Integrated Machine Learning and Optimization in Python

Master of Science Project Report

2019

Kanishk Mair

Department of Chemical Engineering

Advisor: Prof. Nikolaos V. Sahinidis

Contents

List of Figures

List of Tables

Chapter 1

Abstract

In the real world, there are several optimization problems in engineering, whose input-output relationships are too complex and indistinct. These problems are usually bounded but the underlying function and constraints acting on them cannot be determined. Owing to computational expensiveness of these problems, the algorithm is designed to use minimum function evaluations. This work presents an approach to solving the black-box optimization problem using Response Surface Methodology (RSM) using ALAMO, an algebraic modeling framework for modeling precise but low complexity surrogate functions in order to optimize with least number of function evaluations. To make the algorithm adaptable for high dimensions of feature space, the RSM is applied in multi-coordinate search manner. The uniqueness of the approach lies in applying the Machine Learning concept of Nesterov Momentum that boosts the convergence of the problems. This algorithm's convergence was usually third or fourth highest amongst all the compared categories and the best for convex non-smooth problems. A case-study on a hidden constrained problem showed convergence to the local optima.

Keywords: Black-box optimization, Derivative free, Global optimization, Accelerated gradient, Non-smooth, Non-convex

Chapter 2

Introduction

Black–box optimization is one of the most common and important problems in the engineering industry that are expensive to evaluate. The term "black–box" arises from the fact that while we can have input output pairs of x and $f(x)$ there is no other information about $f(\cdot)$, such as the Hessian matrix or the constraints acting within the model. These functions usually have a limited budget for the number of evaluations allowed [7]. These problems are generally NP-hard and hence become difficult to solve. The unavailability of this information means that we have to carry out Derivative-free optimization (DFO). The two main approaches to solve DFO problems are direct or model-based. In direct methods, the algorithm finds the direction to reach the optimal value by using only the function values. Pattern search, Nelder-Mead simplex algorithm, and adaptive search are few of the methods using direct search. On the other hand, model-based algorithms fit the black-box functions evaluated with a surrogated model that guide the model towards the optimal solution. The model-based approach are designed to work locally, like in implicit filtering and trust-region methods or globally, like in Response surface models, Branch and Bound search, Lipschitzian based partitioning techniques. In recent years, Evolutionary methods [15] are gaining prominence owing to their ability to fit on increasingly complex functions by using mutation of the points. The adaption of better individual points to the environment is obtained by assigning higher probability to fitter individuals.

In this work, we write the optimization algorithm in a block coordinate descent (BCD) approach (i.e. optimize over a coordinate hyperplane) using a surrogate model. In order to find out the global solution, the BCD search using three different methods to fit create the surrogate model was examined.

The first model optimizes the response surface model created by the radial basis functions (RBF) [5]. The response surfaces are interpolated using low order polynomials for the RBF.

The second option used to fit the model was ALAMO (Automated Learning of Algebraic Models for Optimization), which is an adaptive-sampling based algebraic modeling framework built specifically for black-box modeling. It utilizes derivative-free optimization techniques for minimizing various error criteria to come up with the simplest model.

Lastly, in order to create a simplistic model to fit the sampled points, a simple multilayer perceptron (MLP) was used to fit a model. Since MLPs act as universal function approximators [3], they can be used to fit the objective with high precision. [10]

The subsequent sections describe these models and a case study of a problem on a chemical engineering system modeled on ASPEN HYSYS.

Chapter 3

Proposed Algorithm

This section deals with the application of BCD search for the converging to the solution. There are numerous methods available for finding the global optimum. [1] Metaheuristic methods like genetic algorithm, simulated annealing are gaining popularity, however, their convergence rate is very slow. Deterministic methods like Branch-and-Fit use Lipschitzian-based partitioning techniques but the problem of finding the Lipschitz constant is sometimes as hard as solving the problem itself. [13] Although there are methods to approximate the Lipschitz constant, a big disadvantage of Lipschitz optimization methods is the need for plenty of function evaluations to get a good estimate of the optimal point. On the other hand, using surrogate models is increasing in popularity as the surrogate modeling capabilities for the objective and constraint functions that can be used for optimization have immensely improved. Moreover, computational time is saved by parallelizing the runs used to fit the surfaces. [2] Another advantage of the approach is that statistical analyses and experimental design can be done to identify which input variables are the most important by finding the variable contributing the highest variance to the output.

3.1 Sampling methods

With function evaluations being computationally expensive, it is essential for the algorithm to select adaptively the points to evaluate over for building the surrogate model. During initial investigation of the modeling, it was observed that the surrogate model built by low-discrepancy sequences or the quasi-random sequences was able to generalize to the model much better than the points sampled randomly. The quasi-random sequences are able to cover larger domain space with the same number of evaluation points since the points sampled are not completely independent of the previously sampled points. The sampling techniques used are discussed below.

3.1.1 Latin hypercube sampling

Widely used in the Monte Carlo simulation, Latin hypercube sampling divides the search domain into N intervals of equal probability. This sampling then selects one sample from each of these intervals.

3.1.2 Sobol sampling

In 1967, Ilya Sobol introduced the Sobol sequencing, which tries to sub-divide the M-dimensional space into 2^M points. We get data points x as:

$$x \in \left\{ \frac{k}{2^M}, k \in [0, 1, ..2^M - 1] \right\}$$

In order to have each of the points to have low discrepancy, some Generative matrices are selected which would mimic the points sampled in a dimension. For instance, for M = 2 dimensions and N = 4, we can have

$$C_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, etc.$$

Using these generative matrices, points for each of the dimensions are selected as:

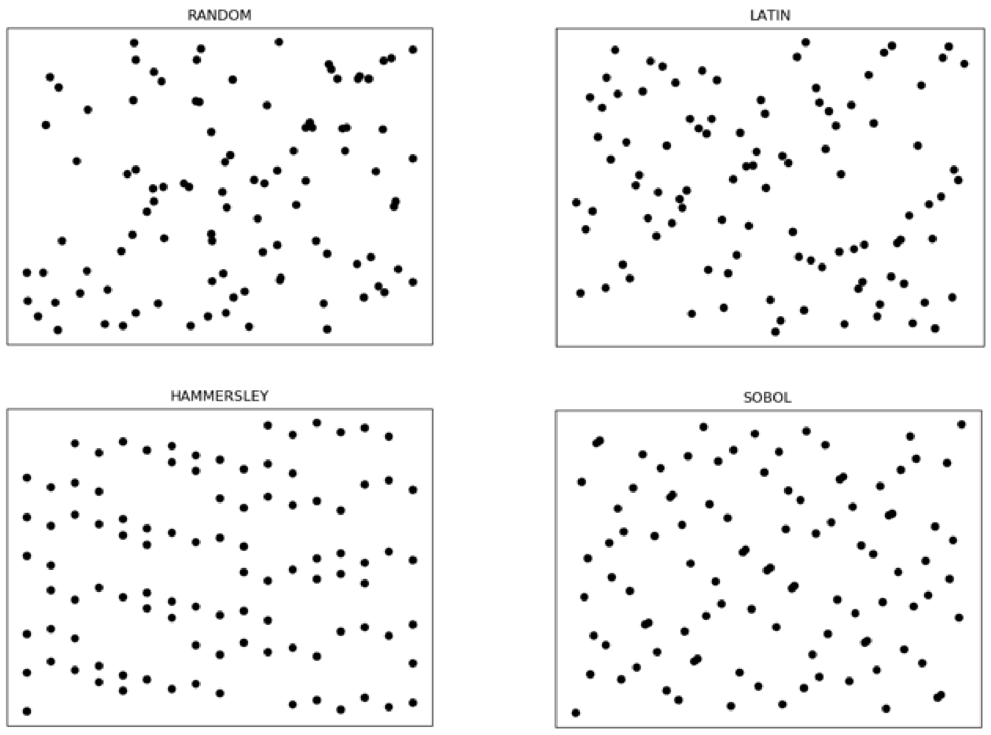


Figure 3.1: Sampling for 100 points carried out by (a) Random, (b) Latin Hypercube, (c) Hammersley, and (d) Sobol sampling in 2 dimensions.

$$x_n = \sum_{M-1}^{k=0} t_k 2^{-k-1}$$

where $t = \sum_i 2^i m_i$. An important property of Sobol sampling is that if there are more points than 2^N, then the x_n computed for the additional points would occupy the finer spaces of each axes.

3.1.3 Hammersley sampling

Hammersley sampling was introduced in 1975, which is an adaptation of the Halton sequence. We select a set of primes $p_j, j \in (1, ..N-1)$ For a given prime (p), if the van der Corput sequence is given by $g_p(n)$, then while the i^{th} sample point for Halton is $x_i = (g_{p_1}(i), g_{p_2}(i), .., g_{p_n}(i))$, for Hammersley, the i^{th} sample point is $x_i = (i/N, g_{p_1}(i), g_{p_2}(i), .., g_{p_{n-1}}(i))$ The Halton and Hammersley sampling is simple and efficient and both achieve asymptotically optimal discrepancy.

All the sampling methods are can be collectively compared as in Figure 3.1. In general, quasi-random sampling techniques are able to sample points from a larger area for the same number of evaluations. For test problems, it was observed that convergence of the model required about half the number of function calls compared to random sampling. Since Sobol sequences can converge faster than other simpler sampling techniques, for all the calculations in the report, Sobol sampling was used unless other method was mentioned.

3.2 Surrogate model function

After sampling the points, a surrogate function is fit on the points. There are plenty of options to build the model. The basis of selecting the right modeling technique is that the function must neither be too simple that it might require several iterations nor be too complex to generalize and be unable to provide the right point for next iteration. To develop good approximation models, response surface methodology will be designed for the system being analyzed. In response surface methodology, scattered data is modeled assuming it will possesses some local/global optima. In general, we assume that we are analyzing the following complex system:

$$y = F(x), \text{ where } x = [x_1, x_2, .., x_m]^T$$

For analysis, we have N sampled points x_i with corresponding output values y_i, i = 1,2,. . . , N. While optimizing using linear or quadratic least square approach provides useful information about the model, it fails to locate multiple minimas or maximas. For this reason, the modeling carried out is not on very simple algebraic functions. Three methods used in this work are discussed below.

3.2.1 Radial Basis Function

The Radial Basis Functions (RBF) uses a function $\phi(\cdot)$, whose value at any given point (x) depends upon distance from the known sampled points. The $\phi(\cdot)$ depends on the kernel selected and for the investigations in this work is set to a polynomial of third order [1]. The function value F(x) is found as:

$$F(x) = \sum_{M-1}^{k=0} w_i \phi(r_i, c)$$

where w_i is coefficient for i^{th} point r_i is the distance of point x from the i^{th} sampled point. The linear model formation of F(x) allows the estimation of the optimal parameter value w_i using linear least squares. The RBF algorithm 2 is inspired from [7].

Algorithm 1 RBF algorithm

```
 1: procedure RBF MODELING($a, b$)
 2:     Set $C \leftarrow 0$                    ▷ Initialize the evaluation count and s to get
 3:     Sample $z_i, f(z_i)$ and get $z_{min}$
 4:     while $C <$ Max function evaluations do
 5:         Get $f_{min}$ and $f_{max}$ to set $f_p(z_i) \forall i$    ▷ $f_p(z)$ is $f(z)$ with penalty term
 6:         Create an rbf model $p(z)$ over these points
 7:         $V_R(x_j) = \frac{s_p(x_j) - s_{min}}{s_{max} - s_{min}}$    ▷ $s_p(z_i) = \sum \lambda_i \phi + p(z)$
 8:         $V_D(x_j) = \frac{\Delta_{max} - \Delta(x_j)}{\Delta_{max} - \Delta_{min}}$    ▷ $\Delta(x)$ is the distance of $x_j$ from closest $z_i$
    Get the scoring using both $V_R$ and $V_D$. Their weights are cyclically updated as:
    Initialize $W_R = 0$ and $W_D = 1$, iteratively decrease $W_D$ to 0 and reset it to initial
    values.
 9:         W$(x_j) = W_R V_R(x_j) + W_D V_D(x_j)$
10:         Select the candidate point with best score,
11:         Sample around it to get $z_i, f(z_i)$ and $z_{min}$
12:     end while
13:     return $z_{min}$
14: end procedure
```

This algorithm had to be used since the scoring criteria based on distance parameter V_D has to be used only in an RBF model.

3.2.2 Multi-layer Perceptron

In the past few years, neural network based models have become a useful tool for regression problems. A Multi-layer Perceptron (MLP) is called a universal function approximator as it can fit on any given input-output data pair. The main challenge with respect to applying this in black-box problem is in deciding the model architecture. The algorithm must be able to handle any new black-box function with any number of input features. In addition, the underlying model can have varying complexity. Moreover, the objective of the surrogate model building is to build a generalized model using the least number of sample points. Thus, building a custom architecture for each model is not feasible and hence we must build a model for each iteration of BCD with the limited number of samples. Using Tensorflow, an MLP was designed with single hidden layer on which an activation function of Rectified Linear Unit (ReLU) was applied. Since fitting this model to highly complex function might cause large error, the LOGCOSH was chosen as the loss function. Considering x as the error, if x is small, $log(cosh(x))$ is approximately equal to $x^2/2$ and if x is large then it is $abs(x) - log(2)$. This loss function was selected rather than mean squared error in order to avoid very high loss values that may cause high variation in training. The training of the function was carried out using ADAM optimizer owing to its ability to handle sparse gradients on noisy problems (8). To converge to a new point, the MLP iterates as

$$X_{i+1} = X_i - \nabla(X_i) * \lambda$$

For optimization, only four iterations were used in each instance to reach a sub-optimal solution. For this reason, the learning rate was kept higher ($\lambda = 0.25$) than usual of to reach the gradient solution quicker.

3.2.3 ALAMO (Automated Learning of Algebraic Models for Optimization)

The last method used for modeling is ALAMO, which is a modeling toolbox built specifically for black-box problems. The main challenge solved by ALAMO [9] is in determining simulation domain by carrying out adaptive sampling and fitting low complexity surrogate models. In order to obtain such low complexity models, ALAMO allows various basis functions as observed in Table below (make table of various options). Various combinations of simple basis functions are considered for generating algebraic models, which are then solved using an optimization framework.

The sampling ALAMO uses is an active learning approach for selecting the next set of points that can achieve best accuracy using minimum evaluated points. An error-maximization strategy is employed to ensure that the points having the largest deviation from the model are included in model building for subsequent iterations.

In order to obtain such low complexity models, ALAMO allows various basis functions that we would want the function to be modeled on. ALAMO can model functions with polynomial, logarithmic, exponential and trigonometric terms. Since this work iterates while converging to the solution and only a small set of points are used keeping the function evaluation budget in mind, the options were set to model a second order polynomial function.

The sampling ALAMO uses is an active learning approach for selecting the next set of points that can achieve best accuracy using minimum evaluated points. An error-maximization strategy is employed to ensure that the points having the largest deviation from the model are included in model building for subsequent iterations.

3.2.4 Selection of modeling method

In order to design the hyperparameters for the modeling algorithm, selection of one of the three methods discussed is vital. The modeling with the MLPs with variation in various hyperparameters was not able to provide good results since it cannot model with the limited number of evaluation points provided. When comparing the results found by modeling with the radial basis function and ALAMO, it was found that the former model's convergence stopped when the dimensions of the black-box function was more than 50. Hence, for rest of the work, the modeling was carried out using ALAMO.

3.3 Optimizers

The functions returned by ALAMO are such that they are linear combinations of differentiable functions (like trigonometric, exponential, or polynomial). The algorithm is designed such that the optimizer selected needed to solve bounded minimization problems. Since we are fitting polynomial models, the following optimizers can be used.

3.3.1 L-BFGS-B method

The BFGS method is a quasi-Newton method (methods not requiring Hessians), so the Hessian matrix is approximated by evaluating gradients. The BFGS has better performance compared to Newton's method for non-smooth optimization instances. In order to accommodate for bounded constraints, BFGS-B variant of the algorithm is used. In order to optimize for large number of variables, the limited memory version of it called the L-BFGS-B (Limited-memory BFGS bound-constrained) method is used.

3.3.2 SLSQP method

The SLSQP (Sequential Least Squares Quadratic Programming) is also a quasi-Newton method and it iterates in a Sequential Quadratic Programming (SQP) manner, In order to enforce constraints, the first and second order optimality conditions are applied.

3.3.3 TNC method

The TNC (Truncated Newton Constrained) algorithm uses gradients [4] like the Newton-Conjugate gradient method and not the Hessian matrix. It also allows each variable to be given upper and lower bounds. Being truncated, the inner solver has limited number of iterations. Thus, this optimizer requires that the inner algorithm has low condition number for the Preconditioned iterative method.

For the given problem set, all of the solvers perform equally well in helping the algorithm converge to the solution. However, since the variable size can be varying, L-BFGS-B is chosen as it is tweaked to work efficiently when there are large number of variables to be optimized.

3.4 Cyclic Coordinate Search

This section describes the methodology of how the algorithm works. For this algorithm, the sampling technique, number of starting points, and tolerance of improvement can be provided or the default values are taken. Since the optimization is carried out in a block coordinate manner, the number of features (m) and the points (N) to be sampled have to be specified.

```
1: procedure NEW POINT(oldData, p)
2:     Presently at $X_0, Y_0$, the points are sampled about m attributes of $X_0$   ▷ This function iterates until improved point is found or the patience factor is 0
3:     if oldData is None then
4:         Sample N points of $(x^{(m)}, y)$
5:     else
6:         Sample N more points of $(x^{(m)}, y)$ and merge this with the oldData
7:     end if
8:     Build a model $f^A(\cdot)$ using ALAMO
9:     Minimize $f^A(\cdot)$ to predict new $x^*$ and get $y^* = f(x^*)$   ▷ $f(\cdot)$ is the black-box function evaluation
10:    if $p > 0$ then   ▷ The patience factor (p) of recursion at this point
11:        Sample N points of $(x^{(m)}, y)$
12:        if $y_{min} < Y_0$ then   ▷ $y_{min} = min(y^*, y_i)\forall i$
13:            $Y_0 \leftarrow y_{min}$
14:            $X_0 \leftarrow x_{min}$   ▷ $y_{min} = f(x_{min})$
15:            Apply scaling factor across these m features
16:        else
17:            $p \leftarrow p - 1$
18:            Add new points to oldData and call the NEW POINT(oldData, p) function
19:        end if
20:    else
21:        Return with same point and iterate over other attributes
22:    end if
23:    return $X_0, Y_0$   ▷ Updated optimal point across m attributes
24: end procedure
```

The main algorithm uses the above procedure for optimizing along the selected coordinates. The work is inspired from the Boosting with Momentum concept utilized in [6]. In the coordinate search, each coordinate step is termed as an iteration and an iteration over all the variables is called an epoch. Following each epoch, the momentum in that epoch is added to boost the convergence. Additionally, the variables are sorted using the momentum values in order to make surrogate functions more representative, else the surrogate model built only has variables with higher momentum.

Various starting points are sampled to start iterating if the improvement in previous epoch was not significant, indicating a local or global minima point. In this work, a relative improvement of minimum 0.1% was needed; else the algorithm iterates over next starting point.

Algorithm 2 Main algorithm

```
1:  procedure BOOSTING WITH MOMENTUM
2:      Set $C \leftarrow 0$                                   ▷ Initialize the evaluation count
3:      Sample multiple starting points $(X_k^{SP}, Y_k^{SP})$ and sort based on output.
4:      while $C <$ Max function evaluations do
5:          while $E <$ MaxEpochs do
6:              Start this epoch from $(X^E, Y^E)$
7:              Set axis order using $\Delta X^{E-1}$ for epoch E
8:              for $i \in [1, B]$ do                          ▷ B is number of blocks
9:                  Add momentum along the axes selected in $i^{th}$ iteration
10:                 $X_i^E \leftarrow X_i^E + \beta(\Delta X_i^{E-1})$
11:                 $Y_i^E \leftarrow f(X_i^E)$
12:                 Optimize across this point using Surrogate model
13:                 $X_i^E, Y_i^E \leftarrow$ NEW POINT$([X_i^E, Y_i^E], 0)$
14:             end for
15:             Compute the momentum in this epoch
16:             $\Delta X^E \leftarrow X^E - X^{E-1}$
17:             $\Delta Y^E \leftarrow Y^E - Y^{E-1}$
18:             if $\Delta Y^E < j$ then                       ▷ Significant decrease in Y value
19:                 Continue to next epoch
20:             else                                           ▷ Point has not improved significantly
21:                 Start from next best starting point
22:             end if
23:         end while                                          ▷ Start iterating from next starting point
24:     end while
25: end procedure
```

Chapter 4

Results and Discussion

4.1 Problem set

The algorithm described receives output value by sending necessary input to the black-box functions. These are synthetic black-box functions whose global optimal solutions are known. These black-box functions vary in input parameters ranging from 1 to 300 variables and are classified in four categories based on whether they are convex or non-convex and whether these functions are smooth or non-smooth. The code for these functions are available as C files to be compiled at the Sahinidis Optimization Group website (http://archimedes.cheme.cmu.edu/?q=dfocomp).

The number of problems for the four categories described is shown in Table 4.1. The detailed overview of the problems evaluated based on number of variables given is seen in Figure 4.1.

Problem Category	**Number of problems**
Non-Convex Non-Smooth	17
Non-Convex Smooth	236
Convex Non-Smooth	150
Convex Smooth	72

Table 4.1: Problem set overview

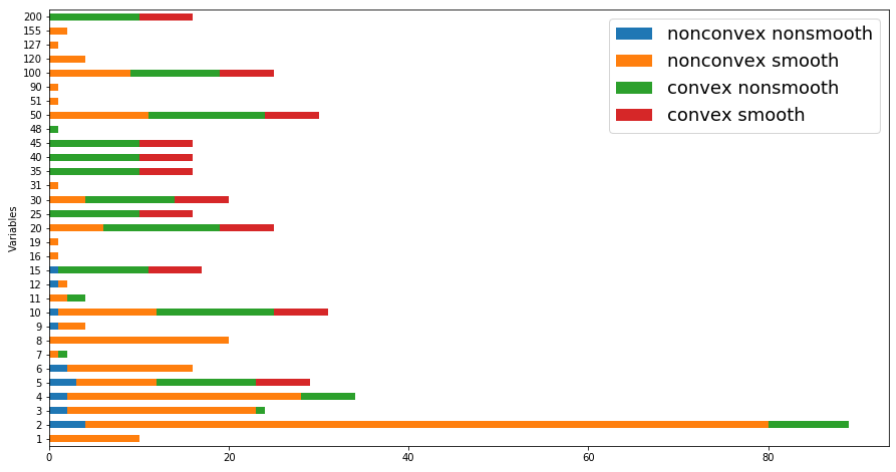


Figure 4.1: Number of test problems based on type and number of variables

4.2 Convergence criteria

In these problems, reaching the exact global optima would require infinite function evaluations for the points to converge. So, in order to examine whether a given problem converged or not, following criteria were setup.

4.2.1 Relative error

The output values were compared based on relative error and if the relative error (RE) was less than 0.5%, then the problem is deemed to have converged. If y* is the global optima and $\hat{y}$ is the present value, then the following expression would be true if the solution has converged.

$$RE(y^*, \hat{y}) = \frac{y^* - \hat{y}}{\hat{y}} \leq 0.5\%$$

4.2.2 Absolute error

For the given set of problems, about half of them had an optimal solution value of zero or values tending to be zero in which case the relative error becomes undefined. For those cases, the absolute error values was instead used. Most of these problems had very high values in the range of 10^{10} to 10^{30} when the solution was far from optimal. Thus, for convergence, the convergence criteria was satisfied if the absolute error was less than one.

4.2.3 Average Manhattan distance

For some instances, the reported optimal y* values are different from the $\hat{y}$ values even when the feature values are the same. Thus, instead of solely relying on the output values, proximity of the feature values with those of optimal solution was also analyzed. To normalize the distance value computed, the distance was normalized with the distance between the upper and lower bounds of that feature value, and the convergence of solution is checked by the below equation. For analysis, the Manhattan distance (MD) metric was selected since it is not biased towards the number of features. Using sum squared distances as the criteria here would make it less stringent on higher number of features.

$$MD(x) = \sum_{i=1}^{M} \frac{|x_i^* - \hat{x}_i|}{x^{(UB)_i - x^{(LB)_i}}} \leq 0.01\%$$

where M is the total number of features in x.

4.3 Experimental setup

The computation for all the analysis was carried out on a 64-bit Intel 2.70 GHz processor in a Windows machine. The code for the models was run using Python 3.7. The same set of problems were tested in [11] where there was a limit of 2,500 function evaluations in each run. Thus, same evaluation budget was reserved for analyzing the convergence.

Some of the problems defined have no bounds, but since the sampling stage of the algorithm relies on bounds, all the variables for these problems were given bounds in the interval [-10000, 10000]. In order to tally the results, the starting point specified for use in the problem data was selected. After selecting the parameters, each problem was run only once and its performance compared.

In order to assess the quality of the solutions obtained, the solutions returned by the algorithm against the globally optimal solution for each problem was analyzed as described in the previous section.

During the computations, the algorithm was able to receive only the output value based on the input feature values given. The algorithm itself calculated all other data, including the gradient of the response surface model built.

4.3.1 Hyperparameter settings

The algorithm has various hyperparameters whose values were tweaked initially to obtain the ones that gave best performance. The momentum factor (β) was set to 0.9 based on research in this domain. The patience factor determines the exploratory versus exploitative trade-off in the algorithm. In the present work, it was set to three occurrences for all problems. The sampling domain was attenuated or augmented for the features selected in the cyclic approach using the scaling factor (SF_i) for the specific iteration number i.

$$SF_i = SF_{i-1} \times \left(\frac{Z_i}{Z_{i-1}}\right)^p$$

where Z_i is the output value for the i^{th} iteration and p was a constant set to 0.5. If the output is negative, the constant p was multiplied by -1. For the first iteration, the scaling factor for all features is initialized to 1. In some cases, the values of output might change signs from negative to positive or vice-versa. In such cases, if the values improved, then the scaling factor was halved and otherwise it was doubled.

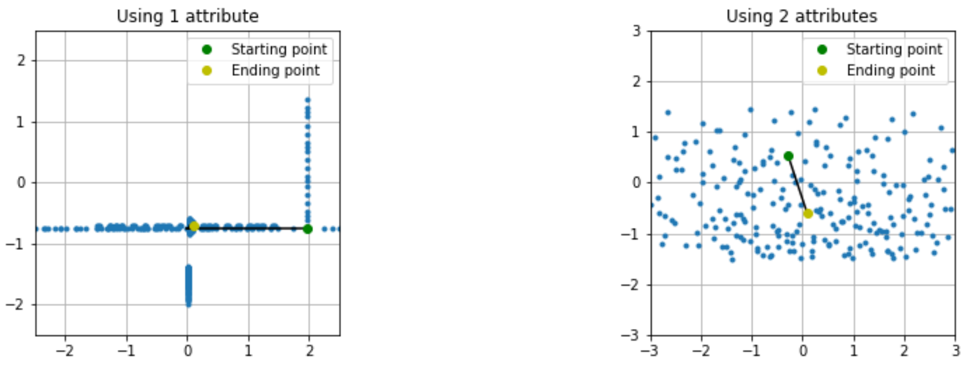
The most important hyperparameter to choose is the number of attributes to be modeled for the cyclic coordinate approach. This becomes an extremely challenging decision when the number of attributes increase over 30. The number of attributes selected for the present analysis is listed in the Table 4.2. Since the black-box function cannot predetermine the convexity and smoothness of the problem, these hyperparameter settings were applied to all kinds of problems.

4.4 Illustrative example: camel6

The camel6 problem, an abbreviation for the 'six-hump camel back' function as seen in the equation below, was used to demonstrate the convergence strategy employed by the

Total Features	Sampled points	Features selected
1-2	20	1
3-6	20	2
7-15	25	3
16-30	30	4
50-100	30	6
110-130	50	12

Table 4.2: Number of samples and features selected per iteration



(a) Search in 1 dimension (b) Search in 2 dimensions

Figure 4.2: Comparison of sampling while the points converge for the camel6 problem

algorithm.

$$f(x) = \left(4 - 2.1x_1^2 = \frac{x_1^4}{3}\right)x_1^2 + x_2x_1 + (4x_2^2 - 4)x_2^2$$

There are two global minima for this function with f(x*) = -1.0316, at x* = (0.0898, -0.7126) and (-0.0898, 0.7126) while the remaining four are local optima. Using the algorithm, the problem was solved with one and two attributes respectively as seen in Figure 4.2. When using one attribute, the problem starting from $x_0^{(1)}$ = (1.975301, -0.75768) converges to the point $\hat{x}^{(1)}$ = (0.095951, -0.71191) with $f(\hat{x}^{(1)})$ of -1.03147. Similarly, when using two attributes, the problem starting from $x_0^2 = (-0.2934, 0.5758)$ converges to the point $\hat{x}^{(2)} = (0.0640, 0.7246)$ with $f(\hat{x}^{(1)})$ of -1.03089. For both experiments, the convergence was achieved within the specified budget of 2500 function evaluations.

4.5 Black-Box Problems

This section compares the performance of this algorithm with those described in [11]. The strategy for boosting the algorithm's performance using momentum was derived from [6] where it was referred to as the BOOM algorithm and the same term is referred to in the plots here. The results for this are computed at convergence while for the rest of the algorithms, the fraction was recorded only after 100, 200, 500, 1000, 2000 and 2500 evaluations. In these plots, the horizontal axis shows the progress of the algorithm as the number of function evaluations gradually reached 2,500.

4.5.1 Computational results for convex problems

Figure 4.3 shows the fraction of convex smooth problems solved by each solver within the optimal tolerance specified. This algorithm solves about 45% of the problems. The TOMLAB/GLCCLUSTER is the best solver in this category followed by MCS, solving about 75-80% of the problems.

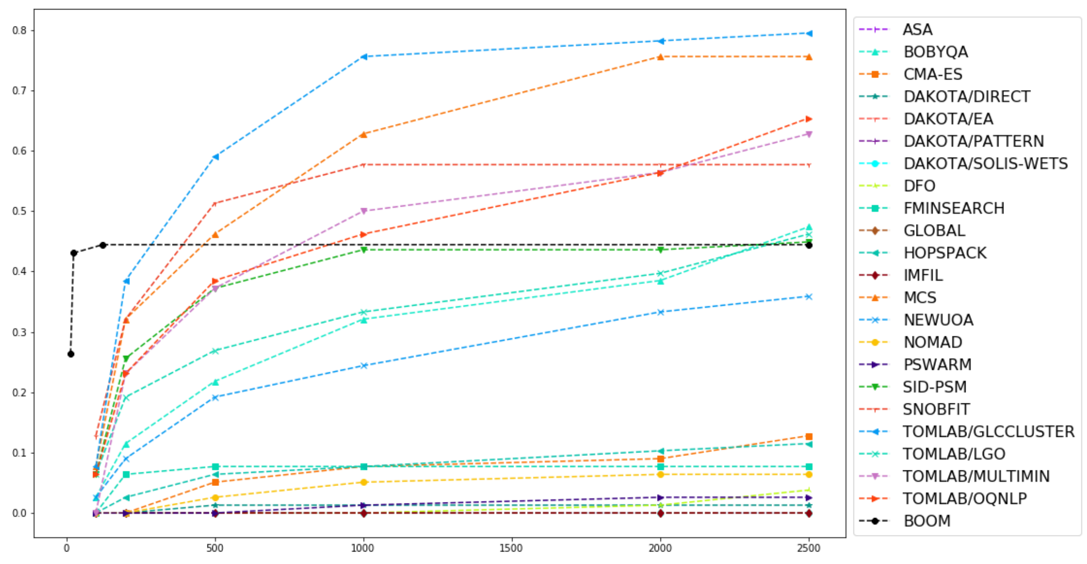


Figure 4.3: Comparison of fraction of convex smooth problems solved as a function of number of function evaluations

Figure 4.4 shows the fraction of convex non-smooth problems solved. This algorithm solves close to 52% of the problems while the solvers TOMLAB/GLCCLUSTER and TOMLAB/MULTIMIN follow closely, solving about 43% of the problems.

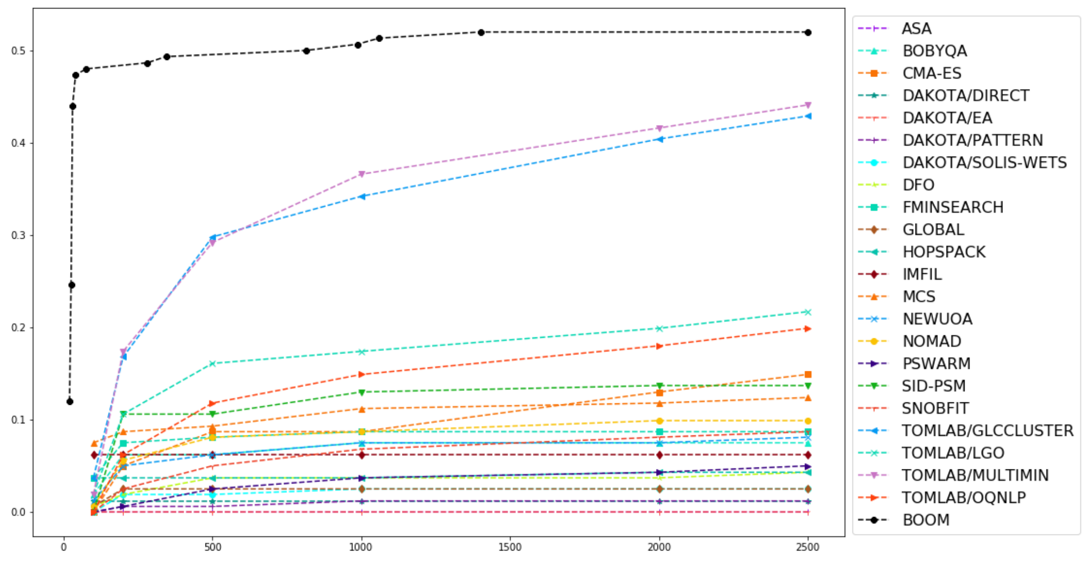


Figure 4.4: Comparison of fraction of convex non-smooth problems solved as a function of number of function evaluations

4.5.2 Computational results for non-convex problems

Figure 4.5 displays the solution progress for non-convex smooth problems. The present algorithm is able to solve about 67% of the problems. The TOMLAB/MULTIMIN and TOMLAB/GLCCLUSTER are the only solvers to solve more than 70% of the problems.

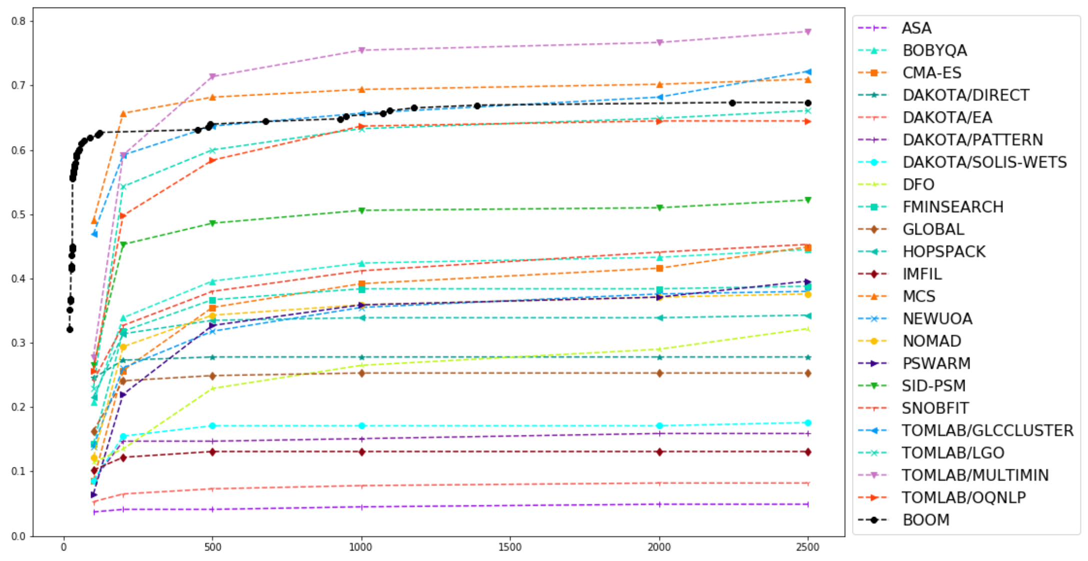


Figure 4.5: Comparison of fraction of non-convex smooth problems solved as a function of number of function evaluations

Figure 4.6 presents the fraction of non-convex non-smooth test problems. The present algorithm solves about 35% of the problems. The TOMLAB/MULTIMIN solves 45% of

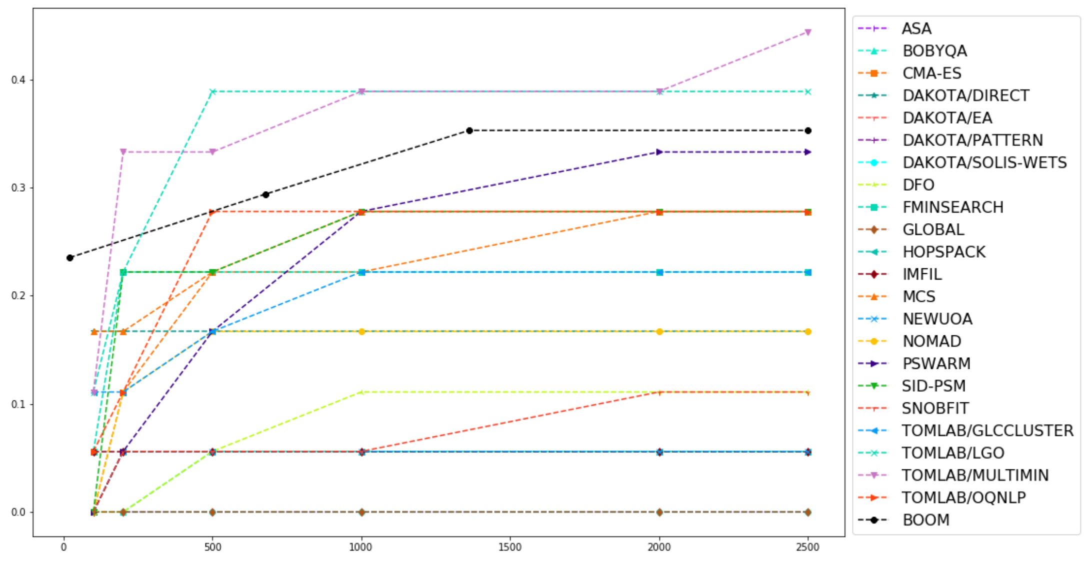


Figure 4.6: Comparison of fraction of non-convex non-smooth problems solved as a function of number of function evaluations

the problems followed by TOMLAB/LGO solving 39% of the problems.

4.5.3 Comparison of results

In this analysis, it must be noted that the present algorithm was solved on 475 problems while rest of the solvers were used to solve all the 500 problems. From the plots, it can be observed that the algorithm converges for many problems initially. This is owing to the quasi-random sampling employed in finding a good starting point. Following this, the points quickly converge to the globally optimal solution using the block coordinate descent approach. Since many of the problems are in low dimensions, a high rate of convergence is attained early on.

However, finding a good initial point in higher dimensions is difficult. This reflects the fact that the algorithm must have the potential to improve and reach the global optima once an initial point is chosen. Recent studies using the ADAM optimizer have shown that using not only the first but also the second moment of the gradient are much more successful in reaching global optima. This is because the additional term helps in escaping the local optima whereas the implementation of the Nesterov Accelerated Gradient (NAG) is susceptible to getting stuck in local optima. In most cases, NAG may converge if fortuitously the momentum is in the direction of the global optima. Thus, calculation of Hessian matrix for the variables and incorporating it along with the first moment will help in increasing the convergence of the algorithm.

4.6 Hidden Constrained problem

As a special case, this algorithm was tested to solve a chemical process system which has a hidden constrained problem. The objective of this problem is to minimize the total energy consumption on a per day basis during LNG (Liquefied Natural Gas) Production. The problem was formulated on ASPEN HYSYS V10, which is a commercial process simulator for the chemical engineering industry. This simulator acts as the Black-Box function, returning the objective value for 8 variables in this system; 4 variables are the pressures at different locations and the remaining 4 are the composition variables (the fifth composition variable is automatically set based on the remaining 4 compositions). All of these functions were linked to the algorithm written in Python, receiving an output value from the simulators.

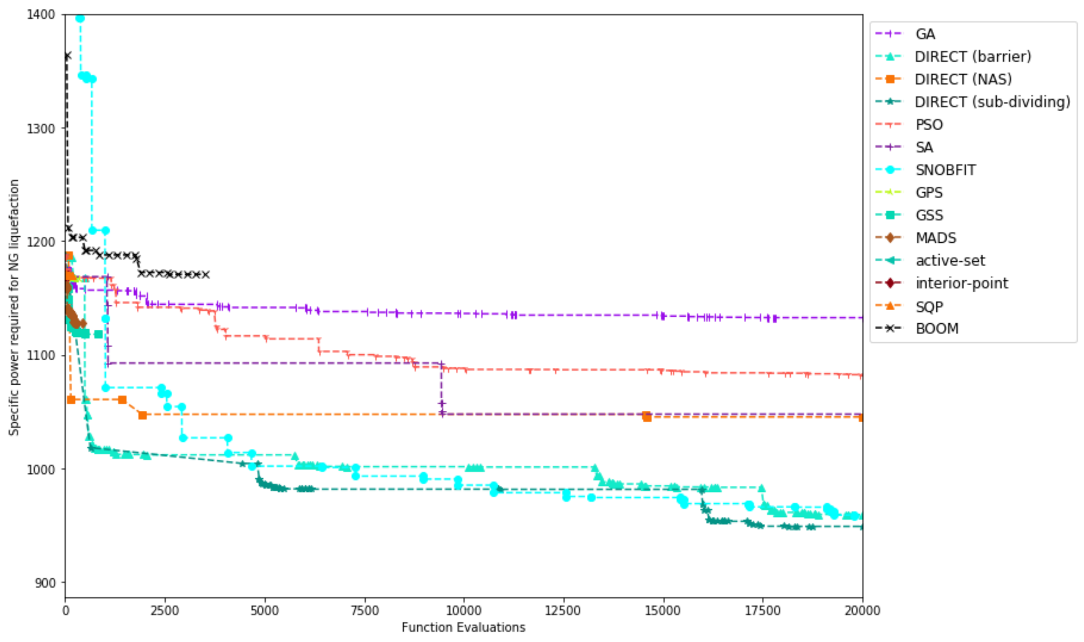


Figure 4.7: Comparative plot of various solvers on the ASPEN HYSYS problem

An analysis of this problem has been carried out in [8] which used a modified DIRECT algorithm that has been able to achieve the best possible solution till date. This paper contains the details of the problem, variables, and their lower and upper bounds. This problem was tested on a budget of 20,000 function evaluations, which has been the limit set by other algorithms solving this problem.

The plot of 4.7 shows that the present algorithm converges on a local optima of 1170.8 $kJ/kg-LNG$. The best solvers compared here obtaining solutions in the range of $950-970$ $kJ/kg-LNG$ were some modifications of the DIRECT (DIviding RECTangles) algorithm. Thus, the DIRECT algorithm is better at solving hidden constrained problems by dividing the search region into feasible spaces. In order to improve the solution in the present algorithm to move from local to global solution, modification to incorporate DIRECT algorithm can be carried out. Although DIRECT was not implemented in the present algorithm owing to the curse of dimensionality faced on using the DIRECT algorithm, future work on this algorithm can look into the adaptive block coordinate and DIRECT algorithm [14] to help the algorithm to escape the local optima.

Chapter 5

Conclusion

The optimization of Black-box class of functions is a crucial issue in the engineering world, however, the difficulty in solving these problems can be drawn from the fact that several commercial solvers cannot achieve the global optima within the evaluation budget. The algorithm discussed relies on quasi-random sampling to not only provide good initial point but also modeling simple and accurate Response Surface Models. Also, to build more representative models, the variables are sorted based on their gradients. For optimizing, it tries to improve upon the vanilla block coordinate descent approach by incorporation of the first moment term to apply the Nesterov Accelerated Gradient.

This algorithm's convergence rate is much better than several other algorithms, but algorithms like the TOMLAB/GLCCLUSTER are able to achieve higher accuracy except in convex non-smooth problems where the present algorithm has better accuracy. To make the algorithm more robust, the second moment term should also be included that is needed to implement the ADAM optimizer. The main objective would be to combine these two optimizers to implement the Nadam (Nesterov-accelerated Adaptive Moment Estimation) [12] which improves upon the convergence of both the algorithms. Another advantage of the algorithm is that it is designed to be easily parallelized that can reduce the evaluation time.

While solving the hidden constraint problem, the algorithm wasn't able to avoid the infeasible search domain; instead, it got stuck on a local minima. So, for any general black-box function, if the search domain can be restricted to feasible space which can have global optima, the convergence steps will decrease and it will increase the convergence. A comparison of various ways for handling these hidden constraints exceeds the scope of this work and must be looked at for future research.

Bibliography

[1] A. I. J. Forrester, A. Sóbester, and A. J. Keane. *Engineering Design via Surrogate Modelling*. Wiley, July 2008.

[2] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, Dec 2001.

[3] T. Kim and T. Adali. Approximation by fully complex MLP using elementary transcendental activation functions. In *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No.01TH8584)*. IEEE.

[4] K. Lindqvist, Z. Wilson, E. Næss, and N. Sahinidis. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984.

[5] D. B. McDonald, W. J. Grantham, W. L. Tabor, and M. J. Murphy. Global and local optimization using radial basis function response surface models. *Applied Mathematical Modelling*, 31(10):2095–2110, Oct. 2007.

[6] I. Mukherjee, K. Canini, R. Frongillo, and Y. Singer. Parallel boosting with momentum. In *Advanced Information Systems Engineering*, pages 17–32. Springer Berlin Heidelberg, 2013.

[7] J. Müller, C. A. Shoemaker, and R. Piché. SO-MI: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. *Computers & Operations Research*, 40(5):1383–1400, May 2013.

[8] J. Na, Y. Lim, and C. Han. A modified DIRECT algorithm for hidden constraints in an LNG process optimization. *Energy*, 126:488–500, May 2017.

[9] S. G. Nash. A machine learning approach to correlation development applied to fin-tube bundle heat exchangers. *Energies*, 11(12):3450, Dec. 2018.

[10] J. D. Olden and D. A. Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling*, 154(1-2):135–150, Aug. 2002.

[11] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, July 2012.

[12] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[13] R. G. Strongin. On the convergence of an algorithm for finding a global extremum. *Engineering Cybernetics*, 11:549–555, 1973.

[14] Q. Tao, X. Huang, S. Wang, and L. Li. Adaptive block coordinate direct algorithm. *Journal of Global Optimization*, 69(4):797–822, Dec 2017.

[15] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. IEEE, Dec. 2016.