Master Computer Science

Benchmarking for Efficient Global Optimization Algorithms on Mixed-Integer Problems

Name: Bartosz Piaskowski
Student ID: s2687194
Date: 06/08/2021
Specialisation: Data Science
1st supervisor: Bas van Stein
2nd supervisor: Anna Kononova

Master’s Thesis in Computer Science
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Efficient Global Optimization is considered an exceptional strategy for solving expensive black-box optimization problems. Most industrial and engineering optimization problems fall into this category, with a vast majority characterized by mixed-integer domains and non-differentiable, non-convex functions. This thesis compares two EGO-based algorithms designed to solve different real-world optimization tasks. Their performance is evaluated and analyzed on a mixed-integer industrial problem of medium-dimensionality as well as on a well-established black-box optimization benchmark extended with integer and categorical constrained variables. The obtained results explain some intricacies of individual approaches and their ability to tackle a wide spectrum of optimization tasks. The thesis is finalized with an exhaustive analysis of the findings, a brief discussion on optimization algorithms benchmarking, future work suggestions and ultimate conclusions.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>ii</td>
</tr>
<tr>
<td><strong>SECTION ONE: INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Examined solutions</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Problem statement</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Thesis structure</td>
<td>3</td>
</tr>
<tr>
<td><strong>SECTION TWO: RELATED WORK</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>4</td>
</tr>
<tr>
<td>2.2 EGO-based research</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Benchmarking suites</td>
<td>5</td>
</tr>
<tr>
<td><strong>SECTION THREE: EFFICIENT GLOBAL OPTIMIZATION</strong></td>
<td>6</td>
</tr>
<tr>
<td>3.1 Background</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Problem formulation</td>
<td>7</td>
</tr>
<tr>
<td>3.3 Surrogate Models</td>
<td>7</td>
</tr>
<tr>
<td>3.3.1 Gaussian Process</td>
<td>7</td>
</tr>
<tr>
<td>3.3.2 Random Forests</td>
<td>9</td>
</tr>
<tr>
<td>3.4 Acquisition Functions</td>
<td>10</td>
</tr>
<tr>
<td>3.4.1 Upper Confidence Bound</td>
<td>10</td>
</tr>
<tr>
<td>3.4.2 Probability of Improvement</td>
<td>11</td>
</tr>
<tr>
<td>3.4.3 Expected Improvement</td>
<td>11</td>
</tr>
<tr>
<td>3.4.4 Moment-Generating Function of Improvement</td>
<td>11</td>
</tr>
</tbody>
</table>
According to a well known empirical observation known as Moore’s Law, "The number of transistors in dense integrated circuits (IC) doubles about every two years" [1]. This phenomenon contributes to the ever increasing demand for innovation and progression in the field of semiconductor industry. What follows is a creation of highly sophisticated and advanced machinery that requires immense precision and calibration. In order for it to work efficiently, the use of a optimization algorithms is crucial and necessary. Industrial problems that require optimization are usually of complex nature. Apart from the two most common cases of continuous and discrete problems, many industrial problems consist of inputs constrained to be of integer type, which makes them fall into the category of mixed-integer programming problems (MIP) [2]. Such problems can be divided into linear and non-linear types, out of which the latter are the most frequently encountered while dealing with industrial optimization tasks. Mixed-integer non-linear programming (MINLP) problems can be portrayed as non-differentiable, black-box optimization functions of rather vague and unknown properties.

Over the last decades, many distinct algorithms were proven to be able to efficiently tackle the aforementioned problems. Some examples include: branch-and-bound algorithm [3], branch-and-cut [4], which were used to solve convex MINLP. In case of non-convex MINLPs, a modified version of spatial branch-and-bound algorithm [5] was able to find the global optima in a finite time.

1.1 Examined solutions

The two algorithms investigated in this thesis are Efficient Global Optimization (EGO) based methods, using surrogate models for approximation of an expensive black-box function, typical for engineering and industrial problems.
The first approach is an algorithm designed specifically to tackle a non-convex MINLP optimization problem of medium dimensionality with inner-dependencies between certain variables. This algorithm’s main characteristic was the implementation of a solution space complexity assessment method as well as a sub-solution space dynamic search that ensures convergence toward the global minimum in case of a detected complex environment.

The second algorithm, which served as an inspiration for the former solution, is an optimization package called MiP-EGO (Mixed integer, Parallel - Efficient Global Optimization) [6]. It was designed to solve a complex MINLP problem of neural network architecture design optimization and one of its features is parallelisation that enhances the velocity of the optimization process for expensive objective functions. Both approaches adopted Random Forests as their surrogate model, contrary to Gaussian Processes typical for EGO, which is necessary to handle the mixed-integer constraints of the optimization problem.

1.2 Problem statement

The main objective of this thesis is to investigate whether EGO algorithms designed with the intention to solve NP-hard non-convex MINLP real world industrial and engineering problems are generalizable and can be efficiently applied to a wider spectrum of optimization tasks of varying complexity.

1.3 Contributions

In order to solve that problem, an extensive experimentation pipeline has been designed and prosecuted. It involved testing and evaluating the performance of the given EGO algorithms on an industrial problem of moderate dimensionality as well as on a benchmarking platform for iterative optimization heuristics, IOHprofiler [7], which incorporates a well-established set of 24 black-box optimization problems from the Comparing Continuous Optimizers platform [8]. The benchmarking framework has been extended with highly customizable mixed-integer functions with integer-constrained variables as well as a composite function option, which also appends a categorical variable to the optimization problem.
1.4 Thesis structure

The remainder of the thesis will consist of a brief overview of related research, followed by a detailed description of the investigated EGO algorithms and then the benchmarking suite used to test them. Next, an outline of the two used experimental setups will be presented directly followed by the results, which will be thoroughly analyzed and discussed, comparing both investigated approaches in an informative manner. Finally, in the final chapter, the thesis will be summarized and concluded with future work suggestions.
SECTION TWO
RELATED WORK

2.1 Overview

In the past decade there have been many researchers aiming to create suitable solutions for the task of MINLP optimization as well as copious benchmarking platforms aiming to provide environments that allows for insightful performance-analysis and comparison of such methods. However, the quantity and diversity of available benchmarking frameworks makes it difficult to establish homogeneity in terms of optimization algorithm evaluation. The related research presented in this section were selected based on relevance to the two main component of this thesis, either EGO-related methodology or benchmarking of black-box optimization functions.

2.2 EGO-based research

In 2020, Bliek et al. investigated the problem of expensive black-box, mixed-integer function optimization and created a "Mixed-Variable ReLU-based Surrogate Modelling (MVRSM)" algorithm in order to solve it. It uses Bayesian Optimization (BO) (interchangeable with Efficient Global Optimization) with the surrogate model based on rectified linear units [9]. The algorithm has been tested on a couple of benchmarking functions (e.g. Ackley and Rosenbrock) with mixed-integer variables. The results show that MVRSM was able to outperform some other state of the art algorithms and efficiently deal with not only mixed-integer problems, but also ones with categorical and continuous variables.

Another publication from 2020 by Ru et al. describes a similar study of an EGO approach for problems with mixed categorical-continuous inputs [10]. The proposed "Continuous and Categorical Bayesian Optimisation (CoCaBO)" algorithm, which combines a novel Gaussian Process (GP)-based kernel with multi-armed bandits approach for handling categorical data, has proven to be successful at solving a handful of selected problems containing categorical inputs.
2.3 Benchmarking suites

The work of Tušar et al. resulted in a creation of new benchmarking suites for mixed-integer single- and bi-objective optimization [11]. The authors conducted research on the impact of discretization of continuous black-box functions on their complexity and properties. Methodology used in this publication served as an inspiration for the benchmarking suite extension implemented as a part of this thesis.

A mixed-integer benchmark called "The Mixed Integer Programming Library" (MIPLIB) has been created in 1992 for the purpose of evaluating optimization algorithms on real-world test problems [12]. Thanks to a collective work of many institutions it has been maintained and updated multiple times, with major releases in 2010 and 2017. The latest version of this benchmark consists of two benchmark sets with test problems of varying difficulty, from easy to not-yet solved. One of the biggest advantages of this benchmark is the diversity of available test problems that can be efficiently used to model real-world mixed-integer optimization tasks.
SECTION THREE
EFFICIENT GLOBAL OPTIMIZATION

This chapter contains the preliminary knowledge connected to the Efficient Global Optimization methodology as well as a brief overview of the two investigated algorithms. First, the background information will be presented, followed by the characteristic of each algorithmic approach.

3.1 Background

The core concept of Efficient Global Optimization, also called Bayesian Optimization (BO) is two-fold. It assumes the use of a surrogate model in order to generate a prior probability distribution that approximates and partially describes the otherwise unknown characteristics of a given optimization problem, it can also be called a probabilistic model. This approximation technique makes EGO a perfect candidate for handling expensive black-box functions.

The second core component is an acquisition function, which is a method of determining the desirability of a potential candidate solution based on the posterior distribution. Due to a large quantity of various surrogate models and acquisition functions, only a few chosen ones will be explained in detail in a following section.

Efficient Global Optimization is an algorithm that relies on a response surface methodology, as explained by Jones et al. [13]. Using only a few number of samples, the response surface is able to model the objective function, providing essential information about its behaviours and properties. The statistical model provides confidence intervals of the objective function values for unknown points, also called uncertainty. Thanks to modelling the correlation between points using a weighted distance formula, a stochastic process approach can be used instead of regression terms. It involves the correlation parameters, the mean and standard deviation, which are all used by the acquisition function to make valid predictions regarding the desirability of unknown points. The general version of an EGO algorithm has been summarized in Algorithm 1.
Algorithm 1 Efficient Global Optimization

**Require:** An acquisition function $\text{AF}$

Using the objective function $f$, generate the initial data set $X, Y$

Construct the initial surrogate model on $(X,Y)$

**while** stop criterion not fullfilled **do**

Maximize the acquisition-function:

$$x' = \arg \max_{x \in C} \text{AF}(x)$$

Evaluate obtained point: $y' \leftarrow f(x')$

Append $(x', y')$ to the current $(X, Y)$

Re-train the surrogate model of $f$ on the augmented data set $(X, Y)$

**end while**

3.2 Problem formulation

Let's consider a general minimization problem of finding the right set of parameters $\hat{x}$ that would return the global minimum of a given function $f(x)$ with variables constrained to be of a certain domain $X$ with finite lower- and upper- bounds. In this case the problem can be formulated as:

$$\hat{x} = \arg \min_{x \in X} [f(x)] \tag{3.1}$$

The function $f$ is in this case a black-box function for which the algorithm attempts to find a set of variables $\hat{x}$ that minimizes it, in as few function evaluations as possible (within a predetermined evaluation budget).

3.3 Surrogate Models

The first of the two components of Efficient Global Optimization is the probabilistic model, which is an approximation of the given function $f$ and its uncertainty.

3.3.1 Gaussian Process

The most standard surrogate modeling approach in BO is the use of Gaussian Processes (GP) for regression. The following formulations and preliminary knowledge is based on information from a book [14] as well as an online resource [15].
**Definition 3.3.1.** "A Gaussian Process is a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions."

A GP can be defined by the function mean \( m(x) \) vector and by a covariance function \( k(x,x') \) matrix. A function distributed with those parameters as a Gaussian Process takes a form:

\[
f(x) \sim GP(m(x), k(x,x'))
\]  

(3.2)

There is a certain distinction between a Gaussian process and a Gaussian distribution. The conversion from one to another has been demonstrated with the following example. In order to draw a sample from the distribution for a finite number of function indexes \( n \), we need to use its mean and covariance, defined respectively as:

\[
\mu_i = m(x_i) \quad \text{for} \quad i = 1, \ldots, n
\]

(3.3)

\[
\Sigma_{ij} = k(x_i,x_j) \quad \text{for} \quad i, j = 1, \ldots, n
\]

(3.4)

A random vector obtained from a Gaussian distributions would look like:

\[
f \sim \mathcal{N}(\mu, \Sigma)
\]

(3.5)

In order to model the function, GP takes advantage of the shared conditional distribution between the previous observations and the predicted samples. Consider a function \( f \) with a set \( (f) \) of \( n \) samples:

\[
f = [f(x_1), f(x_2), \ldots, f(x_n)]
\]

(3.6)
The function value of a prediction $x^*$ can be written down as $f^* = f(x^*)$ and is jointly normally distributed with the previously evaluated observations in the set $f$. This distribution can be expressed using a matrix transformation of covariances:

$$\Pr\left(\begin{pmatrix} f \\ f^* \end{pmatrix}\right) = \text{Norm}\left[0, \begin{bmatrix} K[X,X] & K[X,x^*] \\ K[x^*,X] & K[x^*,x^*] \end{bmatrix}\right]$$ (3.7)

where a $n \times n$ sized matrix $K[X,X]$ is derived from covariance function $k(x_i,x_j)$ with elements sorted by its i,j indexes, a $n \times 1$ matrix $K[X,x^*]$ obtained from $k(x_i,x^*)$, $K[x^*,X]$ and $K[x^*,x^*]$ analogously to the first two. The multivariate conditional distribution of those two sets can be derived from the following equation:

$$\Pr(f^* | f) = \text{Norm} \left[ \mu(x^*), \sigma^2(x^*) \right]$$ (3.8)

with the mean $\mu$ and variance $\sigma$ taking a form of:

$$\mu(x^*) = K[x^*,X]K[X,X]^{-1}f$$ (3.9)

$$\sigma^2(x^*) = K[x^*,x^*] - K[x^*,X]K[X,X]^{-1}K[X,x^*]$$ (3.10)

This relation gives access to the posterior distribution of the function at the predicted point $x^*$. The predicted mean and covariance from equation 3.9 and 3.10 are used by the acquisition function to propose next candidate solutions.

### 3.3.2 Random Forests

An alternative choice of the predictive model for EGO algorithms is the use of Random Forest (RF) regression method. RF is an ensemble learning approach, which uses a great number of decision trees (estimators) and aggregates the individual predictions into a final prediction by either averaging them (regression) or choosing the most frequent one (classification). The main advantage of using RF as a surrogate model is that it is remarkably computationally efficient and able to handle mixed-type variables. Similarly to Gaussian Processes, the Random Forests regression provides crucial parameters for an acquisition function, namely: *mean* of predictions as well
as their \textit{variance}, which is the uncertainty measure of the probabilistic model.

We can formulate the prediction of a RF model for regression using a following example taken from \cite{16}. Given a Random Forest model $h(\hat{x}; \theta_n)$ containing $n = 1, \ldots, N$ decision trees, a covariate parameters vector of length $p$ denoted as $\hat{x}$ linked with two random, equivalently and independently distributed vectors $X$ and $\theta_n$, the final prediction of such model can be expressed as:

$$h(\hat{x}) = \frac{1}{N} \sum_{n=1}^{N} h(\hat{x}; \theta_n) \quad (3.11)$$

Assuming the convergence of the RF model with $n \to \infty$ and no overfitting, we can denote the predicted error of an individual decision tree $h(X; \theta)$ as:

$$PE_n^* = E_{\theta E_{X,Y}}(Y - h(X; \theta))^2 \quad (3.12)$$

Estimating the uncertainty of a given Random Forest model can be done by calculating a Mean Squared Error (MSE) of the decision trees’ variance.

\subsection*{3.4 Acquisition Functions}

The second component of Efficient Global Optimization is called an \textit{acquisition function}, which is a method of estimating the desirability of a given prediction. Selecting the right function for a given optimization problem is a difficult task, partially because the chosen acquisition function is responsible for maintaining the balance between exploration and exploitation of the algorithm. In the following subsections, a few selected instances of acquisition functions will be described and formulated through the prism of the example from subsection 3.3.1.

\subsection*{3.4.1 Upper Confidence Bound}

One of the most popular acquisition functions is called \textit{Upper Confidence Bound}. This method can be described with a formula:

$$UCB(x^*) = \mu(x^*) + \beta \frac{\sigma(x^*)}{2} \quad (3.13)$$
which uses a parameter $\beta$ as a mean of balancing the relationship between the exploration and exploitation tendency of the function [17].

### 3.4.2 Probability of Improvement

Another common choice of an acquisition function is called *Probability of Improvement*. This non-parametric method relies on computing the probability of a given sample performing better compared to the current best known solution. This approach has been proposed in 1964, but still offers an efficient way of guiding sampling procedures for optimization purposes [18].

### 3.4.3 Expected Improvement

An alternative acquisition function, very similar to *Probability of Improvement* is called *Expected Improvement*. The main difference between them is that the former is guided mostly by exploitation and in a greedily way suggest samples that are of low risk and lower gain. The latter method is taking into account also the magnitude of a given improvement by calculating the "expected improvement", which for the previous example would be $f(x^*) - f(\hat{x})$. Both above mentioned approaches can be enhanced with a mechanism of controlling exploration and exploitation, by introducing a trade-off parameter $\xi$ [19].

### 3.4.4 Moment-Generating Function of Improvement

A novel acquisition function for Bayesian Optimization has been proposed in 2017 by Wang et al. [20]. This function is based on the moment-generating function of improvements. Some of its main strengths are that it "exploits the higher moments of improvement" by: combining the entire moment spectrum into one weighted distribution, is computationally efficient, and introduces a "temperature" parameter $t$ (continuous values), which is responsible for smoothly balancing the trade-off between the exploration and exploitation. This function has been tested and compared with Expected Improvement in some experiments conducted as a part of this thesis.
As mentioned in the Introduction, Subsection 1.1., there are two algorithms that were investigated for the purpose of this research. The first algorithm was designed to solve a specific industrial optimization problem. The second is an optimization package called MiP-EGO [6]. For the sake of simplicity, in the remainder of this paper, the first algorithm will be denoted as EGO-engine, while the second one will be denoted as MIPEGO. Prior to the presentation of these two algorithms, a brief overview of the industrial problem solved by EGO-engine will be shown in the following section.

4.1 Industrial Problem Specifications

This industrial problem can be classified as a non-convex, black-box optimization problem of medium-dimensionality (9 dimensions), with variables of various types (continuous, categorical and binary) with inner-dependencies between them. An important thing to notice here is that the objective function used by that algorithm has a form of a lookup table (LUT) with a full grid over all the parameters and their domains. Each of the 42 problems (LUTs) consisted of approximately 6400 samples and their corresponding fitness evaluations. An objective function of such form means that the originally continuous domain of certain variables of this industrial problem has been transformed into its discrete counterpart with a specific graininess. That also means that this previously MINLP optimization task has been converted into an Integer Programming (IP) one.

<table>
<thead>
<tr>
<th>Variable</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
<th>V7</th>
<th>V8</th>
<th>V9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>[a-b]</td>
<td>0/1</td>
<td>a/b/c</td>
<td>a/b/c</td>
<td>0/1</td>
<td>0/1</td>
<td>a/b</td>
<td>0/1</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Table 4.1
Solution space of the industrial problem. Variable V1 is continuous from a-b, discretized with a particular graininess, V2, V5, V6, V8, V9 are binary and finally V3, V4, V7 are categorical
4.2 EGO-engine

EGO-engine algorithm uses a Random Forest surrogate model as well as Expected Improvement acquisition function in order to approximate the objective function and generate new candidates. This choice of probabilistic model has been based on the fact of mixed-input type nature of the problem at hand. Some variants of acquisition function have been tested out, but for the sake of simplicity and its ability to balance exploration and exploitation, Expected Improvement was the function of choice for the algorithm.

Algorithm 2 EGO Engine

Sample initial observation set \((X, Y)\) using Latin Hypercube Sampling (LHS)
Train a random forest regression model \(RF\) on \((X, Y)\)

while True do
  Find the point \(P\) that corresponds to current minimum KPI
  if \(P\) keeps the same for \(N\) iterations then
    break
  end if
  Find the point \((x_t, y_t)\) by maximizing the EI function
  Evaluate the point \((x_t, y_t)\)
  Include the new point in observation set: \((X, Y) \leftarrow (X, Y) \cup (x_t, y_t)\)
  Re-train the RF with \((X, Y)\)
end while

if Solution space is not Complex then
  Return best solution in \(X\)
else
  Initialize the dynamic sub-solution search strategy
  Traverse the sub-solution space using a similar EGO-based mechanism (while loop) with heuristics
  Return best solution in \(X\)
end if

The algorithm starts with drawing equal interval samples of the \(V1\) and uses Latin hypercube sampling (LHS) of all the other variables, in order to effectively capture their overall distribution and accelerate the sampling procedure. The Random Forest model is fitted with the collected initial samples and by maximizing the EI function, new samples are drawn, which are appended to the collection of instances used for refitting the RF model. This procedure is carried on until a better solution has not been found for more than \(N\) iterations \((N=15)\). The author noticed that the uncertainty of the surrogate model does not allow the model to explore the solution.
space sufficiently and implemented a strategy of estimating the complexity of a given problem and dynamically searching through a sub-solution space. For this, a Pearson correlation coefficient is used to examine the linear dependencies between the neighbouring lines from the bottom of the solution space (obtained by permutation sampling of using the current best solution). In case they display linearity, the space is declared simple, otherwise it is distinguished as complex. The sub-solution dynamic search procedure is initialized in case of the latter complexity. Basically, it traverses the solution space starting from the lower bound of V1 and goes up with predefined intervals, using the mechanism displayed in Algorithm 2. This process is repeated a specific number of times. For proprietary reasons, no further details regarding this algorithm’s design can be revealed.

4.3 Mixed integer, Parallel - Efficient Global Optimization (MIPEGO)

The second investigated algorithm called "Mixed integer, Parallel - Efficient Global Optimization" (MIPEGO) in a mixed-integer optimization approach proposed by van Stein et al. [6], which incorporates the methodology of Efficient Global Optimization and features parallelisation, hence the name.

MIPEGO uses a Random Forest surrogate model for the same reason EGO-engine does, the mixed-integer domain of the optimization problem it was designed to solve. The original algorithm’s design incorporates a novel acquisition function called "Moment-Generating Function of Improvement" (MGFI), proposed by Wang et al. [20]. This infill-criterion introduces a strategy of balancing the relationship between exploitation and exploration by using a "temperature" parameter $t$. This parameter enables control over the degree to which proposed solutions associated with higher uncertainty (high risk / high gain) are favored versus the ones that exploit current predicted information. In this thesis, Expected Improvement was set as the acquisition function of choice after initial tests showed its slight advantage over the original option. It is possible that MGFI infill criterion could yield better performance after careful selection of the temperature parameter, but for the industrial problem explored in this paper it has been opted out from. MIPEGO also takes advantage of a Mixed Integer Evolution Strategy (MIES) by Li et al.[21] used for the task of acquisition function maximization. The algorithm used in this thesis has been summarized in
the corresponding pseudocode - Algorithm 3. After initial LHS sampling and fitting
the meta-model, it sequentially generates new candidates and updates the model until
the evaluation budget is exhausted.

Algorithm 3 MIPEGO

Generate the initial samples using LHS
Construct the initial random forest on \((X,Y)\)

while stop criterion not fulfilled do

Maximize the infill-criterion using Mixed integer Evolution Strategy:

\[
x' = \arg \max_{x \in C} EI(x)
\]

Evaluate obtained point: \(y' \leftarrow f(x')\)
Append \((x', y')\) to the current \((X, Y)\)
Re-train the random forest model of \(f\) on the augmented data set \((X, Y)\)

end while

The original version of MIPEGO described in the paper [6], using Random
Forest and MGFI, allows for parallel executions. It can propose several candidates
in a single iteration, using a certain number of acquisition functions, each initialized
with a different \(t\) "temperature" hyperparameter, sampled from a log-normal distribu-
tion. This allows the algorithm to utilize multiple GPUs and accelerate the process
of finding the optimal solution. Although parallelism might have been beneficial, it
was not used for the experiments conducted in this work, due to the properties of
investigates problems (e.g. inexpensive objective function).
In an attempt to investigate the degree to which EGO-based algorithms be
generalized, the two approaches described in Chapter 4 have been tested using an "It-
erative Optimization Heuristics" (IOH) platform introduced in a publication by Do-
err et al. [7]. Integrated in this platform is also a benchmarking suite of black-box
optimization problems from another platform called "COmparing Continuous Opti-
misers" (COCO) [8]. This "Black-Box Optimization Benchmarking" (BBOB) suite
contains 24 noiseless, continuous, black-box test functions that are commonly known
and used as performance test problems for optimization algorithms. They can be di-
vided into: separable, low-moderate conditioning, unimodal with high conditioning,
multi-modal with adequate global structure and multi-modal with weak global struc-
ture. The functions’ search space is from the range \([-5,5]^D\). Examples of some of
the function landscapes are presented in Figures 5.1 and 5.2.

Figure 5.1 Separable function example: Sphere (landscape), source: [22]
5.1 Extending the benchmarking test functions

The BBOB functions integrated into the IOH platform are currently available only in their original, continuous domain form. They can be instantiated by providing three parameters: function index (fid), dimension (dim), and instance index (iid). All of the 24 available functions (chosen fid) are parameterized and can be instantiated with a desired dimensionality (dim) and instance index (iid). The instances of a given function with a specific dimensionality are simply transformations applied to that test function, which creates any number of optimization problem of congenial complexity and properties.

In order to extend the IOH benchmark as well as conduct experiments on the behaviour of the EGO algorithms included in this thesis, the suite has been extended with functions with mixed-integer, and mixed-categorical variables.

5.1.1 Mixed-integer function

The first out of two extensions was creation of a mixed-integer function, whose some of the function variables are constrained to be of integer type. The design of a mixed-integer function class was driven mostly by the customizability and flexibility of the problem creation. This led to a decision of creating a new mixed-integer
function class inheriting all the properties of the original class, with the addition of a new input parameter for function initialization, called `int_mask`. This integer mask is an array of \([\text{dimension,}]\) shape, that decides on which particular variables should be discretized and into how many integer discrete points \((n)\). The discretization strategy is similar to the one proposed by [11]. The effective solution space \([-5,5]\) is divided into \(n\) (number of integer values of a given variable) equidistant spaces and their corresponding representatives from the continuous domain. This form of mapping is stored in a dictionary that is used in the call method of the function that returns the fitness values of a mixed-integer input. The number of such mappings depends on the integer mask provided by the user and specified \(n\) parameters. In order to demonstrate this procedure, consider an example of instantiating a function with a given integer mask. In order to create an arbitrary function of 5 dimensions with the first two variables of integer type with \(n = 40\) and \(n = 50\) integer values, we use this:

\[
\text{f} = \text{IOH\_mixint\_function}(\text{fid} = 21, \text{dim} = 5, \text{iid} = 1, \text{int\_mask} = [40, 50, 0, 0, 0])
\]

The number 0 in the integer mask indicates variables that remain continuous. This form of function initialization allows for a fully customizable declaration of the desired mixed-integer function, where the user can directly specify which variables and with what graininess should be discretized. The surface and contour plot of the function created using the code from the example above has been shown in Figures 5.3 and 5.4.

5.1.2 Composite function

The second contribution towards expanding the functionality of the IOH benchmarking platform was an addition of categorical constrained variables to the mixed-integer function, which resulted in the creation of a composite function. The main focus when creating this function class was the ability to customize it and directly control the properties of the categorical variable. The final design choice ended up with the introduction of an extension of the mixed-integer function class with three categorical modes and two corresponding input parameters controlling their behaviour. The overview of the new input parameters and their domain is shown in Table 5.1.
Figure 5.3 Surface plot of the example mixed-integer function. Projection of two variables x1 (integer) and x5 (continuous)

Figure 5.4 Contour plot of the example mixed-integer function. Projection of two variables x1 (integer) and x5 (continuous)
In order to create a composite function with categorical variable, the user is required to choose one of the available modes and also provide a corresponding array of either instance ids or function ids. The function will always consist of one categorical variable, replacing the last variable of the function, with inputs in a form of integers from range \([1, c]\), where \(c\) is calculated based on the possible combinations of the provided \(\text{"cat_fid"}\) and \(\text{"cat_iid"}\) arrays. Depending on the chosen mode, this categorical variable is going to do the following:

- **\(\text{cat mode = "fid"}\)** - (requires an array of chosen function ids \(\text{cat_fid}\)) - creates a list of underlying mix-int functions of provided function ids and depending on the parsed categorical input, return a corresponding objective function value of all the other variables.

- **\(\text{cat mode = "iid"}\)** - (requires \(\text{cat_iid}\)) - creates a list of underlying mix-int functions of provided instance ids and works analogously to the previous mode.

- **\(\text{cat mode = "fid_iid"}\)** - (requires both \(\text{cat_fid}\) and \(\text{cat_iid}\)) - creates a list of underlying mix-int functions with all the possible combinations of the provided function and instance indexes.

The standard function parameters \(\text{fid}\) and \(\text{iid}\) are overwritten depending on the chosen mode and provided arrays. They only serve the purpose of initializing the base of the test problem. In the example below, a composite function with \(\text{fid} = 1\), \(\text{dim} = 5\), \(\text{iid} = 1\) (overwritten) and \(\text{int\_mask} = \{40, 50, 0, 0, 0\}\) has been shown. The first two variables are of integer type and the last one is a categorical one with \([1,5,12]\) domain. The surface and contour plot of the composite function created using the code from the example below has been shown in Figures 5.5 and 5.6.

\[
f = \text{IOH_composite_function}(\text{fid} = 21, \text{dim} = 5, \text{iid} = 1, \text{int\_mask} = \{40, 50, 0, 0, 0\}, \text{cat\_mode} = \text{"iid"}, \text{cat\_iid} = \{1,3,5,7,12\})
\]
This implementation of a categorical variable allows for a full customization of the test function with the underlying objective functions of chosen characteristic appended in an attribute cat_list. The global minimum of the created composite function is the minimum of the hidden functions from that list.

Both created IOH functions (mix-int and composite) are equipped with methods that return their function fitness (y) targets (global minimum) as well as an approximated function input (x) targets. They also contain updated lower and upper bounds attributes and are compatible with the IOHlogger (logging method), that can be used in the IOHanalyzer (analysis tool of IOH platform) to plot the fixed-budget and fixed-target performance analysis.

Figure 5.5 Surface plot of the example composite function. Projection of two variables: x1 (integer) and x5 (categorical)
Figure 5.6 Contour plot of the example composite function. Projection of two variables: $x_1$ (integer) and $x_5$ (categorical)
SECTION SIX
EXPERIMENTS AND RESULTS

The experiments section is divided into two subsections. The first one will compare the two EGO-based algorithms based on their performance on the industrial problem mentioned in Chapter 4, Section 4.1. The second part will include an optimization performance analysis conducted on a proposed benchmark configuration.

6.1 Industrial Problem - Experiments

The real-world problem described in Chapter 4, Section 4.1 consists of 42 LUT (use cases). They are divided into training (28 instances) and test sets (14 instances). This split into train and test sets is rather arbitrary and its only purpose is to monitor the algorithms behaviour on two different batches of industrial problems. Both EGO-engine and MIPEGO algorithms have been tested on each problem instance by running them 10 times with 10 random seeds. The total number of runs is equal to 840 (2*420).

<table>
<thead>
<tr>
<th>Data set</th>
<th>Use cases</th>
<th>Runs (seeds)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>28</td>
<td>10</td>
<td>280</td>
</tr>
<tr>
<td>Test set</td>
<td>14</td>
<td>10</td>
<td>140</td>
</tr>
<tr>
<td>Overall</td>
<td>42</td>
<td>10</td>
<td>420</td>
</tr>
</tbody>
</table>

Table 6.1
A summary of the industrial problem dataset - each algorithm have been run on each use case 10 times with 10 random seeds.

<table>
<thead>
<tr>
<th>Setting</th>
<th>MIPEGO</th>
<th>EGO-engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surrogate Model</td>
<td>Random Forest</td>
<td>Random Forest</td>
</tr>
<tr>
<td>Acquisition Function</td>
<td>Expected Improvement</td>
<td>Expected Improvement</td>
</tr>
<tr>
<td>RF estimators</td>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>Evaluation budget</td>
<td>150</td>
<td>custom stop criterion</td>
</tr>
<tr>
<td>MIES budget</td>
<td>150</td>
<td>*</td>
</tr>
<tr>
<td>Initial Samples</td>
<td>10</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 6.2
Input settings of each algorithm for the performance analysis on the industrial problem.
6.2 Industrial Problem - Results

The two performance analysis metrics used in this thesis are:

- Number of Evaluations - number of fitness evaluations of the objective function, the more the worse the performance.

- Solution Metric (see equation below) - 0 means finding the global minimum, the higher the metric, the worse the performance.

\[
\text{metric} = \frac{\text{found solution fitness} - \text{global minimum}}{\text{global minimum}}
\] (6.1)

For each algorithm, there are two versions (original and modified). The original versions are the one with default input parameters (shown in Table 6.2) and no changes in the source code. The modification for EGO-engine were related to minor changes in the process of drawing initial samples. The algorithm was omitting a small region of the solution space and the modified version of the code fixes this small flaw. In case of MIPEGO, the modified version means the algorithm enhanced with an heuristic reducing the dimensionality of the solution space based on the inner-dependencies between some nested variables, similarly to EGO-engine.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>#Evaluations</th>
<th>#Solution metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>max</td>
</tr>
<tr>
<td>EGO-engine (original)</td>
<td>97.9</td>
<td>175</td>
</tr>
<tr>
<td>EGO-engine (modified)</td>
<td>100.6</td>
<td>181</td>
</tr>
<tr>
<td>MIPEGO (original)</td>
<td>91.9</td>
<td>150</td>
</tr>
<tr>
<td>MIPEGO (modified)</td>
<td>81.1</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 6.3
Comparison of EGO-engine with MIPEGO on the TRAINING SET of the industrial problem (green shows best results).
The analysis of the results show that EGO-engine is favored in terms of the proposed solution metric and worse in terms of function evaluation. The advantage of EGO-engine in terms of finding the more accurate solutions is most likely due to the dynamic sub-solution space strategy, which was designed in order to bypass getting stuck in local minima and traverse the entire discretized solution space sequentially after distinguishing a complex relationship between variables. MIPEGO was not equipped with such strategies that are only applicable to finite solution spaces, and instead objectively looks for new solutions using the same EGO-based scheme. Discarding regions that supposedly do not include the global minimum is impossible when dealing with mixed-integer problems, which contains infinite spaces of continuous variables. Most likely because of the same dynamic search mechanism, EGO-engine uses on average more function evaluations than MIPEGO. The sequential traversal of the finite space of a particular variable indicates searching through regions that might have already been explored prior to initializing the search, thus a larger number of function evaluations required. This comparison is more of an approximation rather than an ideal, stern analysis, and that is because of the varying stopping criterion of the algorithms. Despite EGO-engine reaching its max evaluation (worst) cap at 181, which is relatively close to 150, it gives it an apprehend in the solution metric for the cases where the budget of 150 has been exceeded, even though it only occurred a handful of times.

Another observation regarding the comparison of original algorithm to their modified version. EGO-engine with the minor fix turned out to still perform slightly worse on the smaller test set. However, on the train set, it outperformed the original solution very decisively, indicating there were multiple instances of skipping a niche
were the global minimum was located. For MIPEGO, both versions of the algorithm obtained very similar accuracy results (metrics), but the version enhanced with heuristics consistently used a lesser number of function evaluations on average, which was expected given the reduced dimensionality of the search space.

6.3 Benchmark problem - Experiments

In order to properly evaluate the investigated algorithms, a new benchmark of in total 192 test function instances has been proposed. The benchmark design can be divided into mixed-integer and composite benchmark functions.

**Mixed-integer benchmark (120 functions in total):**

- Function ids: [1, 8, 12, 21]
- Instance ids: [1, 2, 3, 4, 5]
- Dimensions: [5, 10, 20]
- Discretization: [20%, 60%]

**Composite benchmark (72 function in total, 24 per categorical mode):**

- Categorical modes: ["fid", "iid", "fid_iid"]
- Function ids: [1, 8, 12, 21] (only for "iid" mode)
- Instance ids: [1, 2, 3, 4] (only for "fid" mode)
- Cat_iid array: [1, 2, 3, 4, 5]
- Cat_fid arrays: [[1, 2, 3, 4], [6, 7, 8, 9], [11, 12, 13, 14], [15, 16, 20, 21]]
- Dimensions: [5, 10, 20]
- Discretization: [20%, 60%]

The motivation for choosing such function for the benchmarking suite is that according to the information about the test function from COCO, their difficulty is supposedly increasing with the increasing function id number. Based on this fact, 4
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Functions</th>
<th>Runs (seeds)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed-integer benchmark</td>
<td>120</td>
<td>5</td>
<td>600</td>
</tr>
<tr>
<td>Composite benchmark</td>
<td>72</td>
<td>5</td>
<td>360</td>
</tr>
<tr>
<td>Overall</td>
<td>192</td>
<td>5</td>
<td>960</td>
</tr>
</tbody>
</table>

Table 6.5
A summary of the benchmarking suite - each algorithm have been run on each function 5 times with 5 random seeds.

function indexes corresponding to the following functions: [1: Sphere, 8: Rosenbrock-Original, 12: Bent Cigar and 21: Gallagher’s Gaussian 101-me Peaks] were chosen. The instance indexes of choice were [1,2,3,4,5] in order to preserve diversity of results for each specific test problem. Having instantiated all the necessary functions, the two EGO-based algorithms were tested on all of them, ran 5 times with 5 random seeds. The total number of optimization algorithm executions was 1920 (2 * 960).

The summary of the benchmark suite can be found in Table 6.5.

In order to carry out the tests on the benchmark suite presented above, the previously used EGO-engine had to be severely transformed. This is because it was designed to only work for the particular industrial problem, heavily exploiting the knowledge of the solution space. An important thing to notice is that for the exact same reason, the benchmark had to be transformed as well. For each test problem (function), a LUT with 5000 randomly sampled input settings has been generated, which served as a representation of the benchmark function, compatible with the EGO-engine algorithm. Although this form of a benchmark function does not reflect its actual properties, it was the only feasible method of evaluating the EGO-engine algorithm. First of the changes in the algorithm was replacing the LHS initial sampling method with a random sampling mechanism (20 initial samples). The complexity estimation method, which was only compatible with the industrial problem has been removed. Because of that, the dynamic sub-solution space strategy, that was capable of being preserved, instead was used regardless of the complexity of the solution space, and initialized in a similar way, namely after the current best solution has not been upgraded for a specific number of function evaluations. The algorithm’s stopping criterion has been changed so that they break the loop when a global minimum is found or the number of function evaluations reaches 150. In order to distinguish this algorithm from the original version, it will be denoted as EGO-engine (bare).
Fortunately, the MIPEGO algorithm has been designed in a way that it can be applied to any mixed-integer programming optimization problem. However, in order to compare it with the performance of EGO-engine (bare) in a meaningful way, it had to also use the generated LUT as its objective function instead of the test functions of the benchmark. In order to successfully do that, two options were considered. The first option involved creating a LUT by discretizing all the variables of each function and for each combination of such input settings (grid over all variables) evaluate it and append into the LUT. Unfortunately, this approach would require creating a LUT with 161051 instances in order to depict the simplest of the mixed-integer functions with merely 5 dimensions and discretization of the continuous space [-5, 5] into 11 finite equidistant counterparts. This approach was therefore infeasible, so instead the entire previously generated LUT (5000 random inputs) was used in order to create a Random Forest Regression model, that would approximate the objective function of this discretized space. All 5000 input parameters and their fitness values were used for training the model with 1500 decision trees. The trained RF model was then used by MIPEGO as the objective function of the given LUT. This kind of work-around procedure was only applied to the 5 dimensional part of the benchmarking suite, which in this case involved 40 mixed-integer functions and 24 composite functions. The version of MIPEGO which uses the approximation of a LUT as the objective function will be denoted as MIPEGO (LUT). The MIPEGO tested on the benchmark using the actual benchmark test functions as its objective function will be denoted MIPEGO (original).

6.4 Benchmark problem - Results

The performance analysis metrics used for these experiments were the same as the ones used for the comparison on the industrial problem data set (evaluation number and metric). The results will only be presented for a handful of test function and their parameters, including all of them would be impossible given the number of test problems included.
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>FID</th>
<th>DISC</th>
<th>#Evaluations</th>
<th>#Solution metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>MEAN</td>
<td>MAX</td>
</tr>
<tr>
<td>EGO-engine (bare)</td>
<td>1</td>
<td>60%</td>
<td>37.4</td>
<td>49</td>
</tr>
<tr>
<td>MIPEGO (LUT)</td>
<td>1</td>
<td>60%</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>MIPEGO (original)</td>
<td>1</td>
<td>60%</td>
<td>150</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 6.6
Mixed-integer benchmark results- Dimension: 5, FID: 1, DISC: 60%
(5 test functions)

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>FID</th>
<th>DISC</th>
<th>#Evaluations</th>
<th>#Solution metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>MEAN</td>
<td>MAX</td>
</tr>
<tr>
<td>EGO-engine (bare)</td>
<td>1</td>
<td>20%</td>
<td>38.04</td>
<td>54</td>
</tr>
<tr>
<td>MIPEGO (LUT)</td>
<td>1</td>
<td>20%</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>MIPEGO (original)</td>
<td>1</td>
<td>20%</td>
<td>150</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 6.7
Mixed-integer benchmark results- Dimension: 5, FID: 1, DISC: 20%
(5 test functions)

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>DIM</th>
<th>MODE</th>
<th>DISC</th>
<th>#Evaluations</th>
<th>#Solution metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MEAN</td>
<td>MAX</td>
</tr>
<tr>
<td>EGO-engine (bare)</td>
<td>5</td>
<td>fid</td>
<td>20%</td>
<td>98.6</td>
<td>150</td>
</tr>
<tr>
<td>MIPEGO (LUT)</td>
<td>5</td>
<td>fid</td>
<td>60%</td>
<td>76.5</td>
<td>120.6</td>
</tr>
<tr>
<td>MIPEGO (original)</td>
<td>5</td>
<td>iid</td>
<td>20%</td>
<td>50.15</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>iid</td>
<td>60%</td>
<td>40.3</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>fid_iid</td>
<td>20%</td>
<td>99.1</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>fid_iid</td>
<td>60%</td>
<td>80.3</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 6.8
Mixed-integer benchmark results - Dimension 5 (40 test functions)

The results shown in Tables 6.6, 6.7 and 6.8 clearly indicate the expected discrepancies between the compared algorithms. In this case the EGO-engine(bare) that
operates on the LUT is able to find its local global minimum by executing the EGO-based optimization scheme enhanced with dynamic search over the first input variable (50 integer values - in all cases). It computes the expected improvement of all the data in the table and converges toward the optimal solution within on average 41.3 function evaluations for 40 (5-dimensional) mixed integer function of the mixed-integer suite. This situation is the same for the entire mixed-integer benchmark suite and the remaining dimensions. The analysis of the algorithm performance on the composite benchmark functions of 5 dimensions shows there is a correlation between the discretization and the average number of evaluations used. It can be clearly seen that for each categorical mode, the average evaluation numbers consecutively decrease with the increase in discretization degree. The same applies to the solution metric, which improves as the space gets divided into discrete counterparts. This indicates that the more discretized the space, the faster the algorithm explores its solution space and finds the global optimum. Based on the results for test functions with "fid" and "fid_iid" categorical modes, which are characterized by a significantly more diversified landscape than the "iid" mode. One can deduce that the performance of EGO-engine(bare) also heavily depends on the complexity of the solution space. The more linear the trend of the function's landscape, the easier it is for the algorithm’s surrogate model to approximate the physics-based objective function and compute optimal expected improvement values.

As for MIPEGO, given the fact that the algorithm uses a Random Forest model for objective function approximation, there is very little difference between the original and LUT versions compared here. The approximations are suspected to be relatively similar and capture the same amount of information about predicted functions’ nature. One can also notice that they are not dependant on the degree of discretization, based on the average and max metric performance measures of Tables 6.6 and 6.7.

6.5 Discussion

Building up on the results of the experiments, an important question arises. How well can two optimization algorithms be compared to one another? In order to answer this question it is important to keep in mind a theory by Wolpert and Macready called "No free lunch theorem"[23]. It states that different optimization algorithms
might obtain various results over a specific problem, but when tested on all possible problems in existance, they become uncomparable. What follows is that in order to obtain better results in search of an optimal solution of one problem, the algorithm will likely be outperformed when compared to another search algorithm on a different problem. It is obviously possible to conduct a valid performance analysis and comparison, but it is only possible if the algorithms uses the exact same objective functions and explore the exact same search space. Even then, it is still not possible to determine which one of them is an objectively better solution, as the theorem explains. The increase in performance usually comes with the use of heuristics and prior-knowledge as well as matching the algorithmic components and mechanisms to the given optimization task. The conducted comparison between EGO-engine and MIPEGO is the prime example backing up this statement.

One of the contributions of the thesis was an extension of the benchmark suite with mixed-integer and categorical variables. Unfortunately, based on the conducted experiments it is relatively hard to estimate the properties of the newly created functions and deeply explore their nature. That could be achieved by gathering a larger set of mixed-integer optimization algorithms and testing them on a subset of extended problems and see how they behave. Moreover, this kind of experiment might unravel some more information about, in particular, the composite function, which has a rather intricate, unpredictable design.

Future work recommendations would include validating the experimental benchmark extensions, as mentioned above, and possibly comparing the two investigated algorithms with other optimization strategies, making sure to use the exact same search space and objective function in order to get as informative results as possible.
SECTION SEVEN
CONCLUSIONS

In conclusion, this thesis attempted to explore the characteristics and conduct a performance analysis comparison of two Efficient Global Optimization based algorithms. As a part of that work, two extensions to a benchmark suite have been created, adding integer and categorical constrained variables to the problems. An extensive experimental pipeline has been executed and based on the obtained results an analysis of the given approaches has been proposed. Although the results are inconclusive, as explained in Section 6.5, some important information can be derived from this work. **** REFER TO PROBLEM STATEMENT **** The EGO-engine algorithm is an excellent search algorithm, however it is very limited in terms of its generalizability. It can only be applied to problems with finite domains, which significantly reduces its use and ability to compete with approaches that can handle MINLP domains. The MIPEGO algorithm turned out to be a very efficient method of solving the explored industrial problem, despite using no heuristics or prior-knowledge, with very little hyper parameter tuning and optimization. Efficiently comparing the performance of these two approaches on the designed benchmark suite functions was mostly unsuccessful, however, it contributed to confirming some assumptions and providing more insight about the algorithms’ nature.
Bibliography


