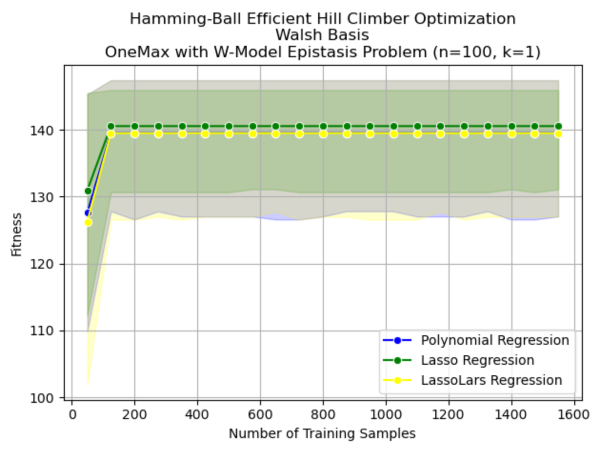


(b) Walsh Basis Models

Figure 58: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=1 with confidence interval (95%).

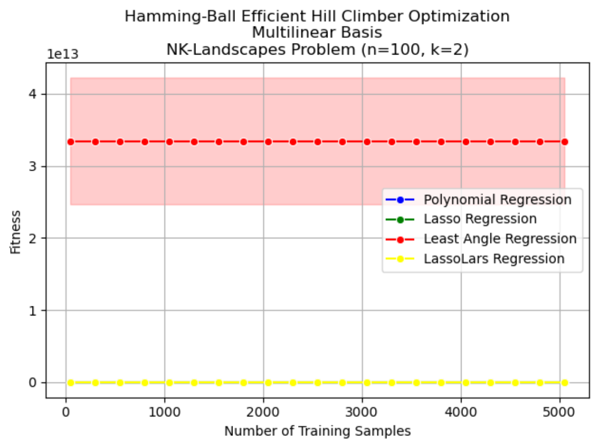


(a) Multilinear Basis Models

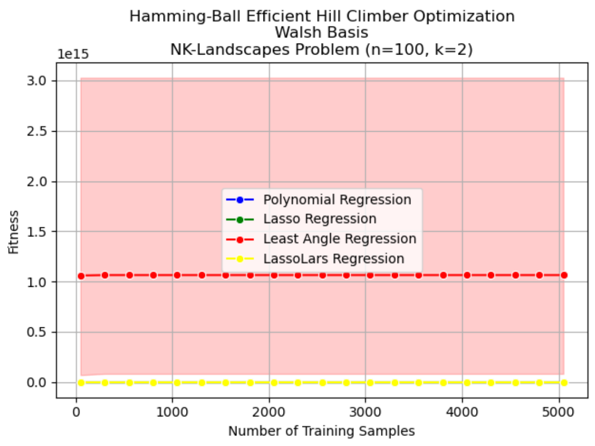


(b) Walsh Basis Models

Figure 59: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=1 with confidence interval (95%). These graphs are the same as Figure 58 excluding the Lars model.

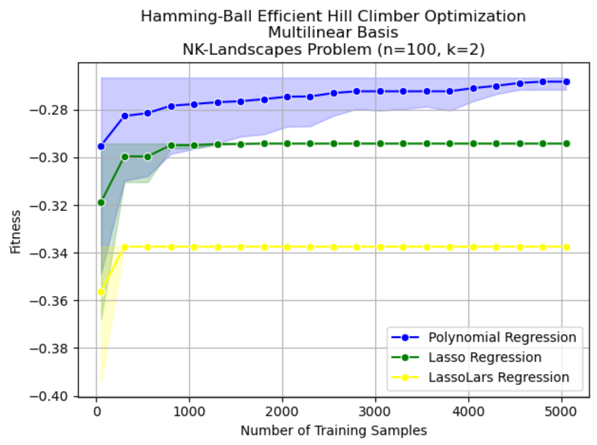


(a) Multilinear Basis Models

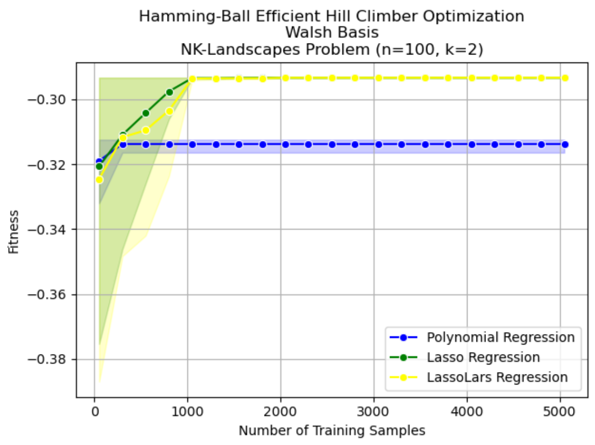


(b) Walsh Basis Models

Figure 60: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=2 with confidence interval (95%).

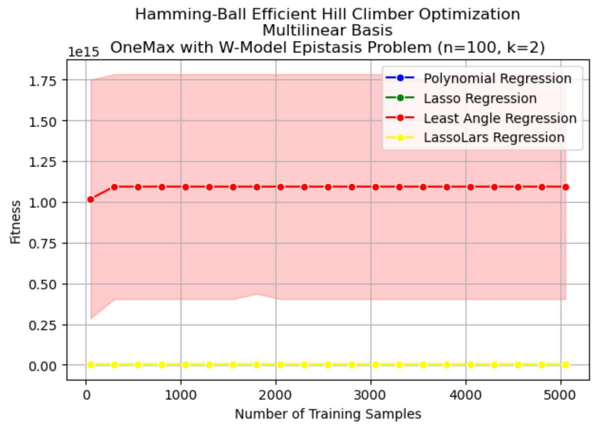


(a) Multilinear Basis Models

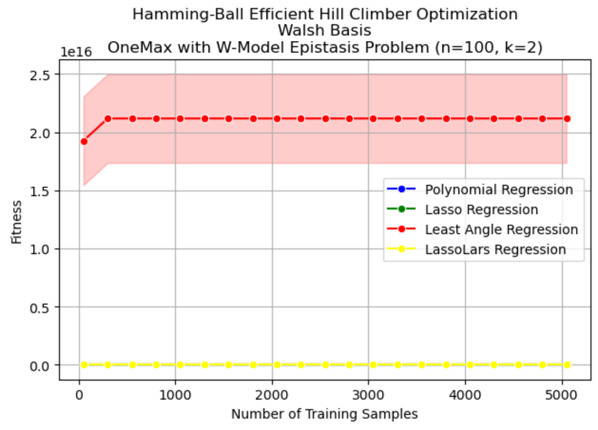


(b) Walsh Basis Models

Figure 61: Best found fitness over training sizes on the NK-landscapes problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=2 with confidence interval (95%). These graphs are the same as Figure 60 excluding the Lars model.

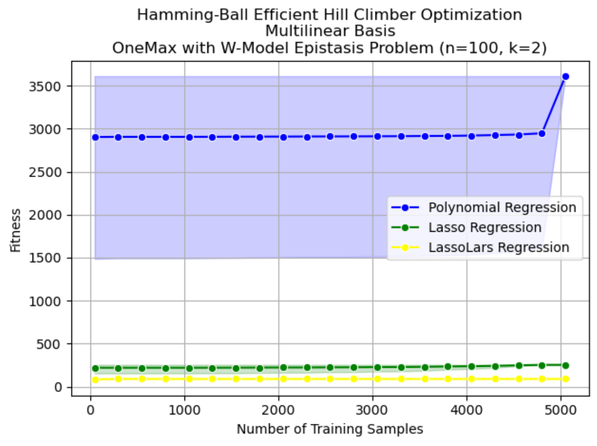


(a) Multilinear Basis Models

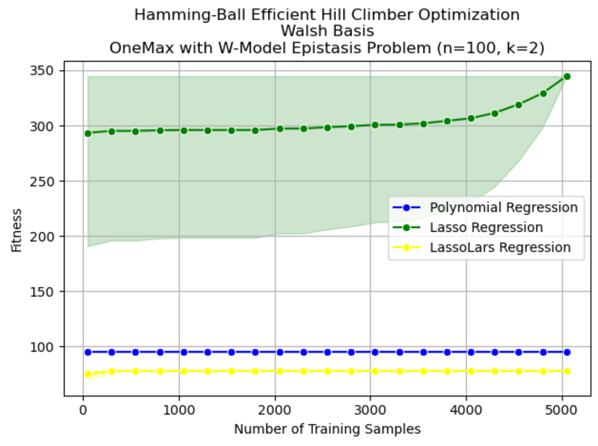


(b) Walsh Basis Models

Figure 62: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=2 with confidence interval (95%).



(a) Multilinear Basis Models



(b) Walsh Basis Models

Figure 63: Best found fitness over training sizes on the OneMax with w-model epistasis problem using the Hamming-ball efficient hill-climber optimization, for n=100 and k=2 with confidence interval (95%). These graphs are the same as Figure 62 excluding the Lars model.

7.3 Results Discussion

Overall, the best performing models in the approximation sections were the Lasso and LassoLars models with the Walsh basis. Throughout the results, the Walsh basis consistently had lower MAE scores than the multilinear basis, regardless of which regression surrogate and the benchmark. When comparing the benchmarks, it is clear that the OneMax w-model epistasis causes some confusion with the models, as unlike the NKL benchmark, the lines do not always maintain a general decrease in MAE as the sample size increases. This is likely due to the addition of new training data that, due to the transform, skew the results of the models, causing spikes as seen in Figure 16. When these irregularities occur, the LassoLars model seems to be the most resistant, as it shows the smallest spike, if any, compared to the other models. The Lasso model is the second most resistant to these errors, but still shows a significant increase in the MAE score. This type of behavior is not demonstrated in the NKL benchmark. It can be concluded from these results that the best suited model for the NKL benchmark is the Lasso model with the Walsh basis and the best suited model for the OneMax benchmark is the LassoLars model with the Walsh basis.

An interesting trend is that the Lars model struggles to maintain consistent MAE values, often showing spikes with large error that are at a much larger scale than the other models. This is likely due to the Lars model failing to converge on the smaller training sizes. As such, many of the graphs have the Lars model starting at a larger training size or removed entirely so that the results of the other models can be viewed clearly. Despite struggling to converge on the smaller training sizes, the Lars model is able to achieve MAE scores similar to the other models at the larger training sizes. The Lars model also performs better with $k = 1$ compared to $k = 2$. These trends can be seen in both benchmarks and in both basis, with little difference between. Interestingly, the polynomial regression model generally performs similarly to the other models, despite lack of sparsity. The polynomial surrogate does tend to have slightly higher MAE values but stays within range of the Lasso and LassoLars models compared to the Lars model, which does not.

As seen in the approximation results, the Lars model also struggles with optimization, often seemingly getting much higher fitness values than the other models, but with a larger error interval. It is likely that the Lars model appears to optimize better due to the larger MAE scores skewing the fitness results. As such, despite the high fitness values, the Lars model is the worst performing model. Since there is only one optimization type used, the optimization results are not necessarily indicative of which model optimizes the best. Rather, the optimization results demonstrate the error interval of the fitness values the models find and the amount of error between the models. For example, in Figure 56, the Lars model for both bases has a large error interval, and in the corresponding figure without Lars, Figure 57, the polynomial surrogate also shows a large error interval. This means that they are reaching inconsistent results, leading to a higher standard deviation. The difference between the best fitness reached for the models is the result of a combination of the models' error and their best fitness found during optimization. The difference between the basis is inconsistent depending on which optimization graphs are considered; on some occasions, all the multilinear basis surrogates reach higher fitness scores, and others the Walsh basis surrogates do. Again, this is more indicative of the error between the models rather than the actual performance between basis. However, based on the approximation results, it is safe to say the Walsh basis reaches more accurate fitness values than the multilinear basis as the Walsh surrogates tend to have lower MAE.

8 Conclusions and Further Research

The first Walsh-based surrogate was proposed by Sébastien Verel and coauthors, using the Lars surrogate [VDL+18]. The next proposed use of the Walsh surrogate was made by Florian Leprêtre and coauthors, using the Lasso surrogate [LVFM19]. Both articles demonstrated that the Walsh surrogate has the potential to outperform other SOTA models in PBO. This report evaluated both surrogates along with a polynomial and LassoLars surrogate to measure which performed best with the Walsh basis. From the results, Leprêtre and colleagues proposed Lasso surrogate was one of the best performing model, along with the LassoLars surrogate used in this report. The Lars surrogate, proposed by Verel and colleagues, was the weakest performing. This can be due to a variety of reasons, such as how using the Scikit-Learn default arguments for Lars may have led to worse convergence. However, Verel and coauthors did not propose alternative parameters to use with the Lars model.

In the future, exploring other parameters with the Lars model would be a good indicator if it's poor performance in this report is a result of the arguments used or the surrogate itself. Furthermore, comparing various optimization techniques with the different Walsh surrogates would be a good indicator of which optimization method works the best. After finding the best Walsh surrogate optimizer by comparing their performances, the next step would be to use the best performing optimized Walsh surrogate to compare with SOTA models such as the Gaussian process and Bayesian optimization of combinatorial structure. These next steps will be crucial in furthering the research for the use of Walsh surrogates' in PBO.

This report concludes that there is a clear improvement in the performance of the Walsh basis compared to the multilinear basis in terms of MAE. From the review of the results, the Lasso and LassoLars surrogates are the best performing with the Walsh basis for approximating pseudo-Boolean functions. Between Lasso and LassoLars, the performance difference appears to depend on the benchmark used rather than the values of n and k. Based on the performance of LassoLars on the OneMax with w-model epistasis, one conclusion that can be drawn is that the LassoLars model is less susceptible to outliers in the data compared to the Lasso model. However, when viewing the results from the NKL benchmark, the Lasso model performs better, indicated that the Lasso surrogate may be better suited for modeling functions that are built from subfunctions, rather than modeling a function by reducing it to subfunctions.

Overall, this report reinforces the potential of Walsh surrogate models, in particular the Lasso and LassoLars surrogates, as competitive models for PBO, and provides a foundation for further research on the use of the Walsh basis in surrogate models.

References

[BH02] Endre Boros and Peter L Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225, 2002.

[BP18] Ricardo Baptista and Matthias Poloczek. Bayesian optimization of combinatorial structures. In *International conference on machine learning*, pages 462–471. PMLR, 2018.

[Csa18] Felipe A Csaszar. A note on how nk landscapes work. *Journal of Organization Design*, 7(1):15, 2018.

[CWS14] Francisco Chicano, Darrell Whitley, and Andrew Sutton. Efficient identification of improving moves in a ball for pseudo-boolean problems. *GECCO 2014 - Proceedings of the 2014 Genetic and Evolutionary Computation Conference*, 07 2014.

[DYH$^+$20] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with iohprofiler. *Applied Soft Computing*, 88:106027, 2020.

[EHJT04] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, April 2004.

[GDMB$^+$24] Miguel Gonzalez-Duque, Richard Michael, Simon Bartels, Yevgen Zainchkovskyy, Soren Hauberg, and Wouter Boomsma. A survey and benchmark of high-dimensional bayesian optimization of discrete sequences. *arXiv preprint arXiv:2406.04739*, 2024.

[Han09] Chris Hans. Bayesian lasso regression. *Biometrika*, 96(4):835–845, 2009.

[LP19] George Lindfield and John Penny. Chapter 8 - analyzing data using discrete transforms. In George Lindfield and John Penny, editors, *Numerical Methods (Fourth Edition)*, pages 383–431. Academic Press, fourth edition edition, 2019.

[LVFM19] Florian Leprêtre, Sébastien Verel, Cyril Fonlupt, and Virginie Marion. Walsh functions as surrogate model for pseudo-boolean optimization problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 303–311, 2019.

[Ost12] Eva Ostertagová. Modelling using polynomial regression. *Procedia engineering*, 48:500–506, 2012.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Say18] Khalid Sayood. Chapter 13 - transform coding. In Khalid Sayood, editor, *Introduction to Data Compression (Fifth Edition)*, The Morgan Kaufmann Series in Multimedia Information and Systems, pages 417–459. Morgan Kaufmann, fifth edition edition, 2018.

[Swi20] Kevin Swingler. Learning and searching pseudo-boolean surrogate functions from small samples. *Evolutionary computation*, 28(2):317–338, 2020.

[Syk93] Alan O Sykes. An introduction to regression analysis. (20), 1993.

[UGH09] M. GRAZIANO USAI, MIKE E. GODDARD, and BEN J. HAYES. Lasso with cross-validation for genomic selection. *Genetics Research*, 91(6):427–436, 2009.

[UML21] Imanol Unanue, María Merino, and Jose A. Lozano. A general framework based on walsh decomposition for combinatorial optimization problems. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 391–398, 2021.

[VDL$^+$18] Sébastien Verel, Bilel Derbel, Arnaud Liefooghe, Hernán Aguirre, and Kiyoshi Tanaka. A surrogate model based on walsh decomposition for pseudo-boolean functions. In Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 181–193, Cham, 2018. Springer International Publishing.

[Wal23] J. L. Walsh. A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45(1):5–24, 1923.

[Wei05] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[Zae18] Martin Zaefferer. Surrogate models for discrete optimization problems. 2018.

[ZSF+14] Martin Zaefferer, Jörg Stork, Martina Friese, Andreas Fischbach, Boris Naujoks, and Thomas Bartz-Beielstein. Efficient global optimization for combinatorial problems. In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pages 871–878, 2014.

[ZZ14] Hongyang Zhang and Ruben H Zamar. Least angle regression for model selection. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(2):116–123, 2014.

Appendix – Github Code

The code used in this report can be found at the following Github repository:
https://github.com/ellakennon/Thesis-Ella-Kennon.git