Comparison of Parallel Surrogate-Assisted Optimization Approaches

Frederik Rehbach
frederik.rehbach@th-koeln.de
TH Köln
Cologne, Germany

Martin Zaefferer
martin.zaefferer@th-koeln.de
TH Köln
Cologne, Germany

Jörg Stork
joerg.stork@th-koeln.de
TH Köln
Cologne, Germany

Thomas Bartz-Beielstein
thomas.bartz-beielstein@th-koeln.de
TH Köln
Cologne, Germany

ABSTRACT

The availability of several CPU cores on current computers enables parallelization and increases the computational power significantly. Optimization algorithms have to be adapted to exploit these highly parallelized systems and evaluate multiple candidate solutions in each iteration. This issue is especially challenging for expensive optimization problems, where surrogate models are employed to reduce the load of objective function evaluations.

This paper compares different approaches for surrogate model-based optimization in parallel environments. Additionally, an easy to use method, which was developed for an industrial project, is proposed. All described algorithms are tested with a variety of standard benchmark functions. Furthermore, they are applied to a real-world engineering problem, the electrostatic precipitator problem. Expensive computational fluid dynamics simulations are required to estimate the performance of the precipitator. The task is to optimize a gas-distribution system so that a desired velocity distribution is achieved for the gas flow throughout the precipitator. The vast amount of possible configurations leads to a complex discrete valued optimization problem. The experiments indicate that a hybrid approach works best, which proposes candidate solutions based on different surrogate model-based infill criteria and evolutionary operators.

CCS CONCEPTS

• **Mathematics of computing → Discrete optimization**; • **Theory of computation → Continuous optimization**; **Gaussian processes**; • **Computing methodologies → Modeling and simulation**;

KEYWORDS

Optimization, Surrogates, Modeling, Parallelization, Electrostatic Precipitator

ACM Reference Format:
Frederik Rehbach, Martin Zaefferer, Jörg Stork, and Thomas Bartz-Beielstein. 2018. Comparison of Parallel Surrogate-Assisted Optimization Approaches. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205587

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205587

1 INTRODUCTION

Real-world optimization problems may require significant resources for each evaluation of a candidate solution. Often, such problems are based on time consuming computational fluid dynamics (CFD) simulations, where a single simulation might take hours, days, or even weeks. Thus, even a few hundred function evaluations may result in run times of several weeks or months. In practice such long time frames are infeasible and render the problems very difficult to solve. Two well established concepts allow to deal with these issues: surrogate model based optimization (SMBO) and parallel computing.

SMBO tries to learn a data-driven surrogate model which replaces the expensive objective function. Under the assumption that the model is cheap to evaluate, an extensive search becomes feasible. Thus, promising new solutions can be suggested, evaluated with the objective function and used to update the surrogate model in an iterative fashion. A more in-depth explanation of SMBO and its applications can be found in [1] and [13].

Parallel computing attempts to exploit the availability of several CPU cores which can operate simultaneously.

While it is straight-forward to parallelize a CFD simulation by spreading it onto several CPU cores, there is a limit on how many cores can be used efficiently for a single simulation. Hence, with increasing numbers of available cores, it will become more efficient to run several simulations in parallel. However, not all existing optimization methods can be applied in a parallel manner. Notably, the standard SMBO approach does not take significant advantage out of a multi-core system. Despite its success in the domain of expensive objective functions, it is not efficient for problems where parallelized function evaluations are possible. Standard SMBO lacks

a straight forward approach for generating multiple new design points in each iteration.

In the following, Section 2 will give an overview on related research and existing methods. Section 3 presents two structures for using SMBO in a parallelized environment. They were used in an industrial project for optimizing an electrostatic precipitator (ESP). Section 4 describes the ESP problem and its complexity in detail. An experimental study compares the different methods based on the ESP problem and artificial test functions and is described in Section 5. Results are presented and discussed in Section 6. A conclusion and an outlook are presented in Section 7.

2 RELATED RESEARCH

2.1 Efficient Global Optimization

One frequently used surrogate model is Kriging. In Kriging, an objective function value can be estimated for a given new candidate solution by building a Gaussian process model, based on a correlation structure derived from the observed training data[5].

Kriging is frequently used in SMBO, because it can provide an estimate of its uncertainty. Conditional on a given candidate solution, Kriging specifies a normal distribution of the corresponding objective value, where the mean is the predicted value and the standard deviation is the uncertainty. Notably, the uncertainty estimate can be employed to calculate the Expected Improvement (EI) [11] of a candidate solution, which is used in the Efficient Global Optimization (EGO) algorithm [8]. The main idea in EGO is not to search the surrogate for the point with the best predicted objective function value. Instead, the uncertainty estimation of the model is taken into account by maximizing the EI criterion. Intuitively, maximizing EI balances exploration (improving the model or knowledge about the search space) and exploitation (improving the objective function value). EI increases when the predicted value gets better, as well as when the uncertainty rises.

Clearly, EGO and EI in their original forms are meant to generate a single, most promising design point (i.e., the solution that maximizes EI).

2.2 Investment Portfolio Improvement

Investment Portfolio Improvement (IPI) tries to further improve the idea of balancing between exploration and exploitation of the search strategy. Ursem [17] suggests that one can judge each candidate solution from the viewpoint of an investment portfolio. The candidate solution with the highest probability of improvement is seen as a low risk investment. Candidate solutions which are optimized towards the highest expected improvement yield a higher risk but also might lead to better function value outcomes. With IPI, it is possible to generate multiple candidate solutions per iteration on a single surrogate model. Each of the generated solutions will aim for a different risk level. This is achieved by either preferring solutions with a high uncertainty (EI) or solutions with low uncertainty but a better predicted value.

IPI generates a set of three design points per iteration: a high, a medium, and a low risk design point. Each of these design points is then evaluated sequentially. Thus, IPI balances risks in single-threaded optimization. However, since IPI is theoretically able to generate any amount of design points with a given surrogate, it can also be employed to parallelize an SMBO algorithm. Then, the amount of generated design points per iteration is equal to the maximum amount of objective function evaluations in parallel.

2.3 Multi-Point Expected Improvement (q-EI)

An intuitive idea for parallel SMBO is to extend EI to point sets. The multi point expected improvement criterion was first proposed in [15] and later further developed in [6]. It calculates the expected improvement criterion for a set of candidate solutions. Ginsbourger et al. [6] provide a comprehensible description of the q-EI calculation. Here, q is the set-size for which EI is computed.

Since q-EI allows to calculate the EI criterion for any number of candidate solutions, it is a very promising choice for parallelized EGO. At each iteration, the set of candidate solutions that optimizes the q-EI criterion is determined. The set size equals the amount of objective function evaluations that can be performed in parallel. Hence, the main difference between SMBO and SMBO with q-EI is the amount of solutions generated per iteration.

One inherent property of the q-EI criterion is that it favors sets of solutions that are spread far from each other in the search space. If two or more candidate solutions move close to each other, their combined expected improvement will approach the value of the best. Effectively, only one point from some small cluster would improve the q-EI of the whole set, the others are negligible.

2.4 Further Approaches

A recent survey of parallel SMBO is given by Haftka et al. [7]. They describe several approaches based on Gaussian process models and uncertainty based infill criteria: parallel EGO/EI, multi-point probability of improvement (PI) and approaches based confidence bounds. Furthermore, the survey discusses several approaches based on models without uncertainty information, multi-model approaches (e.g., one model for each parallel thread), approaches that explore (local) sub-spaces of the search space and discuss hybrids of surrogate-assisted optimization and evolutionary computation.

In addition, there have been studies on parallel SMBO with treed gaussian process (TGP) [16]. They use a combination of TGP models together with a local pattern search optimizer. Both of these techniques are combined into an asynchronous parallel computing environment. Here, a method similar to the q-EI is used to generate multiple design points per iteration, where candidate points are added in a sequential manner.

Another idea proposed by Bischl et al. [2] is to treat the problem in a multi-objective manner. They suggest a number of potential objectives, including EI, mean, probability of improvement, uncertainty (variance estimate), distance to nearest neighbor and distance to nearest better neighbor. They report that a combination of mean, uncertainty and distance to nearest neighbor performed best in a set of numerical experiments.

Furthermore, expensive evaluation times may vary, e.g., CFD simulations may differ in time consumption, depending on the evaluated candidate solution. To that end, Richter et al. [14] propose an asynchronous approach that attempts to produce evaluation schedules that reduce the overall time consumption. They use surrogate models to approximate the objective function results as well as to approximate the required resources.

3 SURROGATE ASSISTED OPTIMIZATION PARALLEL TO STANDARD OPTIMIZERS

We propose to use a hybrid algorithm composed of standard SMBO and EAs. It is able to produce new candidate solutions in two different ways: 1) via standard infill criteria such as best-predicted and EA, and 2) via evolutionary operators. A parallel design for SMBO was developed, which is able to solve the ESP problem. The usage of the best-predicted infill criterion, as well as the EI infill criterion are well established and yield good results in single threaded applications. Thus, the hybrid approach will employ these two infill criteria and utilize any remaining computational resources for optimization with an EA. In the following, we introduce a synchronous and asynchronous parallel design of the proposed hybrid algorithm. Both designs arise from a optimization task in an ongoing project and are aimed for an easy parallelization of an already existing optimizer. The basic concept is visualized in Figure 1.

Algorithm 1 Synchronous SMBO+EA hybrid. Here, n_{init} is the number of initial candidate solutions, design() is a function that produces an initial set of candidate solutions, train() is a procedure to train an adequate surrogate model, and optimize() is an evolutionary algorithm implementation with variation operators eaMutation() and eaRecombination(). The function evalParallel() represents the potentially expensive objective function, which allows for evaluating n solutions simultaneously.

function SMBO+EA-Sync(n_{init}, design(), train(), optimize(), evalParallel(), eaMutation(), eaRecombination())
  $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{n_{init}}\}$ = design(n_{init})
  $\boldsymbol{y}$ = evalParallel($\boldsymbol{X}$)
  while budget not exhausted **do**
    model = train($\boldsymbol{X}, \boldsymbol{y}$)
    $\boldsymbol{x}_{s1}$ = optimize(BP(model)) ▷ optimize best-predicted
    $\boldsymbol{x}_{s2}$ = optimize(EI(model)) ▷ maximize EI
    $\boldsymbol{X}_o$ = eaMutation() + eaRecombination()
    $\boldsymbol{X}_{os} = \{\boldsymbol{X}_o, \boldsymbol{x}_{s1}, \boldsymbol{x}_{s2}\}$
    $\boldsymbol{y}_{os}$ = evalParallel($\boldsymbol{X}_{os}$)
    $\boldsymbol{X} = \{\boldsymbol{X}, \boldsymbol{X}_{os}\}$
    $\boldsymbol{y} = \{\boldsymbol{y}, \boldsymbol{y}_{os}\}$
  end while
end function

The first, synchronous design is presented in Algorithm 1. In difference to standard (synchronous) SMBO procedures, both the best predicted and EI criterion infill criteria are utilized to create two candidates per iteration and a set of candidates is generated in parallel by genetic operators (mutation, recombination). All generated candidates are passed to a task queue, which handles the parallel evaluation of all candidates with the real objective function. The next generation starts after all candidates are evaluated.

The second, asynchronous design is introduced to account for the computational costs of the surrogate model fitting and optimization. Often, surrogates are considered to have zero runtime since they are, compared to the real objective function, very quick to evaluate. However, they often demand considerable resources, depending on given sample sizes, problem dimension and type of employed model. In the synchronous version, no evaluations are started until the surrogate is fitted and utilized to propose new candidates. As this procedure is not performed in parallel, available resources are not used efficiently. This is where an asynchronous architecture may be beneficial.

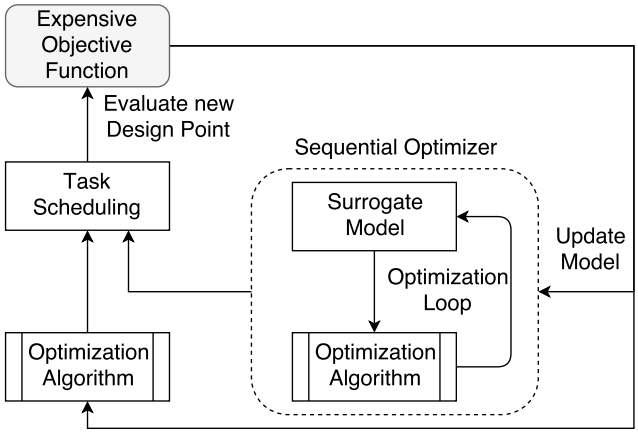

Figure 1: A surrogate model implemented in parallel to the optimization algorithm. One algorithm is used to optimize the surrogate. Another optimizer (which can be another instance of the same optimizer), directly proposes candidate solutions on the objective function. Through a scheduler, both sets of candidate solutions are evaluated. The surrogate is updated with new data from these evaluations.

Algorithm 2 shows a pseudo-code implementation of the asynchronous optimization structure. The main idea of this asynchronous design is that training and optimization of the surrogate model can be performed in parallel to the objective function evaluations. In each iteration of the asynchronous algorithm, a check is performed if there have been any new candidate solutions generated by the surrogate optimization thread. If so, both EA candidates and surrogate candidates are evaluated with the objective function. If not, the free computation slots are filled with candidates generated by the genetic operators.

4 THE ESP PROBLEM

The ESP is a real industrial optimization problem. The ESP is one of the main components of gas cleaning systems. They are used in large scale coal-fired power plants or other industries where solid particles have to be removed from a gas stream. ESPs are large devices with dimensions of around 30m × 30m × 50m, resulting in multiple millions of euros just in building cost. The main task of an ESP is to separate and extract particles from exhaust gases in order to reduce environmental pollution. Figure 2 illustrates this system.

In the flue gas inlet hood of an ESP, a gas distribution system (GDS) (shown in Figure 3) is required to control and guide the gas flow through separation zones in which particles are removed from the exhaust gases. If no GDS is used, or if the system is configured poorly, the fast inlet gas stream will rush through the separation zones of the ESP. This results in very low separation efficiencies. In case of a well configured GDS, the inflowing gas is nicely distributed across the whole surface of the separation zones, resulting in high

Algorithm 2 Asynchronous SMBO+EA hybrid. Here, n_{init} is the number of initial candidate solutions, design() is a function that produces an initial set of candidate solutions, train() is a procedure to train an adequate surrogate model, and optimize() is an evolutionary algorithm implementation with variation operators eaMutation() and eaRecombination(). The function evalParallel() represents the potentially expensive objective function, which allows for evaluating n solutions simultaneously.

function SMBO+EA-ASYNC(n_{init}, design(), train(), optimize(), evalParallel(), eaMutation(), eaRecombination())
  $\boldsymbol{X}_s$ ={}
  $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{n_{init}}\}$ = design(n_{init})
  $\boldsymbol{y}$ = evalParallel($\boldsymbol{X}$)
  function GENERATESURROGATECANDIDATES
    model = train($\boldsymbol{X}, \boldsymbol{y}$)
    $\boldsymbol{x}_{s1}$ = optimize(BP(model)) ▷ optimize best-predicted
    $\boldsymbol{x}_{s2}$ = optimize(EI(model)) ▷ maximize EI
    $\boldsymbol{X}_s = \{\boldsymbol{x}_{s1}, \boldsymbol{x}_{s2}\}$
  end function
  while budget not exhausted **do**
    $\boldsymbol{X}_o$ = eaMutation() + eaRecombination()
    $\boldsymbol{X}_{os} = \{\boldsymbol{X}_o, \boldsymbol{X}_s\}$
    $\boldsymbol{y}_{os}$ = evalParallel($\boldsymbol{X}_{os}$)
    do parallel: generateSurrogateCandidates()
    $\boldsymbol{X} = \{\boldsymbol{X}, \boldsymbol{X}_{os}\}$
    $\boldsymbol{y} = \{\boldsymbol{y}, \boldsymbol{y}_{os}\}$
  end while
end function

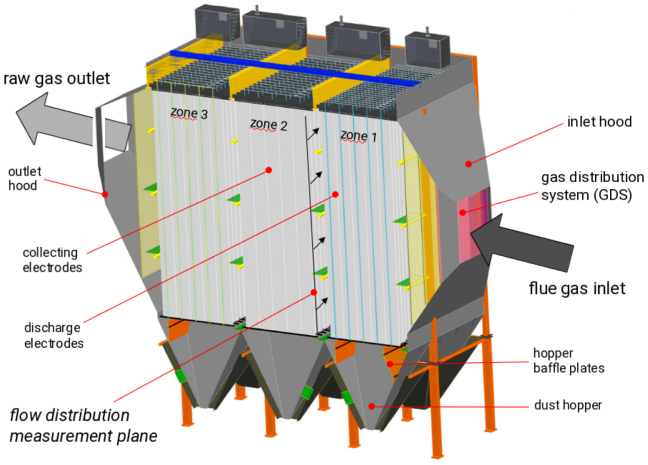

Figure 2: ESP with 3 separation zones. This figure was kindly provided by Steinmüller Babcock Environment GmbH.

efficiency. Hence, the efficient operation of an ESP requires an optimal configuration of the GDS. The GDS in our example consists of 49 configurable slots. Each of these slots can be configured with baffles, as well as blocking and perforated plates. Baffles are metal plates which are mounted at an angle to the general gas flow. They are used to redirect a gas stream into a new direction. Blocking plates completely block a gas stream. Perforated plates are used to slow down and only partially block gas streams. They are created by punching a grid of holes into metal plates. Smaller holes lead to higher pressure drops and thus a slower gas stream. Larger holes allow for a nearly free gas flow. These plates can be mounted into each of the 49 configurable slots. Optionally, some slots can also be left empty.

Here, 40 out of 49 of the slots can be configured with eight different types of plates (including empty). The remaining nine slots can only be configured as "true" or "false", which indicates a blocking plate or an empty slot. The vast amount of possible combinations of the GDS reveals a complex discrete optimization problem. For a single evaluation of a given configuration, a computationally expensive CFD simulation is necessary, which results in hours of computation time. Unfortunately, such large computation times make the ESP problem unsuitable for a large number of tests runs which are necessary to derive reasonable conclusions about the performance of several competing algorithms. Therefore, a second model with a largely reduced amount of cells in the simulation mesh was created. By doing so, the runtime of the model was reduced to only a few seconds per evaluation. This speed up comes at the cost of reduced simulation accuracy. However, the reduced model still captures some of the difficulties and complex features of the actual problem, while enabling a detailed experimental study. Reproducing the rugged problem landscape is much more important than the actual accuracy of each sample point.

The open source CFD framework OpenFOAM [18] was used to implement our simulations. The original landscape of a real industrial problem is transferred into a function which can be evaluated in reasonable computation time. The ESP problem was therefore considered as a good industrial benchmark for this paper. Currently, the accelerated simulation model is used to tune and advance the research in algorithms for the optimization of the full long-running simulation.

5 DESCRIPTION OF EXPERIMENTS

5.1 Performance Measure

One simple way of measuring optimization performance is to record the best objective function value attained after a fixed number of function evaluations. For parallel optimization, this is not suitable. Some algorithms are capable of invoking multiple objective function evaluations at once. Other algorithms are only able to do these sequentially. Thus, algorithms which can not parallelize, use less function evaluations in the same computation time, resulting in an unfair comparison if only the amount of evaluations is considered. Measuring the total execution time of algorithms is also not always feasible. Algorithm run times largely rely on factors like their implementation, programming language or the machine they are run on. Therefore, a different approach for performance measurements was chosen. Each algorithm is given a fixed amount of iterations instead of function evaluations. For each iteration, the amount of possible evaluations in parallel is fixed. We record the best objective function value attained after a fixed number of iterations. Thus, an algorithm which uses all available cores should be more efficient as it can do more function evaluations.

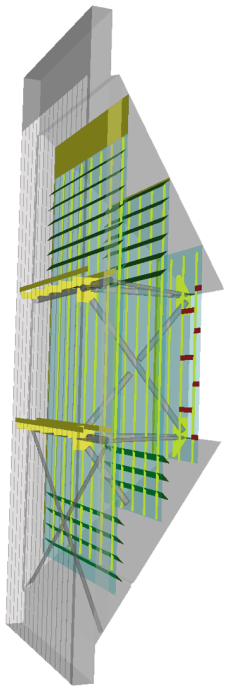

Figure 3: Visualization of a gas distribution system (GDS) mounted in the inlet hood of of an ESP. This figure was kindly provided by Steinmüller Babcock Environment GmbH.

5.2 Methods and Configurations

Each optimization algorithm starts of with an initial design of the size of n, where n is the amount of objective function evaluations which are possible in parallel. After the initialization, each algorithm performs 50 iterations. Depending on the value of n, which is varied between 3 and 15, the algorithms were able to do a maximum of 150-750 function evaluations.

With this algorithm setup, 14 different methods were compared, including a set of base line comparisons, variations of SMBO+EA, IPI, and two versions of q-EI. An overview of all methods and their composition is given in Table 1. All algorithms were implemented in R. Each time an EA was applied, the optimEA method of the CEGO [19] package was used. The population size and mutation rate were set to 20 and 0.05 respectively. They were determined to work best in preliminary runs. The other parameters were kept at the package defaults.

In the following the implementation of each of the 14 methods is described shortly. Two variants of single threaded standard SMBO were implemented as a base line for the comparison. Here, single threaded indicates that no parallel evaluations will be performed. Each variant is based on a different infill criterion (BP and EI), and both use Kriging. Hence, they will be denoted as Krig-BP and Krig-EI. The optimum of the infill criteria was determined with a simple EA from the same package. To judge the performance of both of these single threaded SMBO implementations, a single-threaded and model-free EA is used (denoted as EA singleCore).

Table 1: Optimization methods. Some of the methods are hybrids, the table shows their composition by detailing the amount of objective function evaluations allowed to each method. Here, n is the total amount of evaluations possible in parallel. $m = n - 3$ for all cases where $n > 3$ otherwise $m = n - 2$, $p = 1$ for $n > 3$ and $p = 0$ for $n = 3$.

Index	EA	BP	EI	Rnd. Samp.	Space Filling	IPI	q-EI
EA singleCore	1						
Krig-BP		1					
Krig-EI			1				
K.BP+K.EI		1	1				
EA-nCores	n						
K.BP+K.EI +Rnd.Samp.		1	1	$n-2$			
K.BP+K.EI+LHS		1	1		$n-2$		
SMBO+EA-async	$n-2$	1	1				
SMBO+EA-async+SF	m	1	1		p		
SMBO+EA-sync	$n-2$	1	1				
SMBO+EA-sync+SF	m	1	1		p		
IPI-n-cores						n	
q-EI							n
q-EI-Bounded							n

As another base line, we test a model-free EA that generates as many individuals per iteration as there are slots for parallel function evaluation (EA-nCores). The proposed SMBO+EA algorithm uses both BP and EI together. Thus, to judge the EAs impact on the hybrid algorithm, another experiment was added where both EI and BP are implemented without further algorithms.

Methods like latin hypercube sampling (LHS) are often argued to increase the model quality of surrogates by generating space-filling (SD) designs in the search space. This methodology for model quality improvements was also implemented in a CFD airfoil optimization by Marsden et al. [10]. To test this hypothesis two further methods were tested: K.BP+K.EI+Rnd.Samp and K.BP+K.EI+LHS. Both generate two candidate solution per iteration, one via BP and one via EI. In K.BP+K.EI+LHS, the remaining $n - 2$ available evaluation slots are populated with LHS. In addition to the Latin hypercube property, the points are determined to maximize the distance to their nearest neighbor. In K.BP+K.EI+Rnd.Samp, the slots are populated with random samples.

Furthermore, four distinct versions of a hybrid SMBO+EA were tested. Both the synchronous and asynchronous versions of SMBO +EA were implemented (SMBO+EA-sync, SMBO+EA-async). In addition, the benefits of a space-filling (SF) infill point were investigated in the SMBO+EA structure. To that end, the synchronous and the asynchronous version of SMBO+EA were extended as follows. Firstly, one less solution is generated by the EA's variation operators. This slot is filled with a candidate solution that maximizes the minimal distance to the other candidate points. This approach is similar to the distance to nearest neighbor objective in the study by Bischl et al. [2]. The respective algorithms are denoted with SMBO+EA-sync+SF and SMBO+EA-async+SF.

The IPI method and q-EI were implemented as described in the Related Research Section 2. Since IPI was originally implemented to generate multiple candidate solutions that are sequentially evaluated, the methodology was slightly altered. In our implementation,

the CEGO buildKriging function is used to build a Kriging model. Then, an EA is used to optimize each of IPIs infill criteria, so that n candidate solutions are generated in each iteration. They are evaluated in parallel and the model is trained with the updated data set.

For q-EI, we employ the suggested implementation from Ginsbourger et al. [6], which is available in the DiceKriging R-Package. Their implementation of the q-EI infill criterion was optimized by CEGOs optimEA. However, the task of the optimization was not to search a single point, but the best set of points that optimizes the q-EI criterion. Due to the behavior of the q-EI optimization in first tests, a second variant of q-EI based SMBO was implemented, where bound constraints are respected. Note, that otherwise all methods only respect bound constraints where applicable, and only during design creation. This issue will be discussed in more detail in Section 6.

5.3 Test Functions

In addition to the real world ESP problem, the described methods were applied to the following test functions: Rosenbrock 2D, Rastrigin 5D, Rastrigin 10D, Branin 2D, Hartmann 6D, and Colville 4D. As these additional functions are cheap to evaluate, using surrogate models to optimize them is not efficient. Yet, we argue that the results are transferable to other more costly functions. Due to the various landscapes and features provided by the test functions, the experiments yield useful information on the surrogates performance. Making numerous experiments on more expensive functions would lead to an infeasible computational cost. For each test function and method, 60 repeated optimization experiments were performed to account for the stochastic behavior of the algorithms. Only 30 repeats were done for the ESP problem, due to its larger computation times. The q-EI variants were not tested on the ESP problem. As noted above, the q-EI implementation from the DiceKriging package is used. The Kriging model in this package requires that the number of observations are greater than or equal to the number of variables. Given the high dimensionality and computation cost of the ESP problem as well as its discrete nature, this condition can not be reasonably satisfied. Especially in the early stages of the experiments, data set sizes will be smaller than required.

6 RESULTS AND DISCUSSION

Our presentation of the results is based on a statistical analysis. For each test-problem, we performed a statistical multiple-comparisons test. Differences are judged significant if the corresponding p-values are smaller than $\alpha = 0.05$. We computed a ranking based on the derived pair-wise comparisons. Here, the ranking is performed as follows. Any algorithm that is never significantly worse than any other algorithm receives rank one, and is removed from the list. Of the remaining algorithms, the ones that are not worse than any other receive rank two and are also removed. This procedure repeats until all algorithms are ranked.

We chose to use the Kruskal-Wallis test [9] (to check whether any significant differences are present) and a corresponding post-hoc test based on Conover (to determine which algorithm pairs are actually different) [3, 4]. We use the implementations from the PMCMR R package [12]. These tests were chosen as the data is not normal distributed, and is also heteroscedastic (i.e., group variances are not equal). Hence, parametric test procedures that assume homoscedastic (equal variance), normal distributed data are unsuited. Note, that non-parametric tests are not free of assumptions. In fact, the Kruskal-Wallis test assumes that the data are random samples, statistically independent within each group and between groups, and have an ordinal measurement scale [3, p. 289]. These assumptions should hold for the optimization performance results we consider. An overview of the analysis results can be found in Table 2.

It is clearly visible that, as expected, all single threaded baseline implementations perform worst on each test functions. This confirms that parallelization is necessary to efficiently optimize in an environment where more than one function evaluation is possible at the same time. In the first set of results with $n = 3$ parallel evaluations, the ESP problem is best solved by methods which apply both EI and BP plus an additional method to specify the third point to be evaluated. Interestingly, there seem to be no significant differences between methods for determining the third point (EA, SF, or random sampling). On the standard test functions the bounded version of q-EI scored best.

Yet, the fairness of a comparison to this bounded approach is arguable. As first tests showed that the original implementation of the EA-optimized q-EI criterion yielded unsatisfactory results, the problem was further investigated. In many of the experiments, it was observed, that q-EI only positioned one or two design points in a reasonable search area. The other design points were rather far spread out into regions where the kriging model is estimating maximum uncertainty. Thus, q-EI often yields points that lie far outside the previously sampled regions. This was the main reason for introducing bound constraints to the q-EI implementation. In this algorithm variant, the EA which is used to optimize the q-EI criterion is limited to a search in a bounded area. This solved the basic issues q-EI was facing and yielded much better results. However, it has to be considered that problem bounds are not always known a priori, and that the other SMBO implementations were not subject to bounds (with the exception of the initial design generation that is shared by all methods). The striking performance of q-EI with bounds and $n = 3$ should therefore be considered with care.

Given five or more evaluations in parallel, scaling seems to become an issue in IPI and q-EI as their mean rank drops. Here, SMBO seems to deliver the best results for the ESP-problem, Rastrigin, Branin and Hartmann. On the Rosenbrock and Colville functions, q-EI consistently delivers the best results.

Figures 4 and 5 are taken as examples to present results for a given test function and specific value of n parallel evaluations in the form of box plots. It can be observed, that for larger n, the synchronous and asynchronous version of SMBO+EA are not significantly distinguishable from one another. Thus, since the asynchronous version yields less overall computation time, it should be preferred. In cases where $n < 10$, the synchronous model outperformed the asynchronous one. Therefore, a closer analysis of the computation time reduction through the asynchronous model is required in order to choose the best approach. Lastly, Figure 5 reveals that on the

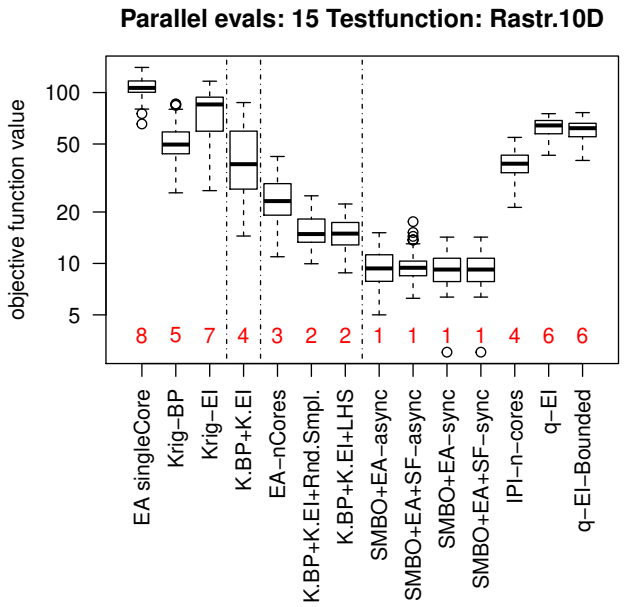

Figure 4: Boxplot showing the optimization results on the 10 dimensional rastrigin function. In these experiments 15 function evaluations were possible in parallel. Red numbers at the bottom indicate the given rank based on pairwise multiple-comparison tests. NAs due to q-EI implementation on ESP problem, described in Section 5.3.

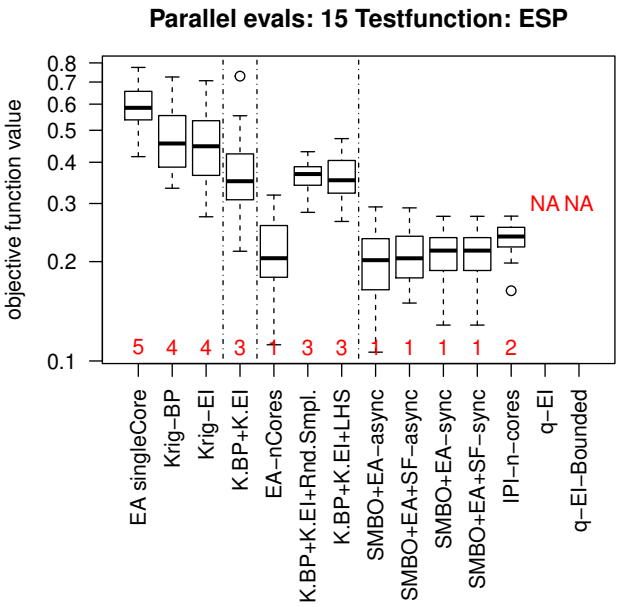

Figure 5: Optimization results on the high dimensional ESP-problem. In the experiments 15 function evaluations were possible in parallel. Red numbers at the bottom indicate the given rank based on pairwise multiple-comparison tests.

ESP problem with $n = 15$, SMBO+EA does not outperform a simple model-free parallelized EA.

The same is true for $n = 10$, but not for smaller n. This behavior may be explained by the nature of the SMBO+EA hybrid. The number of solutions suggested by the model is constant (here: two), whereas the number of solutions suggested by the EA operators increase with n. Hence, the hybrid will tend to behave more similar to a model-free EA for larger n. This indicates that it may be necessary for the model-based part to scale with n, too, to provide better performance. Hence, it may be profitable to combine it with the q-EI or IPI approaches.

7 SUMMARY AND OUTLOOK

In conclusion, the given results show that SMBO is very well applicable to parallelized server environments. However, the given results indicate that scaling to a high level of parallelization is still an issue in current state of the art SMBO algorithms. More future research in the field of parallelization of expensive to evaluate objective functions is required.

For the SMBO+EA algorithm design, many possibilities for future improvements exist. Firstly, selection criteria for choosing which EA generated individuals shall be evaluated on the objective function should be compared and implemented. That is, the surrogate model may be employed to select the more promising of the candidate solutions proposed by the EA operators.

Also, a switching criterion between the synchronous and asynchronous SMBO+EA architecture should lead to better performance. In the beginning of any optimization run, few data points are available to fit a surrogate model. Thus, each data point is essential for model quality. Due to the low amount of data at this point, model training and optimization requires less time. However, with the growing amount of evaluated candidate solutions, the impact of each new candidate solution on the model quality diminishes and computational cost of the model increases. Thus starting with the synchronous implementation and switching to the asynchronous one in the later optimization progress should yield better performance.

REFERENCES

[1] Thomas Bartz-Beielstein and Martin Zaefferer. 2017. Model-based methods for continuous and discrete global optimization. *Applied Soft Computing* 55 (2017), 154 – 167. https://doi.org/10.1016/j.asoc.2017.01.039
[2] Bernd Bischl, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs. 2014. MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization. In *Learning and Intelligent Optimization*, Panos M. Pardalos, Mauricio G.C. Resende, Chrysafis Vogiatzis, and Jose L. Walteros (Eds.). Springer International Publishing, Cham, 173–186.
[3] William Jay Conover. 1999. *Practical Nonparametric Statistics, 3rd Edition.* Wiley.
[4] William Jay Conover and Ronald L. Iman. 1979. *On multiple-comparisons procedures.* Technical Report Tech. Rep. LA-7677-MS. Los Alamos Sci. Lab.
[5] Alexander Forrester, Andy Keane, et al. 2008. *Engineering design via surrogate modelling: a practical guide.* John Wiley & Sons.
[6] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. 2010. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems.* Springer, 131–162.
[7] Raphael T. Haftka, Diane Villanueva, and Anirban Chaudhuri. 2016. Parallel surrogate-assisted global optimization with expensive functions – a survey. *Structural and Multidisciplinary Optimization* 54, 1 (01 Jul 2016), 3–13.

Table 2: Overview of optimization performances on all problems. Results are ranked based on pairwise multiple-comparison tests. For each problem, the best method is marked in bold font. The last column gives the mean rank. Best mean rank is underlined and printed in bold font.

n	method	ESP	Rose.2D	Rast.5D	Rast.10D	Bran.2D	Hart.6D	Colv.4D	mean
3	EA singleCore	4	5	5	6	4	6	4	4.857
	Krig-BP	3	5	**1**	**1**	6	6	2	3.429
	Krig-EI	3	6	6	7	5	5	5	5.286
	K.BP+K.EI	2	4	3	5	4	4	3	3.571
	EA-nCores	2	4	3	3	2	3	**1**	2.571
	K.BP+K.EI +RandomSamples	**1**	3	3	4	3	4	3	3.000
	K.BP+K.EI+LHS	**1**	3	4	4	3	4	3	3.143
	SMBO+EA-async	**1**	3	2	**1**	2	3	**1**	1.857
	SMBO+EA+SF-async	**1**	3	2	2	2	3	**1**	2.000
	SMBO+EA-sync	**1**	3	2	**1**	2	2	**1**	1.714
	SMBO+EA+SF-sync	**1**	3	2	**1**	2	3	**1**	1.857
	IPI-n-cores	2	2	2	4	5	2	2	2.714
	q-EI	NA	2	4	4	2	2	3	2.833
	q-EI-Bounded	NA	**1**	2	3	**1**	**1**	**1**	**1.500**
5	EA singleCore	5	5	5	6	4	7	5	5.286
	Krig-BP	4	6	**1**	2	5	7	4	4.143
	Krig-EI	4	6	6	6	4	6	6	5.429
	K.BP+K.EI	3	5	4	5	4	5	5	4.429
	EA-nCores	2	5	3	3	3	4	2	3.143
	K.BP+K.EI +RandomSamples	2	4	3	4	3	5	4	3.571
	K.BP+K.EI+LHS	3	4	3	3	3	5	4	3.571
	SMBO+EA-async	**1**	4	**1**	**1**	2	2	2	1.857
	SMBO+EA+SF-async	2	4	2	2	3	3	3	2.714
	SMBO+EA-sync	**1**	4	**1**	**1**	2	**1**	2	**1.714**
	SMBO+EA+SF-sync	**1**	3	**1**	2	3	3	3	2.286
	IPI-n-cores	2	3	2	4	4	**1**	2	2.571
	q-EI	NA	2	5	5	2	4	**1**	3.167
	q-EI-Bounded	NA	**1**	3	4	**1**	2	**1**	2.000

n	method	ESP	Rose.2D	Rast.5D	Rast.10D	Bran.2D	Hart.6D	Colv.4D	mean
10	EA singleCore	5	7	6	8	6	6	6	6.286
	Krig-BP	4	7	3	4	6	5	4	4.714
	Krig-EI	4	7	6	8	5	5	6	5.857
	K.BP+K.EI	3	6	4	5	3	4	5	4.286
	EA-nCores	**1**	5	4	3	4	3	3	3.286
	K.BP+K.EI +RandomSamples	3	4	2	2	2	3	4	2.857
	K.BP+K.EI+LHS	3	4	2	2	2	3	4	2.857
	SMBO+EA-async	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA+SF-async	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA-sync	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA+SF-sync	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	IPI-n-cores	2	2	2	4	3	2	3	2.571
	q-EI	NA	2	5	7	2	3	**1**	3.333
	q-EI-Bounded	NA	**1**	5	6	**1**	2	**1**	2.667
15	EA singleCore	5	6	5	8	6	5	7	6.000
	Krig-BP	4	6	3	5	6	4	6	4.857
	Krig-EI	4	5	4	7	5	4	7	5.143
	K.BP+K.EI	3	5	3	4	4	3	6	4.000
	EA-nCores	**1**	4	3	3	4	3	3	3.000
	K.BP+K.EI +RandomSamples	3	4	2	2	3	3	5	3.143
	K.BP+K.EI+LHS	3	4	2	2	3	2	5	3.000
	SMBO+EA-async	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA+SF-async	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA-sync	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	SMBO+EA+SF-sync	**1**	3	**1**	**1**	**1**	**1**	2	**1.429**
	IPI-n-cores	2	2	2	4	3	**1**	4	2.571
	q-EI	NA	2	4	6	3	2	**1**	3.000
	q-EI-Bounded	NA	**1**	4	6	2	2	**1**	2.667

https://doi.org/10.1007/s00158-016-1432-3

[8] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (01 Dec 1998), 455–492. https://doi.org/10.1023/A:1008306431147

[9] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (1952), 583–621.

[10] Alison L. Marsden, Meng Wang, John E. Dennis, and Parviz Moin. 2004. Optimal Aeroacoustic Shape Design Using the Surrogate Management Framework. *Optimization and Engineering* 5, 2 (01 Jun 2004), 235–262. https://doi.org/10.1023/B:OPTE.0000033376.89159.65

[11] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. 1978. *Towards Global Optimization 2.* North-Holland, Chapter The application of Bayesian methods for seeking the extremum, 117–129.

[12] Thorsten Pohlert. 2014. The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR). (2014). http://CRAN.R-project.org/package=PMCMR R package.

[13] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. 2005. Surrogate-based analysis and optimization. *Progress in aerospace sciences* (2005).

[14] Jakob Richter, Helena Kotthaus, Bernd Bischl, Peter Marwedel, Jörg Rahnenführer, and Michel Lang. 2016. Faster Model-Based Optimization Through Resource-Aware Scheduling Strategies. In *Learning and Intelligent Optimization*, Paola Festa, Meinolf Sellmann, and Joaquin Vanschoren (Eds.). Springer International Publishing, Cham, 267–273.

[15] Matthias Schonlau. 1997. Computer experiments and global optimization. (1997).

[16] Matthew A. Taddy, Herbert K. H. Lee, Genetha A. Gray, and Joshua D. Griffin. 2009. Bayesian Guided Pattern Search for Robust Local Optimization. *Technometrics* 51, 4 (2009), 389–401. https://doi.org/10.1198/TECH.2009.08007 arXiv:http://dx.doi.org/10.1198/TECH.2009.08007

[17] Rasmus K. Ursem. 2014. *From Expected Improvement to Investment Portfolio Improvement: Spreading the Risk in Kriging-Based Optimization.* Springer International Publishing, Cham, 362–372. https://doi.org/10.1007/978-3-319-10762-2_36

[18] Henry G Weller, G Tabor, Hrvoje Jasak, and C Fureby. 1998. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics* 12, 6 (1998), 620–631.

[19] Martin Zaefferer, Joerg Stork, Martina Friese, Andreas Fischbach, Boris Naujoks, and Thomas Bartz-Beielstein. 2014. Efficient Global Optimization for Combinatorial Problems. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO'14).* ACM, New York, NY, USA, 871–878. http://doi.acm.org/10.1145/2576768.2598282