Contributions to the Analysis and Design of Parallel Batched Bayesian Optimization Algorithms

Maxime Gobert

Université de Mons - Faculté Polytechnique
Informatique et Gestion - Mathématique et Recherche Opérationnelle

Contributions to the Analysis and Design of Parallel Batched Bayesian Optimization Algorithms

Thèse de doctorat

Presentée et soutenue par

Maxime GOBERT

le 21 juin 2024
Dans le cadre de l'obtention du grade de
Docteur en sciences de l'ingénieur et technologies

Sous la direction de :
Daniel TUYTTENS (Promoteur)
Nouredine MELAB (Co-promoteur)

Devant le jury composé de :

VALLEE François	Professeur	Président	Université de Mons
TUYTTENS Daniel	Professeur	Promoteur	Université de Mons
MELAB Nouredine	Professeur	Co-promoteur	Université de Lille
DANOY Grégoire	Chargé de Recherche, HDR	Rapporteur	Université du Luxembourg
SENS Pierre	Professeur	Rapporteur	Sorbonne Université
MAHMOUDI Saïd	Professeur	Secrétaire	Université de Mons
GMYS Jan	Ingénieur de Recherche	Membre	Université de Lille

Remerciements

Par ces quelques lignes, je souhaite tout d'abord remercier mes promoteurs, Daniel Tuyttens, Professeur et chef du service MARO à l'Université de Mons, et Nouredine Melab, Professeur à l'Université de Lille et directeur de l'équipe BONUS (CRIStAL et Inria Lille), pour l'opportunité qu'ils m'ont donnée mais aussi pour leur encadrement, leurs conseils et encouragements tout au long de ce travail.

J'adresse des remerciements particuliers à Jan Gmys, Docteur et Ingénieur de recherche au Mésocentre de Calcul Scientifique Intensif de l'Université de Lille, pour sa gentillesse et ses conseils autant sur le plan scientifique et technique que pédagogique.

Je suis très reconnaissant envers les nombreuses personnes avec qui j'ai eu la chance de collaborer durant ces années. Je remercie notamment Guillaume Briffoteaux, Chercheur à l'Université de Newcastle (Australie) et anciennement Doctorant dans le service MARO pour nos nombreux échanges et nos productions scientifiques. Un bon nombre des études comprenant des cas d'application ont pu être réalisées grâce à l'intervention de François Vallée, Jean-François Toubeau, Pietro Favaro, respectivement Professeur, Chargé de Recherche et Doctorant au sein du service de Génie Electrique de l'Université de Mons; Édouard Rivière-Lorphèvre, François Ducobu, et Nithyaraaj Kugalur Palanisamy, respectivement Professeur, Professeur et Docteur au sein du service Génie Mécanique; Romain Ragonnet, Chercheur à la Monash University de Melbourne; je les remercie tous vivement.

Je tiens à remercier François Vallée, Professeur à l'Université de Mons, d'avoir accepté de présider le jury du comité d'accompagnement et de défense de thèse.

Je souhaite également remercier les rapporteurs Grégoire Danoy, Chercheur à l'Université du Luxembourg, et Pierre Sens, Professeur à Sorbonne Université, pour leur évaluation des travaux réalisés durant cette thèse.

Je remercie également tous les membres du comité d'accompagnement, qui ont su me guider jusqu'à l'aboutissement de ce travail.

J'adresse également mes remerciements à tous les membres du service MARO et plus largement à tous les collègues avec qui j'ai eu la chance de travailler, ainsi que les membres de l'équipe BONUS avec qui j'ai eu l'occasion d'échanger.

Enfin, je veux remercier ma famille et mes amis pour leur présence et leur soutien au quotidien. Je n'aurai pas la place de citer tout le monde ici, mais je suis certain qu'ils se reconnaîtront.

Abstract

The optimization of computationally expensive black-box problems is a challenge faced in many application fields. Those problems are characterized by the lack of information about their landscape and their computational cost. For instance, in engineering design, the objective function frequently results from complex numerical simulations and only its output is available. We investigate two major ways of dealing with such problems. First, we rely on Machine Learning to build surrogate models that approximate the expensive objective function at a lower computational cost. Second, we use parallel computing to reduce the computational burden of the optimization process. The major research question addressed in this thesis is how surrogate modeling and parallelism can help to efficiently and effectively sample the design space.

We distinguish two main approaches of using the surrogate model inside an optimization process. In Surrogate-Driven Optimization (SDO), the surrogate model actively drives the process through the definition of an Acquisition Function (AF) that evaluates the promisingness of a candidate decision. This is typically the case in Bayesian Optimization (BO), where Gaussian Process (GP) surrogate models are used. The Acquisition Process (AP) points out the most valuable candidates through the optimization of the AF. Alternatively, Surrogate-Assisted Optimization (SAO) uses the surrogate model to discard unpromising candidates and/or to partially replace the objective function. In SAO, the candidates are generated by external operators and filtered out using an Evolution Control (EC) based on the surrogate model.

In this thesis, we focus on the AP of BO Algorithms (BOAs) and its challenge of providing a valuable batch of candidates to be exactly evaluate in parallel. Firstly, the scalability of BO is limited when considering only the parallel evaluation of the candidates. Indeed, the sequential parts of the algorithm tend to be also computationally expensive. Secondly, the effectiveness of the batch acquisition is lesser compared to the sequential selection. An efficient use of parallel computing necessitates an efficient AP and smartly allocating the overall time budget to the AP (including the model fitting) and the simulations. We propose a new approach introducing parallel computing into the AP by leveraging space partitioning. From this, we derive two algorithms, namely Binary Space Partitioning Efficient Global Optimization (BSP-EGO) and Local models BSP-EGO (ℓBSP-EGO). The first one uses a global model to guide the optimization while the second one sets up multiple local models inside the sub-regions. The two developed algorithms are confronted with recent state-of-the-art BOAs using very different APs (*e.g.*, using trust regions, multiple AFs, etc.). We demonstrate the better scalability and batch effectiveness of the BSP-EGO-based algorithms compared to other BOAs.

We also compare the BO approach to other Surrogate-Based Optimization (SBO) algorithms, more precisely to Surrogate-Assisted Evolutionary Algorithms (SAEAs). These latter are usually more time-efficient since the acquisition of the candidates does not require an expensive surrogate model. The experimental protocol involves both benchmark functions and real-world engineering problems. It is designed to identify which approach is the most suitable depending on the operational constraints. The results indicate that BOAs are extremely sample-efficient, providing good outcomes with a few simulations. However, they are generally hampered by their expensive AP (including the model fitting). When the computational budget is higher, either because the simulator is computationally cheap, or because the time frame is large enough, SAEAs are to be preferred. We also demonstrate that both approaches can be combined into a hybrid algorithm benefiting the sample-efficiency of BO and the time-efficiency of SAO.

Résumé

L'optimisation de problèmes coûteux en calculs de type *boîte noire* est un défi rencontré dans de nombreux domaines. Ces problèmes sont caractérisés par le manque d'information dont on dispose à propos de leur nature, et par leur coût en calculs. Par exemple, en ingénierie, la fonction objectif résulte fréquemment de simulations numériques complexes dont seul le résultat final est accessible. Nous étudions deux moyens principaux permettant de traiter ces problèmes. Dans un premier temps, nous nous appuyons sur des modèles de substitution issus de l'Apprentissage Machine pour approximer la fonction objectif coûteuse en calculs par une alternative à moindre coût. Dans un second temps, nous utilisons le calcul parallèle pour accélérer le processus d'optimisation. La problématique majeure de cette thèse est d'identifier et de développer les méthodes d'optimisation combinant calcul parallèle et modèles de substitution les plus efficaces pour échantillonner l'espace de recherche.

On distingue principalement deux approches dans lesquelles l'optimisation est basée sur un modèle de substitution. L'optimisation *guidée* par modèle de substitution - Surrogate-Driven Optimization (SDO) - emploie le modèle pour guider l'optimisation à travers la création d'une fonction d'acquisition - Acquisition Function (AF). Cette dernière évalue la *valeur* d'un point candidat. L'Optimisation Bayésienne - Bayesian Optimization (BO) - en est un exemple notable dans lequel le modèle de substitution repose sur les Processus Gaussiens - Gaussian Process (GP). La fonction d'acquisition est optimisée pour déterminer le meilleur point à évaluer. Cette étape est appelée Processus d'Acquisition - Acquisition Process (AP). L'optimisation *assistée* par modèle de substitution - Surrogate-Assisted Optimization (SAO) - est une alternative dans laquelle le modèle est utilisé pour écarter les candidats les moins prometteurs et/ou pour remplacer partiellement la fonction objectif. Dans ces méthodes, les candidats sont générés par des opérateurs externes puis sont filtrés par le Contrôle d'Evolution - Evolution Control (EC) - qui s'appuie sur le modèle de substitution.

Dans cette thèse, nous nous intéressons particulièrement au processus d'acquisition des algorithmes d'optimisation bayésienne et à son défi qui consiste à sélectionner simultanément le meilleur ensemble de points à évaluer en parallèle. Premièrement, la scalabilité de l'optimisation bayésienne est limitée lorsque l'on ne considère le calcul parallèle que pour évaluer les candidats. En effet, les parties séquentielles de l'algorithme peuvent devenir coûteuses en temps également. Deuxièmement, la qualité de l'ensemble de points obtenus simultanément est moindre comparée à une sélection séquentielle. Un usage efficace des ressources de calcul nécessite un processus d'acquisition efficace et de qualité, et une allocation pertinente des moyens entre l'acquisition de nouveaux candidats et leur évaluation. Nous proposons une nouvelle approche qui introduit le calcul parallèle dans le processus d'acquisition grâce à un partitionnement de l'espace de recherche. Cette approche est déclinée en deux algorithmes, nommés BSP-EGO et ℓBSP-EGO. Le premier utilise un modèle de substitution global pour guider la recherche, tandis que le second s'appuie sur plusieurs sous-modèles propres à chaque sous-domaine. Ces deux algorithmes sont confrontés à plusieurs algorithmes d'optimisation bayésienne de l'état de l'art, qui utilisent des processus d'acquisition très différents. Nous démontrons une meilleure scalabilité et qualité des points acquis grâce aux méthodes développées, en comparaison avec l'état de l'art.

Nous comparons aussi ces methodes d'optimisation bayésienne à d'autres algorithmes basés sur les modèles de substitution, et plus précisément aux Algorithmes Evolutionnaires Assistés

par modèle de Substitution - Surrogate-Assisted Evolutionary Algorithm (SAEA). Ces derniers sont généralement plus efficaces en termes de temps d'exécution car l'acquisition des candidats ne nécessite pas de modèle de substitution coûteux en temps. Le protocole expérimental considère à la fois des fonctions benchmark et des problèmes réels d'ingénierie. Il est établi afin d'identifier l'approche la plus adéquate, compte tenu des contraintes opérationnelles. Les résultats indiquent que l'optimisation bayésienne est extrêmement efficace lorsque le budget est très serré, permettant de fournir une bonne solution en peu d'évaluations. Néanmoins lorsque le budget est plus important, soit parce que le coût en calcul du simulateur est plus faible, soit car le temps alloué est suffisamment grand, les algorithmes évolutionnaires assistés par modèle de substitution sont à privilégier. Nous démontrons également que les méthodes bayésiennes et évolutionnaires peuvent être combinées pour bénéficier des atouts de chacune des approches.

Contents

List of Acronyms

ℓBSP-EGO Local-model Binary Space Partitioning Efficient Global Optimization.

qEGO q-points Efficient Global Optimization.

qEI q-points Expected Improvement.

AF Acquisition Function.

AP Acquisition Process.

BO Bayesian Optimization.

BOA Bayesian Optimization Algorithm.

BSP-EGO Binary Space Partitioning Efficient Global Optimization.

CL Constant Liar.

EA Evolutionary Algorithm.

EC Evolution Control.

EGO Efficient Global Optimization.

EI Expected Improvement.

GA Genetic Algorithm.

GO Global Optimization.

GP Gaussian Process.

HPC High-Performance Computing.

KB Kriging Believer.

LCB Lower Confidence Bound.

LHS Latin Hypercube Sampling.

MACE Multi ACquisition Ensemble.

MIC-qEGO Multi-Infill Criteria qEGO.

MILP Mixed-Integer Linear Programming.

MLE Maximum Likelihood Estimation.

PBO Parallel Bayesian Optimization.

PHES Pumped Hydro-Energy Storage.

PI Probability of Improvement.

PSO Particle Swarm Optimization.

SaaEF Surrogate as an Evaluator and a Filter.

SaaF Surrogate as a Filter.

SAEA Surrogate-Assisted Evolutionary Algorithm.

SAGA Surrogate-Assisted Genetic Algorithm.

SAO Surrogate-Assisted Optimization.

SAPSO Surrogate-Assisted Particle Swarm Optimization.

SBO Surrogate-Based Optimization.

SDO Surrogate-Driven Optimization.

TBTC Tuberculosis Transmission Control.

TS Thompson Sampling.

TuRBO TrUst Region Bayesian Optimization.

UPHES Underground Pumped Hydro-Energy Storage.

General Introduction

Finding the optimal solution to a problem has always been a significant concern, particularly in recent times where optimization has taken an important place in decision-making processes. What is the best shape of an object for a given purpose? What is the best strategy to minimize the impact of a disease on a population? Or even what is the best bidding decision for an actor in the energy sector? To be optimized, the impact of the decision must be measured in some way, and this is where *numerical simulation* steps in. Often, the information about the quality of a decision comes with a cost that prevents from making many attempts. Actually, this cost is essentially a time cost and in the manuscript, costly and time-consuming are employed as synonyms. To tackle this kind of problem a popular choice is to resort to surrogate modeling to approximate the outcome of the costly simulator [1–3] and to parallel computing to execute several simulations at once and reduce the overall execution time.

Scope and Research Interest

In this thesis, we are interested in the synergy between the three following topics:

- Global Optimization of black-box functions
  Global Optimization (GO) consists in the minimization of a real-valued cost function f with respect to a vector of real values $\boldsymbol{x} \in \mathbb{R}^d$, d being the dimension of the considered problem. In many applications, $\boldsymbol{x}$ is restricted to a sub-domain of $\mathbb{R}^d$ which is denoted as Ω. In mathematical terms, we state this problem as:

  $$\min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x}) \tag{1}$$

  where f is defined as $f : \mathbb{R}^d \to \mathbb{R};\ \boldsymbol{x} \mapsto y = f(\boldsymbol{x})$. In our context, f is the time-consuming simulator mentioned earlier and the only information we dispose of is its output $y = f(x)$. The term *time-consuming* is rather ambiguous and will be discussed in the following. Nevertheless, the fact remains that in such a situation, we might resort to approximations of the objective function, namely surrogate models, or metamodels.

- Surrogate modeling
  A surrogate model is a Machine Learning model dedicated to the approximation of a function. It constitutes an alternative to the objective function we can have access to at (almost) no cost compared to the black-box objective function. In this thesis, we are mostly interested in the Gaussian Process- (GP-) based surrogate models [4]. Unlike many other surrogate models, GPs have the capability of providing a measure of uncertainty over the prediction of the objective value. We also refer to the prediction as the approximation, while the exact evaluation is called *simulation*.

- High-Performance Computing (HPC)
  Assuming the time-consuming simulation can be automated and performed autonomously without human intervention, coupled with surrogate modeling, parallel computing is an efficient asset in limiting the overall optimization time. Thus, the use of HPC platforms

such as Grid5000 [5] is a valuable advantage to address computationally intensive optimization tasks.

Many real-world optimization problems require the use of the three precedent research lines, including those considered in our works (energy storage, virus transmission control, etc.). Within the scope of this thesis, we focus on Parallel Bayesian Optimization (PBO). Indeed, BO Algorithms (BOAs) are particularly efficient in simulation-based optimization since they usually achieve good results using only a few simulations. BOAs manage to considerably reduce the number of calls to the expensive simulator by carefully choosing which point is to be evaluated. The process of choosing the next point to evaluate is called *Acquisition Process (AP)*, and the suggested point is referred to as the candidate. It relies on a surrogate model to approximate the objective function and then uses the model to build a utility function that guides the optimization process. GP models are generally assumed in BOAs and their purpose is to evaluate the benefit of a candidate of the search space and give an indication of whether or not it is worth sampling in the area.

Challenges and Objectives

To benefit from the computing power offered by HPC platforms, or even recent multi-core workstations or laptops, the AP selects at each iteration of the BOA a batch of candidates to be simulated instead of only one. However, this task is not straightforward since BO is an intrinsically sequential process [6, 7]. The algorithm relies on an Acquisition Function (AF) that evaluates the *promisingness* of a candidate. Many AFs for sequential BO have been defined in the literature [2, 8–10], and the correct choice remains sensitive since it has been observed that it depends on the objective function, or even on the stage of the optimization [11]. When it comes to providing a batch of candidates to be exactly evaluated in parallel, the problem is all the more complex. The batch acquisition incurs a significant overhead and the quality of the batch is largely reduced compared to sequential selection [6, 12–15]. In fact, the time of the batch acquisition might not be negligible compared to the simulation time. Increasing the batch size to exploit parallel computing implies managing larger sets of data which can become impractical considering that the GP modeling scales in $\mathcal{O}(n^3)$, where n is the size of the data set. The impact of increasing the amount of parallel resources is only scarcely investigated in the literature [6]. In addition, the dimension of the tackled problem impacts the choice or design of the algorithm (model, AF, etc.), and usually a large dimension implies dealing with more data, coming back to the previous point [3]. Consequently, the objectives of this thesis are the following:

- Analyze the state-of-the-art algorithms and identify their range of application and limitations on real-world applications as well as the lever we can act on for improvement;
- Develop a new approach accounting for the previously observed limitations and investigate its performances against the related state-of-the-art algorithms;
- Evaluate our BO methods and the state-of-the-art ones (not exclusively BOAs) on their ability to scale with the computational power, and in regards of the effectiveness of the batch selection;

- Derive some recommendations for the efficient design of PBO algorithms accounting for the budget in terms of simulations or total computing time, the dimension of the problem, and amount of computing resources.

Scientific Contributions

The main contributions of this doctoral thesis can be decomposed into three parts, following the structure of the present document. First, we **analyze the suitability of some state-of-the-art algorithms through different real-world test cases**. These latter include the optimization of a decision-making process in electrical engineering and medicine, and the calibration of the model's hyper-parameters in mechanical engineering. **We demonstrate the relevance of PBO on various real-world applications, but also emphasize strong limitations regarding the effectiveness of the parallel APs.** The simulators used in these applications have very different characteristics, in particular regarding their execution time. It is almost consensual that BO performs very well on computationally expensive black-box problems [6]. For instance, it is not refuted by the mechanical engineering application, referred to as the *inverse identification problem* in Section 2.1, where the well-known Efficient Global Optimization (EGO) (sequential) algorithm [1] is successfully applied. Actually, the simulator lasts about 40 minutes and is already exploiting parallel computing itself, making (non-parallel) BO perfectly fitted. We also demonstrate the good performance of PBO on real-world applications with moderately expensive simulations by improving the profit of a *virtual power plant optimal commitment problem* (see Section 2.2) using the parallel qEGO [16, 17] algorithm. Regarding the medical application referred to as the *tuberculosis transmission control problem* in Section 2.3, which is also a budget management problem with moderate time cost, we confront BOAs to Surrogate-Assisted Evolutionary Algorithm (SAEA) and emphasize the computational cost of the AP in the PBO framework. The dimension of those problems is at most 6, however, they have objective functions with different computational costs. A common assumption is that the surrogate part of the optimization is negligible with respect to the simulation time. Nevertheless, the range of applications of BO can be larger. For instance, the simulator associated with the virtual power plant optimal commitment problem is considered time-consuming mainly because operational constraints impose a short time frame for the optimization process.

In this context, the time dedicated to surrogate modeling can not be set aside. Indeed, in this situation most PBO algorithms are limited in terms of parallelization because of their time-consuming candidate selection, which can become excessive when a large batch of candidates is required. Adding a large batch of candidates at each cycle makes the size of the data set grow rapidly, increasing the time required for fitting the surrogate model. In addition to the poor scalability, the analysis points out a limited effectiveness of the batch acquisition (compared to sequential one) that decreases the efficiency of parallel algorithms.

Secondly, based on the latter observations, **we derive a new PBO approach reducing the overall execution time by leveraging design space decomposition. We use the decomposition paradigm to introduce parallelism and diversification into the AP.** From this approach, we decline two new algorithms using a recursive binary decomposition of the domain. They are named Binary Space Partitioning Efficient Global Optimization (BSP-EGO) and Local BSP-EGO (ℓBSP-EGO). BSP-EGO uses a global model, trained over the whole data,

to perform local APs in the sub-regions while ℓBSP-EGO uses a local model inside each sub-region that is trained on a subset of data. Both perform local searches for new candidates in distinct sub-regions of the search space. As a consequence, each local AP can be executed independently in parallel. Using parallel computing inside the AP offers a fast way of operating the batch querying strategy. As a consequence, the proposed algorithms are suitable for optimizing problems where the time budget is restricted. The two proposed algorithms are widely investigated and compared with existing state-of-the-art methods in Section 3. The test suite includes benchmark functions in dimensions 6 and 12, as well as a 12-dimensional real-world resource management problem from electrical engineering, named *Underground Pumped Hydro-Energy Storage (UPHES) management problem* hereafter. The experimental framework is designed to challenge the new algorithms with state-of-the-art methods, and analyze their strengths and weaknesses. The selected contestant algorithms employ various strategies, allowing us to examine their impact on the batch AP. Despite a faster candidate selection, BSP-EGO also faces the fast-increasing time cost of surrogate modeling. When it comes to ℓBSP-EGO, the local models are set up on subsets of data which makes the learning much faster. The fast candidate selection, the controlled learning time, and the enhanced intensification make ℓBSP-EGO one of the best-performing algorithms of the study. This algorithm shows improved performances compared to the state-of-the-art algorithms regarding benchmark functions in dimensions 6 and 12. TrUst Region Bayesian Optimization (TuRBO) [18], another BOA dealing with the search space also achieves good results, especially for 12-dimensional benchmark functions. This indicates the effectiveness of partitioning and reducing the search space in BO. Resorting to multiple AF also shows evident improvement compared to single AF methods. The investigation is amplified with the real-world UPHES management problem, involving 12 dimensions, where a multi-criteria AP performs best. The main drawback of this analysis is that it does not really account for the simulation time. Indeed, benchmark functions have a very low computational cost, making fast acquisition a huge advantage that might not generalize to real-world conditions. A legitimate question would be asking when each algorithm is more suited regarding the execution conditions.

In the last part of contributions of the thesis, we attempt to provide elements of answers to the previous question by better **characterizing the condition of applicability of BOAs with respect to the budget, the resources, etc. and in comparison with other Surrogate-Based Optimization (SBO) algorithms**. Consequently, **we propose an experimental set-up that accounts for the simulation time, the available computing resources, and the time budget.** Actually, depending on the time budget allocated to a problem, BOAs are not always the most profitable choice. For black-box problems, we can also rely on the widespread (surrogate-free) metaheuristics, such as Evolutionary Algorithms (EAs). Metaheuristics usually require a large number of simulations and thus are more suited to a computationally cheap simulator. Their application can be extended to moderately time-consuming simulators by integrating surrogate modeling inside the optimization process and spare simulator calls. This is typically the case in Surrogate-Assisted Optimization (SAO), where the surrogate model assists the pre-existing algorithm. As for BOAs, they are part of the Surrogate-Driven Optimization (SDO) family, where the surrogate model is used to build the AF. SDO algorithms put a larger computational effort in the candidate selection and are more fitted for computationally intensive simulators. Surrogate-free and SBO algorithms are then complementary in their application. Surrogate-driven algorithms, and more precisely BO ones, show large improvement with few simulations and are more suitable for time-consuming problems. However, it might be difficult

to select the most suitable one given a particular context without quantifying precisely what *time-consuming* means in the context of BO. In this analysis, we relate the operating conditions, namely the budget, the simulation time, and computing resources, with the appropriate choice of algorithm. We conduct an intensive investigation involving the best BOAs of this study which we challenge with recent SAEAs [19] and EAs to identify their domain of performance regarding the different contexts. **We observe a threshold from which SAEAs are outperforming all the BOAs, which can be used for selecting the best algorithm, but also to design hybrid ones. We propose a hybrid algorithm that switch from a BOA to a SAEA at some point defined by the threshold.** This hybrid strategy is investigated and compared with the best-performing methods of the test suite. It shows a good any-time performance by combining the benefits of SAEAs and BOAs. Finally, the best-performing algorithms are investigated on the Pumped Hydro-Energy Storage (PHES) problem involving 30 dimensions. We demonstrate that BOAs can be efficient in higher-dimensional problems, considering the appropriate acquisition strategy.

Along this work, the considered methods are investigated on both classical benchmark functions to conduct extensive experiments, but also on challenging engineering problems with tangible impact in their respective fields. From the analysis, we derive guidelines for developing efficient algorithms adapted to the considered problem, and perspectives for the improvement of existing algorithms.

Dissertation Structure

The manuscript is organized into two parts, containing two chapters each. Part I intends to give the necessary theoretical background for SBO, and provides a first analysis of existing related algorithms and their applications. Chapter 1 is dedicated to the state of the art in PBO. Important features of PBO are given, with a focus on surrogate modeling and the AP. It offers an overview of existing methods as well as a short theoretical background necessary to understand the lever on which we can act to improve the optimization. Three real-world applications are tackled in Chapter 2 using widespread algorithms. The algorithms are experimented on benchmark problems, but not always on real-world ones. The objective of this chapter is to get familiar with the concepts of BO and PBO, and to extract current limitations of the approaches in real-world conditions.

Part II, including Chapter 3 and Chapter 4, builds on the observations of the previous chapters to derive a novel BSP-based AP fitted for large-scale acquisition and improving the parallel efficiency of current BOAs. Chapter 3 presents the new AP that responds to the limitations pointed out above. The BSP-EGO and ℓBSP-EGO algorithms are detailed and confronted with state-of-the-art algorithms in a benchmark analysis as well as in real-world conditions on the UPHES problem. Finally, BOAs and SAEAs are opposed in Chapter 4 to identify the best choice regarding the application conditions. A hybrid algorithm is designed to combine the strengths of both classes and extend its applicability. The used algorithms are investigated for a higher dimensional PHES problem in the last section.

The manuscript ends with a concluding chapter summarizing the contributions and guidelines derived from this work. We also give some future research directions and outline ongoing works.

The supplementary material includes the lists of acronyms, figures, tables, and algorithms, and three appendices. The four lists are found on pages ix, I, III, and VII respectively. Appendix A provides some mathematical concepts used in BO. Appendix B gives additional information about multi-objective optimization. Finally, Appendix C reports additional results and information about the conducted benchmark analysis.

Each chapter is preceded by a short introduction and the list of our publications related to the chapter. The contributions of this thesis have been published in the following papers:

In academic journals:

- Briffoteaux, G., Gobert, M., Ragonnet, R., Gmys, J., *et al.* Parallel surrogate-assisted optimization: Batched Bayesian Neural Network-assisted GA versus q-EGO. *Swarm and Evolutionary Computation* **57,** 100717. ISSN: 2210-6502. http://www.sciencedirect.com/science/article/pii/S2210650220303709 (2020)

- Kugalur Palanisamy, N., Rivière Lorphèvre, E., Gobert, M., Briffoteaux, G., *et al.* Identification of the Parameter Values of the Constitutive and Friction Models in Machining Using EGO Algorithm: Application to Ti6Al4V. *Metals* **12.** ISSN: 2075-4701. https://www.mdpi.com/2075-4701/12/6/976 (2022)

- Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Batch Acquisition for Parallel Bayesian Optimization; Application to Hydro-Energy Storage Systems Scheduling. *Algorithms* **15.** ISSN: 1999-4893. https://www.mdpi.com/1999-4893/15/12/446 (2022)

- Ducobu, F., Kugalur Palanisamy, N., Briffoteaux, G., Gobert, M., *et al.* Identification of the Constitutive and Friction Models Parameters via a Multi-Objective Surrogate-Assisted Algorithm for the Modeling of Machining - Application to ALE orthogonal cutting of Ti6Al4V. *Journal of Manufacturing Science and Engineering*, 1–54. ISSN: 1087-1357. eprint: https://asmedigitalcollection.asme.org/manufacturingscience/article-pdf/doi/10.1115/1.4065223/7324066/manu-23-1749.pdf. https://doi.org/10.1115/1.4065223 (2024)

- Gobert, M., Briffoteaux, G., Gmys, J., Melab, N. & Tuyttens, D. Observations in Applying Bayesian versus Evolutionary approaches and their Hybrids in Parallel Time-constrained Optimization, ***Currently under review*** *in Engineering Applications of Artificial Intelligence*

- Favaro, P., Gobert, M. & Toubeau, J.-F. Multi-fidelity Optimization for Pumped Hydro Energy Storage Participating in Energy and Reserve Markets, ***Currently under review*** *in Applied Energy*

In conference proceedings, with peer review and oral presentation:

- Gobert, M., Gmys, J., Toubeau, J.-F., Vallée, F., Melab, N. & Tuyttens, D. Surrogate-Assisted Optimization for Multi-stage Optimal Scheduling of Virtual Power Plants. in *2019 International Conference on High Performance Computing Simulation (HPCS)* (2019), 113–120
- Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Adaptive Space Partitioning for Parallel Bayesian Optimization. in *HPCS 2020 - The 18th International Conference on High Performance Computing Simulation* (Barcelona / Virtual, Spain, 2021). https://hal.inria.fr/hal-03121209
- Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Parallel Bayesian Optimization for Optimal Scheduling of Underground Pumped Hydro-Energy Storage Systems. in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2022), 790–797

And as short papers or abstracts:

- Filipič, B., Depolli, M., Zupančič, J., Gmys, J., *et al.* ECG Simulator Tuning: A Parallel Multiobjective Optimization Approach. in *Proceedings OLA'2018 International Workshop on Optimization and Learning: Challenges and Applications* (2018), 25–28
- Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Towards Adaptive Space Partitioning for Large-scale Parallel Bayesian Optimization. in *OLA'2020 - International Conference on Optimization and Learning* (Cadix, Spain, 2020). https://hal.archives-ouvertes.fr/hal-02898960
- Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Space Partitioning with multiple models for Parallel Bayesian Optimization. in *OLA 2021 - Optimization and Learning Algorithm* (Sicilia / Virtual, Italy, 2021). https://hal.archives-ouvertes.fr/hal-03324642

Part I

Parallel Bayesian Optimization: Background and Preliminary Analysis

Chapter 1

Parallel Bayesian Optimization

This chapter is organized as follows. Section 1.1 presents Parallel Bayesian Optimization (PBO), which is part of the Surrogate-Based Optimization (SBO) framework, tackling Global Optimization (GO) problems. We first introduce the important concepts of black-box GO to progressively focus on Bayesian Optimization (BO) and its parallelization. In SBO, surrogate models are used to approximate the time-consuming objective function and partially replace it in order to limit the overall execution time of the algorithm. In this section, we clarify the distinction between Surrogate-Assisted Optimization (SAO) and Surrogate-Driven Optimization (SDO). The first one requires an external mechanism (such as evolutionary operators) to generate candidates that are evaluated either using the objective function or the surrogate model. As for the second, the model actively participates to the query of new candidates by the definition of an Acquisition Function (AF). The candidates are obtained by optimization of the latter, and not by external operator independent from the surrogate model. BO is part of the SDO framework, but uses probabilistic models such as Gaussian Process (GP) surrogate models that

provide an uncertainty in addition to a prediction. Then, we expose the challenges arising from the parallelization of BO.

The constitutive elements of a Bayesian Optimization Algorithm (BOA) are the surrogate model and the Acquisition Process (AP). Section 1.2 presents the necessary theoretical background for Gaussian Process (GP) surrogate models, which are predominantly used in BOAs. Section 1.3 presents a state-of-the-art in PBO and the current challenges regarding the acquisition of new candidates. A BOA is a sequential process where a candidate is found through an Acquisition Process (AP), exactly evaluated, and integrated into the data set. Once updated with the new information, the surrogate is used to perform a new AP, and the process is repeated until the budget runs out. The AP usually consists of the optimization of the chosen AF, which yields one candidate. However, in order to perform parallel evaluation of the objective function, the AP must provide a batch of candidates. One big challenge consists in providing a batch of candidates that brings as much information as if they were chosen sequentially. We refer to this quality as the batch effectiveness. Different strategies have been proposed, generalizing the single-point AFs to multi-point ones, resorting to single-point AF multiple times, or resorting to multiple single-point ones.

1.1 Introduction to Bayesian Optimization

To better understand Bayesian Optimization (BO), we need to take a step back and look at the more general definition of black-box Global Optimization (GO).

1.1.1 Black-Box Global Optimization

Black-box objective functions

The term *optimization* is employed in a very large spectrum of situations. As soon as we have an objective, we optimize our actions to get the best result, whatever *best* means for us. We often apply simple heuristic strategies, based on previous experiences or common knowledge to find our *optimal* solution to a problem. However, when the task becomes more complex, we need to define precisely what is the objective, what solutions are acceptable and what are the constraints.

Then, we first define what *best* means, this is the definition of the **objective function**, also called the cost or fitness function. It measures the value of a candidate solution, and allows one to choose which ones are good compared to others. Of course, for a given problem it exists different formulations of the objective function which can result in different optimal solutions. The candidate solutions are characterized by their **design variables**, *i.e.* the variables we are trying to tune to get the best possible outcome. In some situations, not all candidates are **admissible**. It often happens that **constraints** limit the design space and thus the range of the design variables. This aspect is only scarcely approached in the context of this work. In this thesis, we address only optimization problems that have a finite number of decision variables, are unconstrained (except for the design space), and target global solutions. They are often gathered under the term Global Optimization (GO).

The objective function is assumed black-box since we do not know its analytical expression. For any design vector $\boldsymbol{x}$, we can evaluate $y = f(\boldsymbol{x})$, and the couple $(\boldsymbol{x}, y)$ is the only available information we can extract from the objective function. Without any other information than the outcome, the objective function is generally called **black-box**. This scenario is often faced in engineering when the value to optimize results from numerical simulations, possibly noisy, so that the objective function is generally referred to as the **simulator**. The black-box framework is presented in Figure 1.1, with input $\boldsymbol{x}$ and output y as previously defined.

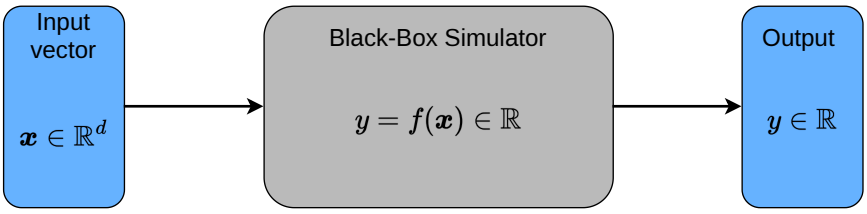


Figure 1.1: Black-box function representation

Exploration and exploitation

The way candidates are generated and selected is responsible for the fundamental concepts of **exploration** and **exploitation** (or diversification and intensification respectively). Exploration favors the introduction of very different possible solutions (combinations of decision variables, marked with crosses in Figure 1.2) while exploitation refines the existing ones. The whole optimization process is a trade-off between exploration and exploitation. Figure 1.2 succinctly illustrates this process. Let us assume the minimization of the objective function $f : \mathbb{R} \to \mathbb{R}$. The red ones mark the points for which we know the exact objective values, and we want to optimize the unknown function given these points. We have a choice between exploring the search space by evaluating, for example, the points marked with blue crosses, or exploiting the region we know to be good (in terms of objective values) by evaluating the points located at the green crosses.

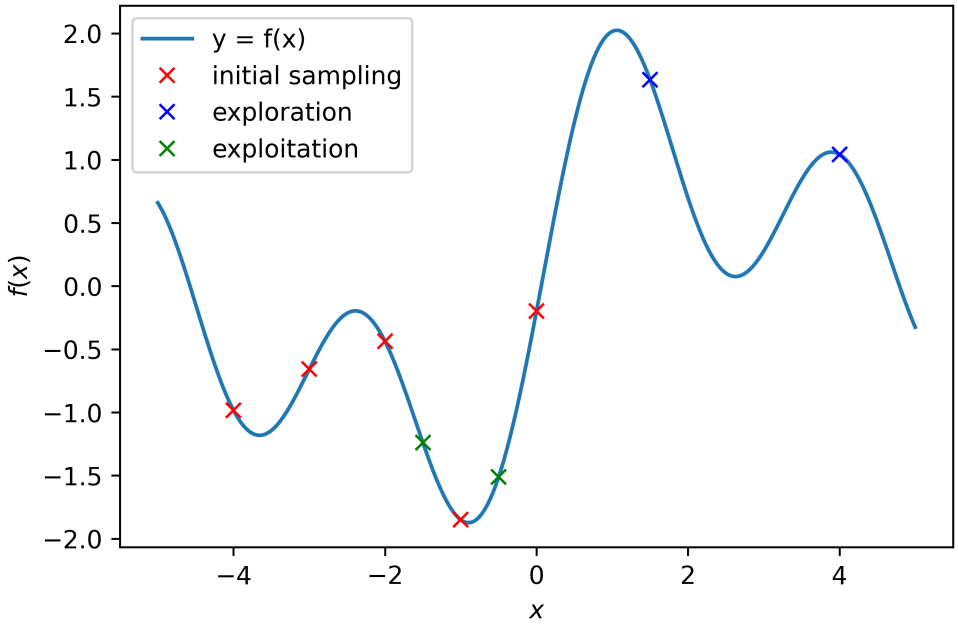


Figure 1.2: Illustration of the exploration and exploitation processes

Based on the different strategies to which we will come later, a point (or a batch of points) is selected to be evaluated hoping for an improved objective value. The optimization process consists of a succession of three steps: generate the candidate points, evaluate them, and update the best output. These steps are repeated until a stopping criterion is satisfied.

Global optimization

The classical approach in GO considers the minimization of a real-valued cost function f with respect to a vector of design variables $\boldsymbol{x} \in \mathbb{R}^d$, where d is the dimension. The design vector $\boldsymbol{x}$ is restricted to a sub-domain of $\mathbb{R}^d$, denoted as Ω. Unless explicitly mentioned, minimization is assumed all along this manuscript.

The optimization consists of choosing at each iteration some candidate points, that w e will denote $\boldsymbol{x}_{new}$, that might be better than the actual known minimum (since we consider minimization). The way of choosing the new points to be evaluated is called **Acquisition Process (AP)** or equivalently **acquisition strategy** and is subject to the nature of the objective function. To deal with black-box objective functions, we might resort to heuristics or metaheuristics in order to choose which points to evaluate (with the simulator) and lead the optimization process through the exploration and exploitation of the design space. Metaheuristics will be useful all along this manuscript, however since the methods are only used and not directly related to the scope of this thesis, only the used methods will be described when needed. We refer to [32] for exhaustive information on metaheuristics.

Even though it is a widespread approach for solving black-box optimization problems, it often requires an extremely high number of simulations to get a good result. Hence, it might not be suited when we deal with a time-consuming objective function. The meaning of *time-consuming* will be further discussed in Section 2.

1.1.2 Surrogate-Based Optimization

Surrogate modeling for optimization

A **surrogate model** is a predictive model which, for any location $\boldsymbol{x} \in \mathbb{R}^d$, returns $\hat{y} \in \mathbb{R}$. Other denominations are often used and will be used indifferently such as metamodel or simply model. In the following, the notation $\mathcal{M}$ is used to refer to unspecified model.

The surrogate model is designed to approximate the output $y \in \mathbb{R}$ of this function in order to substitute the costly objective function f by a prediction $\hat{y}$ that can be obtained at a much lower computational cost. Figure 1.3 illustrates the setting up of a surrogate model alongside the real objective function, providing a prediction $\hat{y}$ of the candidate $\boldsymbol{x}$, which is hopefully close to the output of the simulator ($|\hat{y} - y| < \epsilon$).

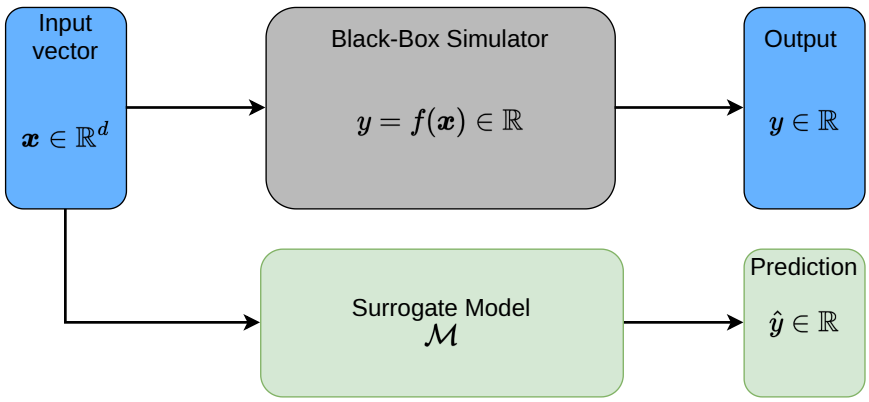


Figure 1.3: Surrogate model principle

In order to reduce the number of calls to the objective function and avoid its associated computational cost, one may choose to substitute the exact evaluation by a prediction. The surrogate model will be called instead and the cost of the real objective function evaluation is

spared. In this way, optimization can be performed in a reduced time. As the optimization is based on the surrogate model, it is called Surrogate-Based Optimization (SBO). Of course, not all evaluations will be predictions. Different approaches exist to integrate surrogate modeling into the optimization process. In the following, we refer to the output of the surrogate model as the prediction or the approximation, and to the objective function as the exact evaluation or simulation.

Integrating surrogate models into the optimization loops

Strategies for integrating the surrogate models into the optimization fall into two main families:

- **Surrogate-Assisted Optimization (SAO)**: the surrogate model replaces (at least partially) the simulator, and classical optimization (*e.g.* metaheuristic such as an Evolutionary Algorithm (EA)) is performed. In that case, we have to make a choice whether to use the surrogate model or the real objective function. This process is illustrated in Figure 1.4. This is also referred to as the fitness replacement in evolutionary computation literature [33, 34]. The initial state represents the set of exactly evaluated points, prior to the beginning of the optimization loop, which starts with the creation of a metamodel. Afterward, the algorithm proposes one or several candidate point(s) for evaluation. Then, two decisions are considered: (i) filter the candidates according to the model and exactly evaluate the most valuable ones; (ii) decide whether or not to trust the surrogate model for the evaluation of each candidate. In the first case, referred to as Surrogate as a Filter (SaaF), only exactly evaluated candidates supplement the data set. In the second case, referred to as Surrogate as an Evaluator and a Filter (SaaEF), both simulated and predicted candidates can join the data set. The obtained data is integrated in the total information, and if the stopping criterion is not met, the loop starts over. Strategies regarding the decision to call the simulator or the surrogate model are referred to as the Evolution Control (EC). This will be shortly addressed in this work, but we refer to [19] for detailed information about EC and the candidate generation process.
- **Surrogate-Driven Optimization (SDO)**: the surrogate model is the driving force of the algorithm. It does not replace the simulator but is exclusively used to propose new candidates to the simulator, and only the most valuable one(s) is/are exactly evaluated. The value of a candidate is assessed with an Acquisition Function (AF), and the selection process (based on the AF) is called the Acquisition Process (AP). Usually, the AP is also an optimization problem, sometimes called inner optimization in opposition to the outer optimization, which is the initial problem. The SDO loop is detailed in Figure 1.5, where α denotes the AF. A model is created based on initial data and is used in the AP to propose valuable candidates. In that sense, the optimization is not only assisted but *driven* by the surrogate model. Then the selected candidate(s) is/are exactly evaluated before being integrated into the data set. The meaning of *valuable* needs to be clarified, and will be in Section 1.3.

In both cases, the main objective of using a surrogate model is to reduce the computational burden associated to simulation by reducing the number of exact evaluations. Surrogate modeling takes an important part of this work. More detailed information is given in Section 1.2

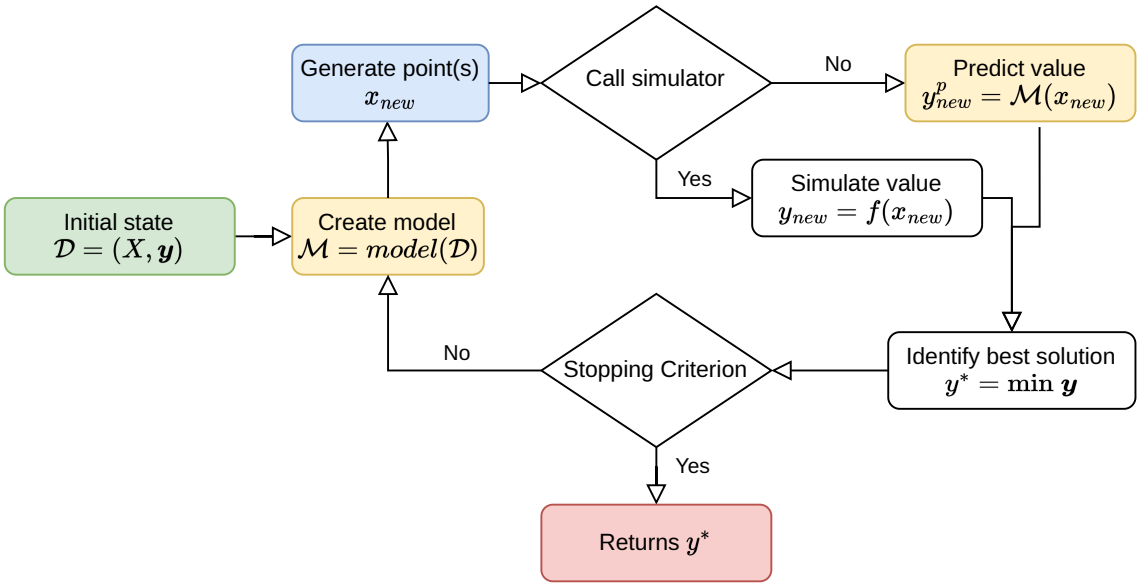


Figure 1.4: Surrogate-Assisted Optimization loop

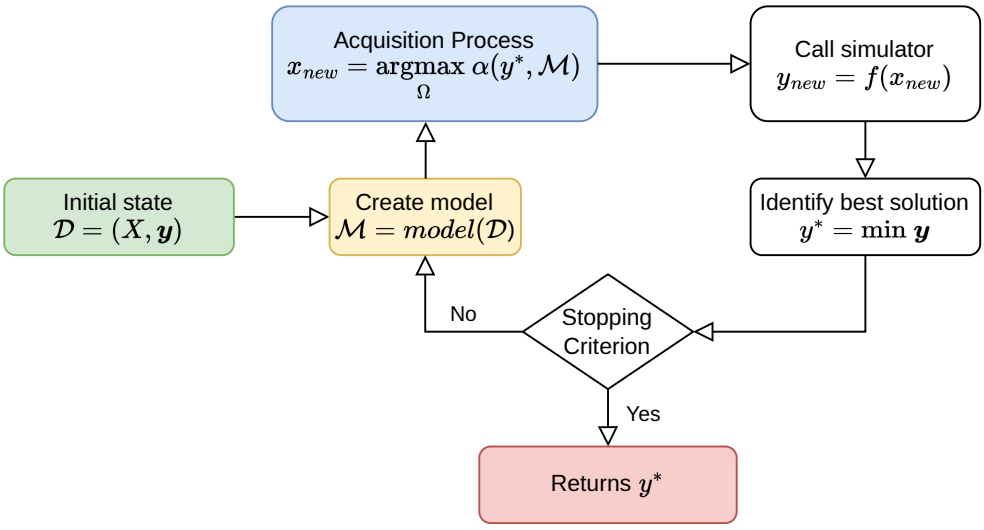


Figure 1.5: Surrogate-Driven Optimization loop

to define more precisely what is a surrogate model and what kind of surrogate models we are interested in.

1.1.3 Bayesian Optimization

Bayesian Optimization (BO) is an optimization approach of the SDO family and thus operates as previously explained in Figure 1.5. The distinctive aspect of BO is the model it uses for its AP: as the *Bayesian* term stands for, BO uses Bayesian models, which are commonly Gaussian Process (GP) regression models. A substantial feature of GP compared to non-bayesian models is the ability to provide a measure of uncertainty around the prediction of the surrogate model. This uncertainty is extremely valuable to assess the reliability of the predictive model, and consequently, it is an insightful information that will be used in the AP.

As in SDO, the general idea of BO is to rely on a figure of merit (*i.e.* the AF), that provides an indication of how desirable it is to sample a location $\boldsymbol{x}$. The approach has been used for a few decades and has proved to be very efficient in time-consuming black-box optimization problems [2], [8], [1]. Basically, BO consists of a succession of what we call **cycles** in which a metamodel - almost exclusively a GP model - is fitted with the data set (denoted as $\mathcal{D}$) in order to search for the best point to exactly evaluate next. This/these point(s) is/are chosen among all candidates according to its/their AF value, the selection of candidates is then an optimization problem itself with the objective of optimizing the AF value.

The pseudo-code of the standard BO loop is shown in Algorithm 1. Based on the known data $\mathcal{D}$, the model is trained to be used in the AP displayed in line 7. Afterward, the retained candidate(s) is/are exactly evaluated using the simulator, and integrated into $\mathcal{D}$. Algorithm 1

Algorithm 1 Bayesian Optimization

1: **Input**
2: f: objective function
3: Ω: design space
4: $\mathcal{D} = (X, \boldsymbol{y})$: known data
5: **while** budget available **do**
6: $\mathcal{M} = \mathcal{GP}(\mathcal{D})$
7: $\boldsymbol{x}_{new} = argmax_{\mathcal{D}}(\alpha(\boldsymbol{x}))$
8: $y_{new} = f(\boldsymbol{x}_{new})$
9: $\mathcal{D} = \mathcal{D} \cup (\boldsymbol{x}_{new}, y_{new})$
10: **end while**
11: **return** $\min_y \mathcal{D}$

introduces the notion of budget in line 5. The budget is defined by the user according to the operational constraint. Usually, the budget is defined as the total number of simulations, or, less frequently, as the total time dedicated to the optimization. At the end, the algorithm returns the best-found decision according to the objective.

1.1.4 Parallel Computing in Bayesian Optimization

Parallel computing is a type of computation that allows one to perform different computing tasks simultaneously. More generally speaking, High-Performance Computing (HPC) includes all techniques to optimize computer codes, and mostly minimize their execution time (wall-clock time). It includes parallel computing, vectorization, cache management, etc. It is a powerful advantage in optimization that allows to perform the optimization in a reduced wall-time by operating different parts of the algorithm in parallel. This is especially relevant in case the budget is a fixed wall-clock time. Usual parallelization techniques include partitioning the design space, multi-start algorithms, etc. In BO, the simulation is assumed to be black-box, so no work can be done at this level regarding implementation. In addition, the objective function evaluation is considered time-consuming. Regarding the latter aspect, the simulation time is generally dominant (in the mathematical meaning) compared to the remaining of the optimization process. With this assumption, the main profitability of parallel computing is the parallelization of the objective function evaluations.

More precisely, in PBO, the AP provides a batch of candidates that will be evaluated in parallel and integrated into the data set. Basically, the **master** node yields all the tasks to perform (*i.e.* all the candidates to evaluate) and distributes them to the **worker units**. This represents the basic master-worker parallel scheme of BO. Even though less profitable in the context of expensive objective functions, the parallelization could be applied to other parts of the algorithm. For instance, the optimization of the AF could be executed in parallel, or we could fit several surrogate models in parallel by creating subsets of data. Actually, we will see in Chapter 3 that parallel computing can be efficiently applied inside the AP, reducing the overall execution time.

In the parallel setting, we evaluate the performance of an algorithm in terms of **scalability** and the **batch effectiveness**. The batch effectiveness refers to the ability to preserve the performance (in terms of solution quality) of the sequential version while being executed in parallel. In BO, it consists in having the same effectiveness in terms of quality of objective values for q points selected sequentially than q ones selected simultaneously. Obviously, this is quite difficult to achieve since the q points are chosen with less information. Strictly speaking, scalability measures the ability to effectively using an increasing number of processing units, and thus to speed-up the resolution of a problem of fixed size. Equivalently, scalability measures the ability to perform more evaluations with more computing resources. Assuming k performed evaluations in one computing core, the best we can hope for with n_{cores} computing cores is to perform $k \times n_{cores}$ evaluations. It somehow measures the non-reducible part of the algorithm (*i.e.* the sequential part).

1.2 Surrogate Modeling for Bayesian Optimization

This section addresses the construction of surrogate models applicable to SBO. We present the GP models which are primarily used in BO. For the sake of clarity, let us set some notations first. Bold type is used for vectors and upper-case symbols for matrices or random variables. In particular, a point is denoted as $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, where d is the dimension of the

objective function f. The matrix of n points is denoted $X = (\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)})$ and $\boldsymbol{y} = f(X) = \big(f(\boldsymbol{x}^{(1)}), \dots, f(\boldsymbol{x}^{(n)})\big) = (y^{(1)}, \dots, y^{(n)})$. We call an observation the result of the simulator at a certain location, which gives the couple $(\boldsymbol{x}, y = f(\boldsymbol{x}))$, and the set of locations and observations $\mathcal{D} = (X, \boldsymbol{y})$ is referred to as the data set.

1.2.1 Gaussian Process Regression

Talking about Gaussian Process (GP), we often assume a Bayesian point of view which is characterized by the use of the Bayes rule but mostly by the definition of a prior probability distribution over an event. Given relevant data (or evidence), the prior is updated to give the posterior distribution. According to Rasmussen and Williams in *Gaussian Processes for Machine Learning* [4], the prior is taken to represent our prior beliefs over the kind of functions we expect to observe, before seeing any data.

Bayesian Linear Regression

Assuming the standard linear model as stated in [4], the observation y is a function of the decision vector $\boldsymbol{x}$ plus a small perturbation denoted ϵ:

$$y = \boldsymbol{a}^T \boldsymbol{x} + \epsilon \tag{1.1}$$

Learning the $\boldsymbol{a}$ parameters will then allow us to predict $y(\boldsymbol{x}^*) = \boldsymbol{a}^T \boldsymbol{x}^*$. The observation/output value is often considered noisy, hence the ϵ error term which is assumed Gaussian: $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Placing a Gaussian prior over the weights allows us to derive a posterior probability distribution over them by including the observations in the model. So let us assume:

$$\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_p) \tag{1.2}$$

To build the prediction at an unknown location, we need to *learn* the weights. More precisely, we compute the posterior distribution over the weights, according to the data and our prior distribution. Using Bayes Theorem the posterior writes:

$$p\left(\boldsymbol{a}|X, \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y}|X, \boldsymbol{a}\right) p\left(\boldsymbol{a}\right)}{p\left(\boldsymbol{y}|X\right)} \tag{1.3}$$

where $p\left(\boldsymbol{y}|X, \boldsymbol{a}\right)$ is the likelihood, $p(\boldsymbol{y}|X)$ is the model evidence, $p(\boldsymbol{a})$ the prior (also called normalizing constant), and $p\left(\boldsymbol{a}|X, \boldsymbol{y}\right)$ the posterior. The X conditioning is omitted in $p(\boldsymbol{a})$ since the prior is independent of the inputs.

Also, the model evidence can be marginalized over the weights to give the marginal likelihood $\int_{\boldsymbol{a}} p\left(\boldsymbol{y}|X,\ \boldsymbol{a}\right) p(\boldsymbol{a}) d\boldsymbol{a}$, and finally we have:

$$p\left(\boldsymbol{a}|X, \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y}|X, \boldsymbol{a}\right) p\left(\boldsymbol{a}\right)}{\int_{\boldsymbol{a}} p\left(\boldsymbol{y}|X,\ \boldsymbol{a}\right) p(\boldsymbol{a}) d\boldsymbol{a}} \tag{1.4}$$

Equation 1.3 can be stated as :

$$posterior = \frac{likelihood \times prior}{marginal\ likelihood} \tag{1.5}$$

The input data $\boldsymbol{x}$ may be projected into a feature space with the help of a set of basis functions ϕ_i, $i \in \{1, \dots, L\}$ and we denote as Φ the projected vector. Using the same notation, we can simply replace X by Φ in Equation 1.3.

With the Gaussian assumption, the posterior will also be Gaussian. According to Equation 1.3 the posterior is proportional to the likelihood times the prior, which is proportional to:

$$p(\boldsymbol{a}|X, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y} - X^T\boldsymbol{a}\right)^T\left(\boldsymbol{y} - X^T\boldsymbol{a}\right)\right)\exp\left(-\frac{1}{2}\boldsymbol{a}^T\Sigma_p^{-1}\boldsymbol{a}\right) \tag{1.6}$$

by keeping only the terms depending on $\boldsymbol{a}$ in the normal distribution (Definition A.1.2).

It can be shown[1] that:

$$p(\boldsymbol{a}|X, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{a} - \bar{\boldsymbol{a}})^T\left(\frac{1}{\sigma^2}XX^T + \Sigma^{-1}\right)(\boldsymbol{a} - \bar{\boldsymbol{a}})\right) \tag{1.7}$$

and we recognize the form of a Gaussian distribution with mean $\bar{\boldsymbol{a}} = \frac{1}{\sigma^2}A^{-1}X\boldsymbol{y}$ and covariance matrix A^{-1}, where $A = \left(\frac{1}{\sigma^2}XX^T + \Sigma^{-1}\right)$. This means that:

$$\boldsymbol{a}|X, \boldsymbol{y} \sim \mathcal{N}\left(\frac{1}{\sigma^2}A^{-1}X\boldsymbol{y}, A^{-1}\right) \tag{1.8}$$

The posterior distribution of the weights expresses their probability in light of the observations and allows to make predictions by marginalizing over the posterior. The predictive distribution of a given location $\boldsymbol{x}^*$ is given by $y^*|X, \boldsymbol{y}, \boldsymbol{x}^*$. Marginalizing over the weights gives the predictive distribution of a design point $\boldsymbol{x}^*$ which writes:

$$p(y^*|X, \boldsymbol{y}, \boldsymbol{x}^*) = \int_{\boldsymbol{a}} p(y^*|\boldsymbol{a}\boldsymbol{x}^*)\, p(\boldsymbol{a}|X, \boldsymbol{y})\, d\boldsymbol{a} \tag{1.9}$$

and is also Gaussian.

Using the form of the linear predictor $y^* = \boldsymbol{a}^T\boldsymbol{x}^*$ and the posterior distribution of $\boldsymbol{a}$ (Equation 1.8), by the linear transformation rule on the normal distribution (A.1.2), we can derive the law of the predictive distribution which writes:

$$y^*|X, \boldsymbol{y}, \boldsymbol{x}^* \sim \mathcal{N}\left(\frac{1}{\sigma^2}\boldsymbol{x}^{*T}A^{-1}X\boldsymbol{y}, \boldsymbol{x}^{*T}A^{-1}\boldsymbol{x}^*\right) \tag{1.10}$$

One can decide to project $\boldsymbol{x}$ into a feature space using basis functions $\phi_i(\boldsymbol{x}), \cdots, \phi_L(\boldsymbol{x}) = \boldsymbol{\phi}$. The predictive law becomes:

[1] Demonstration in Annex. A.2

$$y^*|\Phi, \boldsymbol{y}, \boldsymbol{x}^* \sim \mathcal{N}\left(\frac{1}{\sigma^2}\boldsymbol{\phi}^{*T}A^{-1}\Phi\boldsymbol{y}, \boldsymbol{\phi}^{*T}A^{-1}\boldsymbol{\phi}^*\right) \tag{1.11}$$

by simply replacing X by Φ. Furthermore, using $K = \Phi^T\Sigma\Phi$ it can be shown[2] that Equation 1.9 can be equivalently written as:

$$\begin{aligned} y^*|X, \boldsymbol{y}, \boldsymbol{x}^* \sim \mathcal{N}(&\boldsymbol{\phi}^{*T}\Sigma\Phi(K+\sigma^2 I)^{-1}\boldsymbol{y}, \\ &\boldsymbol{\phi}^{*T}\Sigma\boldsymbol{\phi}^* - \boldsymbol{\phi}^{*T}\Sigma\Phi(K+\sigma^2 I)^{-1}\Phi^T\Sigma\boldsymbol{\phi}^*). \end{aligned} \tag{1.12}$$

With $y = \boldsymbol{a}^T\boldsymbol{\phi}$, as y is assumed of mean zero : $\mathbb{E}[y] = 0$ and $\mathbb{C}ov(y_i, y_j) = \boldsymbol{\phi}_i^T\mathbb{E}[\boldsymbol{a}\boldsymbol{a}^T]\boldsymbol{\phi}_j = \left(\boldsymbol{\phi}^T\Sigma\boldsymbol{\phi}\right)_{i,j}$ (see A.1.1). Using $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{\phi}^{(i)}\Sigma\boldsymbol{\phi}^{(j)}$, and $K = \left(k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})\right)_{i,j\in\{1,\dots,n\}}$ the covariance function defined by the covariance kernel k.

Re-writing Equation 1.12 with this notations:

$$\begin{aligned} y^*|X, \boldsymbol{y}, \boldsymbol{x}^* \sim \mathcal{N}(&\boldsymbol{k}^{*T}(K+\sigma^2 I)^{-1}\boldsymbol{y}, \\ &k(0) - \boldsymbol{k}^{*T}(K+\sigma^2 I)^{-1}\boldsymbol{k}^*) \end{aligned} \tag{1.13}$$

The K matrix is known as the covariance matrix, or the Gram matrix.

Distribution over the functions

Similar results to the previous section can be obtained by considering directly the functions as a collection of Gaussian processes.

A GP is defined by its mean function and covariance function:

$$\begin{aligned} m(\boldsymbol{x}) &= \mathbb{E}[f(\boldsymbol{x})] \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))] \end{aligned} \tag{1.14}$$

with

$$f(\boldsymbol{x}) \sim \mathcal{N}\left(m(\cdot), k(\cdot,\cdot)\right). \tag{1.15}$$

Let us consider first that f is a zero mean linear predictor, similarly to Equation 1.1: $f = \boldsymbol{a}^T\boldsymbol{\phi}$. With the previously defined notations we have:

$$\begin{aligned} \mathbb{E}[f(\boldsymbol{x})] &= \boldsymbol{\phi}(\boldsymbol{x})^T\mathbb{E}[\boldsymbol{a}] = 0 \\ \mathbb{E}[f(\boldsymbol{x})f(\boldsymbol{x}')] &= \boldsymbol{\phi}(\boldsymbol{x})^T\mathbb{E}[\boldsymbol{a}\boldsymbol{a}^T]\boldsymbol{\phi}(\boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T\Sigma\boldsymbol{\phi}(\boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}'). \end{aligned} \tag{1.16}$$

[2] Demonstration in Annex. A.2.2

Kriging and Best Linear Unbiased Predictor

Kriging models are generally equivalent to GP and both often designate the same surrogate model. The first step of Kriging is inspired by the mining industry [35] and in particular by Danie G. Krige and his work on distance-weighted average gold grades at the Witwatersrand reef complex in South Africa [36]. Kriging is named after him, considering he is the pioneer of the method. Other occurrences of similar works are found in the literature, for example in forestry (B. Matèrn [37]) and meteorology (L.S. Gandin [38]). The theory is formalized later by Georges Matheron in *Traité de géostatistique appliquée* [39] (1962).

The common trait of the different works is the idea of spatial correlation and the fact that observations are more likely to be similar (correlated) if they are geographically close. Under this formalism, the Kriging predictor takes the form of the Best Unbiased Linear Prediction - see *Design and Analysis of Computer Experiments*, by Santner, Williams, and Notz [40]. The minimization of the mean square error implied by the linear predictor also requires to determine the covariance matrix K, and results in a formulation similar to Equation 1.12.

1.2.2 The Covariance Kernel

As highlighted in the previous section, the covariance kernel is an important hyper-parameter to build our regression model. It is usually chosen among the following usual kernels:

- Gaussian :
$$\kappa(x, y) = \exp\left(-\frac{(x-y)^2}{\theta^2}\right) \tag{1.17}$$
- Matern $\frac{5}{2}$:
$$\kappa(x, y) = \left(1 + \frac{\sqrt{5}|x-y|}{\theta} + \frac{5(x-y)^2}{3\theta^2}\right)\exp\left(-\frac{\sqrt{5}|x-y|}{\theta}\right) \tag{1.18}$$
- Matern $\frac{3}{2}$:
$$\kappa(x, y) = \left(1 + \frac{\sqrt{3}|x-y|}{\theta}\right)\exp\left(-\frac{\sqrt{3}|x-y|}{\theta}\right) \tag{1.19}$$
- Exponential :
$$\kappa(x, y) = \exp\left(-\frac{|x-y|}{\theta}\right) \tag{1.20}$$

Multi-dimensional case:
$$k(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{d} \kappa(x_i, y_i, \theta_i) \tag{1.21}$$

The parameters $\boldsymbol{\theta}$ are referred to as the characteristic length scales [4].

A lot of more kernels can be defined and created using known kernels, we refer to the works of N. Durrande [41] [42] and D. Duvenaud [43][44] to see how to create kernels and choose them for specific purposes. Although mixing kernels is possible and can be useful in some contexts (*e.g.*, additive kernels [44]), it is not explored in this work.

Maximizing the likelihood

The covariance kernel is a parametric function that needs to be fit to the data. A popular method is the Maximum Likelihood Estimation (MLE) of the parameters which consists of finding the *most probable* parameters given the observation. Assuming the parameters to be fit are $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)$. The likelihood is a function of the parameters to optimize, that expresses the probability of the observation under the parameter's assumptions. In our case:

$$L(\boldsymbol{\theta}) = p(\boldsymbol{y}|X, \boldsymbol{\theta})$$

Since the likelihood often takes the form of a product, it is common to use the log-likelihood, which preserves the maximum of the likelihood function. Using the Gaussian hypothesis $\boldsymbol{y} \sim \mathcal{N}(0, K)$, the log-likelihood writes:

$$log\left(p(\boldsymbol{y}|X, \boldsymbol{\theta})\right) = -\frac{1}{2}\boldsymbol{y}^T K^{-1}\boldsymbol{y} - \frac{1}{2}\log\left(|K|\right) - \frac{n}{2}\log(2\pi) \tag{1.22}$$

The MLE consists in finding the $\boldsymbol{\theta}$ parameters that maximize Equation 1.22. It can be done by finding the derivatives of Equation 1.22 with respect to θ_i to use gradient-based routines. The complexity of the parameter fitting through MLE is dominated by the inversion of K which time is $\mathcal{O}(n^3)$, where n is the matrix size (*i.e.*, the number of observations). Consequently, once K^{-1} is computed, the remaining part requires only time $\mathcal{O}(n^2)$ per parameter [4], so a gradient-based optimizer is well suited. As an example, the `L-BFGS-B` algorithm [45] might be applied.

The second term of Equation 1.22 is sometimes called the complexity penalty and its optimization favors less complex models. The marginal likelihood automatically incorporates a trade-off between model fit and complexity. A precise fit to the data can be achieved by reducing the length scale however the marginal likelihood does not favour this [4]. The Bayesian treatment can be extended to other hyper-parameters of the GP models, including for example the noise σ or the parameters of the basis functions $\boldsymbol{a}$.

1.2.3 Considerations from some Observations on GPs

Even though the investigation and contributions reported in this manuscript do not concern the theory of GPs, it is important to understand the different aspects of the surrogate models on which we can act.

Regarding the hypothesis of the model

- The zero or constant mean assumption (most common) is not a drastic limitation since the mean of the posterior distribution is not confined to be constant [4] as shown in Figure 1.6. The latter also illustrates different samples drawn from the prior and posterior

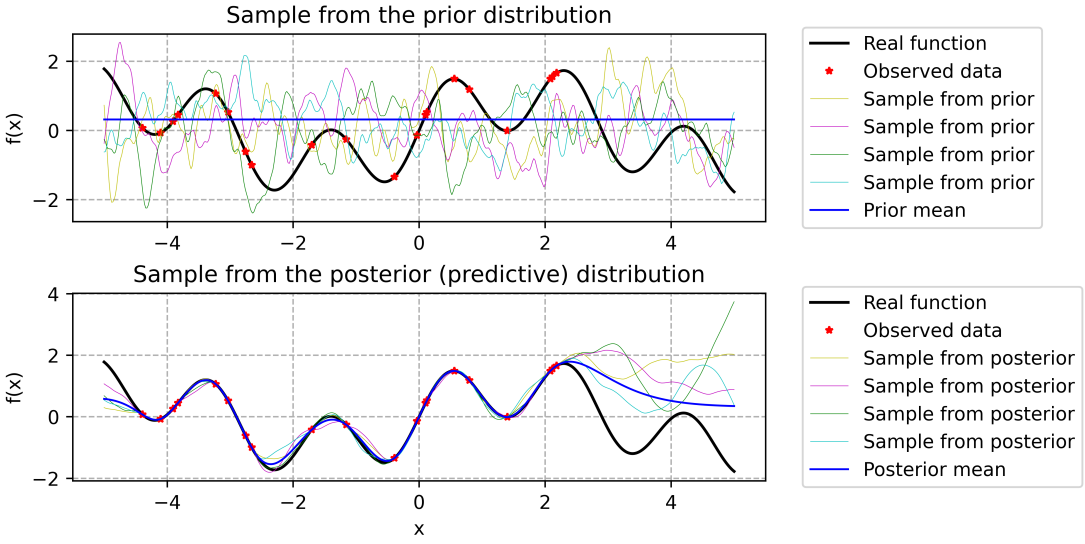


Figure 1.6: The posterior distribution is non constant even with constant mean prior.

distributions, which are *possible* observations regarding the assumptions. The samples from the prior are with constant mean, and the samples from the posterior are conditioned by the data. Hence, the posterior provides suitable observations close to the data.

- The observations close to the location of the point being predicted have more impact on the prediction.

- Covariance matrices and kernels strongly impact the models and the kernel is chosen as a hyper-parameter. However, except for specific purposes, the choice of the kernel might not be excessively important in the optimization context [46].

Numerical considerations

- Assuming noisy observations even for a deterministic simulation adds a hyper-parameter to optimize but it is necessary in case of computer simulation with no way to assert the stationarity. Numerically, it reinforces the diagonal of the covariance matrix and limits the bad conditioning of the matrix [47]. Figure 1.7 illustrates the numerical inaccuracy of the posterior distribution (the prediction) resulting from noisy observations. The blue curve results from a model that considers no noise, while the green one is built using a noise parameter estimated in the model. Without noise, the model interpolates the observations causing high variations in some situations. The addition of a noise hyper-parameter helps to deal with this phenomenon, even with noise-free observations.

- The noise estimation also improves the variance estimation. Figure 1.8 shows 100 samples from the posterior distribution built with and without noise estimation. The blue

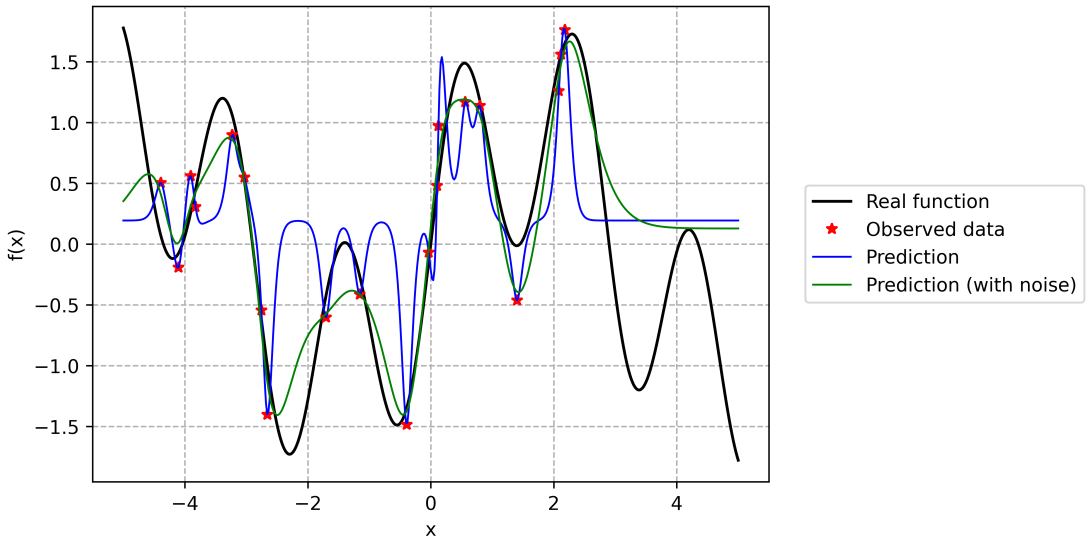


Figure 1.7: Prediction of a noisy objective with and without noise estimation in the model

background indicates a 2σ confidence interval ($\approx 95\%$) for the prediction. We can see that the variance is 0 at observed points in the top graph (without noise), while the prediction and variance are quite high in non-sampled areas. The prediction and variance of the bottom graph (with noise estimation) seem much more accurate.

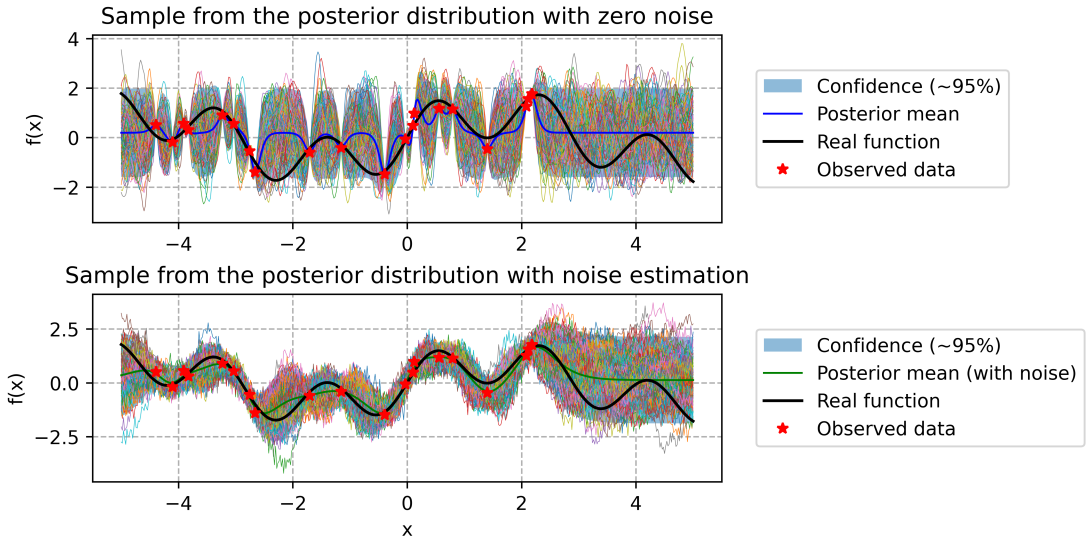


Figure 1.8: Predictive distribution and confidence interval for models build with and without noise estimation

- The fitting time of the model scales with $\mathcal{O}(n^3)$ due to the inversion of the covariance matrix.
- The kernels reflect the smoothness of the function to approximate, they also impact the stability: Matern $\frac{5}{2}$ is usually a good choice for numerical stability thanks to its derivability of order 2, as recommended in [48].

- The correlation fades with distance which can result in *flat* predictions and misleading results [48]. Even though the MLE estimation already favors less complex models and thus non-negligible length scales, a suitable option is to impose a minimum value for each length scale parameter.
- The input data X is scaled into the $[0,\ 1]^d$ hyper-cube, and the output is either standardized (zero mean and unit variance) or (less frequently) scaled between 0 and 1.

Non stationary processes: In broad terms, non-stationary refers to objective functions that abruptly vary in certain (non-canonical) dimensions. Some approaches are dedicated to tackle non-stationary random processes, since it is not tackled here we only report some references: non stationary kernels [49]; partitioning [50][51]; heteroscedastic GP coupled to space partitioning [52]; deep GP [53, 54]; or warped GP [55].

1.3 Acquisition Strategy and Parallel Computing

What makes BO special amongst SBO approaches is its AP that optimizes the AF to propose the most valuable candidate to be exactly evaluated. This section is dedicated to defining more precisely what does *valuable* means in this context.

1.3.1 Single-Point Strategies

The origin of BO is often associated with the works of H. Kushner [2] and J. Mockus [8] who respectively defined the Probability of Improvement (PI) and Expected Improvement (EI), which are single-point AFs. The framework has been popularized by D.R. Jones *et al.* in [1] where they present the Efficient Global Optimization (EGO) algorithm, whose pseudo-code is basically the same as Algorithm 1. This algorithm is further described in Section 2.1 since EGO lends itself to the tackled application.

Shahriari *et al.* propose in [11] an introduction to BO through several examples (A/B testing, Recommender System, Reinforcement learning, etc.) and set up a classification of the different AFs used in the BO framework. Three classes are identified:

- Optimistic AFs: they assume the best-case scenario regarding the uncertainty, such as in the Lower Confidence Bound (LCB) [9]. A decision $\boldsymbol{x}^*$ is favored (more likely to be selected for evaluation) by a good prediction $\mu(\boldsymbol{x}^*)$ and a high variance $\sigma(\boldsymbol{x}^*)$.
- Improvement-based AFs: they use the improvement which is defined as $I(\mathbf{x}) = max(f_{min} - f(\mathbf{x}), 0)$. For example PI [2], EI [8], Scaled EI [56], etc. Such AFs also take into account the current best-found value f_{min}.
- Information-based AFs: they focus on the information rather than the improvement. They principally rely on Thompson Sampling (TS) such as in [57] or on entropy-based AFs such as Entropy Search [10] Predictive Entropy Search [58], Max-Value Entropy Search [59], Conditional Entropy [60], etc. Such AFs mainly rely on the predictive distribution.

The last category is based on information theory, whose literature and point of view are slightly different. However, in a BO setting, it is exactly analogous to the AF.

The optimization of the selected AF returns the design point that will be evaluated next. The AP is then an optimization process itself, however, it is fundamentally different from the original global problem. First, the AF is known analytically and the gradient can be used. Second, its computational cost is much lower so that we are not restricted in the number of evaluations of the objective function. This makes this inner optimization problem easier to tackle. Obviously, it also has a computational cost, as well as the surrogate fitting, and we will come back to it later.

Despite the large variety of AFs, EI remains a popular generic choice [7]. It will be used in the following and adopting standard notations (μ and σ^2 for predictive mean and variance), it states as:

$$\mathrm{EI}(\boldsymbol{x}) = \mathbb{E}[I(\boldsymbol{x})] = \mathbb{E}[\max(f_{min} - f(\mathbf{x}), 0)] \tag{1.23}$$

$$= (f_{min} - \mu(\boldsymbol{x}))\,\Phi\left(\frac{f_{min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\phi\left(\frac{f_{min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) \tag{1.24}$$

where Φ and ϕ denote respectively the Gaussian cumulative density function and probability density function. For the same reason, we also explicitly express the LCB criterion which simply states:

$$\mathrm{LCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) - \beta\sigma(\boldsymbol{x}) \tag{1.25}$$

where $\beta > 0$ is a parameter setting the exploration/exploitation trade-off.

Recent advances in hardware and mainly general-purpose hardware generalized the use of parallel computing to solve optimization problems. However, at first sight the BO principle is inherently sequential. As previously mentioned, the main profitability of parallel computing concerns the simultaneous evaluation of several candidate points. Consequently, we need APs capable of providing batches of candidates. Of course, in sequential mode the next sampling point is chosen with maximum information. The challenge is then to be able to provide a batch of equivalent quality, as if the candidate points of the batch were chosen sequentially.

1.3.2 Multi-Point Acquisition Processes

Many methods have emerged to get around the problem of providing a batch of q candidate design points at each cycle.

In the following, we review the existing approaches that we classify into three (non exclusive) classes:

- Inherently multi-point AFs,
- Repeated single-point AFs,
- Multi-criteria approaches.

Inherently multi-point criteria

A pioneering work is presented in [16] using the q-points EI (previously defined in [61]) whose objective is to simultaneously provide q candidate design points from the optimization of the qEI criterion. However, its exact computation would require multivariate integrals which are not mathematically tractable and require approximations to be numerically computed [16, 17]. Therefore, they suggest approximating qEI with Monte-Carlo simulations. Another approach from Marmin *et al.* [62] uses the analytical form of the multi-point EI gradient to optimize the function with gradient information. We can also mention the Parallel Predictive Entropy Search from Shah and Ghahramani [63], where approximations are made to compute the predictive entropy; or the closed-form approximation of qEI of Chevalier *et al.* [64]. Wang *et al.* [65] use infinitesimal perturbation analysis to construct a gradient estimator of the qEI surface, and a multi-start technique to find the set of local optima. Nonetheless, its computation remains expensive as q increases. Indeed, the optimization problem is of dimension $q \times d$, where d is the number of design variables.

Therefore, despite attractive theoretical properties of multi-point AFs such as qEI, and efficient methods for approximating them, they will not be of much use in our parallel context since limited in batch size (typically, $q < 10$) [62], or of high computational cost. Still in [16], the authors present two heuristics allowing to mimic the q-points EI using sequentially the single-point EI. These two heuristics, namely Kriging Believer (KB) and Constant Liar (CL), are the rudiments of the second family of AP, alluded to in Section 1.3.1, where a single-point strategy is repeated to yield q points.

Repeated single-point acquisition

Intuitively, instead of simultaneously optimizing a batch of points through a multi-point AF, we try to locate distinct local optima of a single-point one. To do so, the basic idea is to modify the information returned by the surrogate model, so that the response from the AF will also be modified. One approach promoted in [16] and [17] consists in updating the surrogate model (without hyper-parameter fitting) using a *false* value of $f(\boldsymbol{x}_{new}) = y_{new}$ (cf. Algorithm 1). This *false* value must be easy to get so that it spares the time cost of the exact evaluation by the simulator. This process is repeated sequentially q times to get q candidates and only then the batch of candidates is evaluated in parallel. This approach is referred to as qEGO in reference to the EGO algorithm [66].

Algorithm 2 presents the qEGO algorithm using sequential heuristics such as CL and KB (*cf.* below). The core of the algorithm is similar to EGO (Algorithm 1), its difference lies between lines 3 and 7, where a second loop is executed instead of simply maximizing the AF. The objective of this inner loop is to form a batch of points, without any call to the simulator, which is the computationally expensive part. It consists in sequentially fitting a GP model with temporary data composed of the known data $\mathcal{D}$ to which is added the partial batch of points already assembled. Each candidate $\boldsymbol{x}^{(k)}$ results from the sub-AP of line 5 and is added to the temporary data set, along with the associated *false* response $\tilde{y}^{(k)}$. The *false* response is the predicted value (the model is trusted) in the KB heuristic and an arbitrary constant value (*e.g.* the

current best observation) in the CL one. This allows updating the surrogate model to choose another candidate until we get q ones to evaluate in parallel.

Algorithm 2 qEGO using sequential heuristics

Input
f: objective function
Ω: design space
$\mathcal{D} = (X, \boldsymbol{y})$: known data

while budget available **do**
  $\mathcal{D}_{tmp} = \mathcal{D}$
  for k in $1 : q$ **do** ▷ sequential loop
    $\mathcal{M} \leftarrow \mathcal{GP}(\mathcal{D}_{tmp})$
    $\boldsymbol{x}^{(k)} \leftarrow \arg\max_{\Omega}(\alpha(x))$
    $\mathcal{D}_{tmp} \leftarrow \mathcal{D}_{tmp} \cup \{\boldsymbol{x}^{(k)}, \tilde{y}^{(k)}\}$
  end for
  $X_{new} \leftarrow (\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(q)})$
  $\boldsymbol{y}_{new} = f(X_{new})$ ▷ parallel evaluation
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{X_{new}, \boldsymbol{y}_{new}\}$
end while
return $\min_y \mathcal{D}$

This approach introduced in [16] is of particular interest to our work since it is very simple to set up and works with any AF, or even set of AFs. However, requiring several metamodel fittings is a major drawback. Even though in the original work, the hyper-parameters of the model are not fitted in the intermediate states, the time cost remains a potential bottleneck.

During the past few years, many methods emerged trying to improve the scalability and the batch effectiveness of the BO framework. A common trait of all methods falling in the second category (repeated single-point acquisition) is that the landscape of the AF must change in some way so that optimizing AF leads to different candidates. Instead of modifying directly the landscape of the AF, one can search for distinct local optima of the single-point AF imposing a sort of distance between candidates or a penalty to discard already visited areas. With this idea, we can mention Zhan *et al.* [67] or Wang *et al.* [68] where a niching strategy is used to locate several optima of the single point EI. Gonzalez *et al.* [69] use the Lipschitz constant in order to iteratively optimize a penalized AF. The penalty has a repulsive effect on already visited areas.

Multi-point sampling from different AFs or models

Another option to simply gather a batch of points, keeping the idea of modifying the AF landscape, is to optimize the same AF but built with different models. Modifying either the hyper-parameters of the model (*e.g.* the kernel) or its learning sample [70] allows one to propose multiple candidates easily, even from the same AFs. This approach presents the advantage of being easily parallelizable. Wang *et al.* [71] use p AFs coupled with multi-point proposal

inspired by the KB heuristic from [17] to get a batch of candidates. The parallel stepwise uncertainty reduction strategy is also investigated with different AFs in [72]. Kandasamy *et al.* [57] use parallel Thompson Sampling (TS) to draw several samples (functions) from the posterior distribution and find their optimum. The work is also extended in De Palma *et al.* [14] with Acquisition TS.

Mixing selection criteria by using an ensemble of AFs is also a suitable option, and can benefit from parallel computing. The chosen AF must address different objectives (*e.g.* intensification, diversification) and then if needed, a selection is performed among all available candidates. For example, in [73] 4 AFs are used to parallelize over 4 cores. The GP-hedge algorithm from Hoffman *et al.* [74] associates a score to each AF in a portfolio and tries to employ the best choice for each cycle. We define this AP as competitive since the selection criteria are in competition with each other. Each AF provides one candidate, hopefully complementary, but independently.

On the contrary, it is also possible to use the AFs in a cooperative way. For example, using multi-objective optimization combining different AFs allows to sample from the Pareto optimal set of candidates [75][76]. We consider this kind of approach as cooperative since they act together to find a trade-off instead of acting independently. The idea of Pareto optimality between the minimum prediction and the maximum variance was already present in earlier works such as [16], without being developed. The multi-criteria approaches are also referred to as *multi-infill*.

Other strategies

A large set of strategies fall into the previous classes, however some others do not. For example, De Ath *et al.* [77] propose ϵ-Shotgun that selects a first candidate using single-point AF and then samples around it according to a Gaussian distribution centered at $\boldsymbol{x}_{b1}$ with the standard deviation computed using a locally estimated Lipschitz constant.

1.4 Chapter's Conclusion

In this chapter, we first provided the theoretical background for PBO. We detailed the construction of the GP-based surrogate model that is essential in the BO framework. We also exposed different options for the parallelization of the algorithms, and we have seen that being able to provide an effective batch of candidates is critical in the parallel setting. Finally, we reviewed the different approaches for the batch acquisition.

In optimization, we often refer to the No Free Lunch Theorem [78] as to say there is no perfect algorithm that suits any context. In particular regarding the acquisition of large batches of candidates, current approaches suffer from some limitations. Despite the variety of existing approaches, most of the mentioned methods are not suited for large batches either because they become excessively time-consuming (*e.g.*, the sequential heuristic used in KB-qEGO), or less efficient (*e.g.*, TS tends to over-exploit and thus be redundant) [12, 15].

In the following chapter, we demonstrate the usefulness of BO on real-world problems of different dimensions and execution times. We also emphasize the previously stated challenge that consists in choosing the most effective batch of candidates.

Chapter 2

Observations on Real-World Problems

We have detailed in Chapter 1 the components of PBO which are the surrogate model and the acquisition strategy to obtain batches of points. BO offers a valuable framework for many real-world applications, and we demonstrate its efficiency in this chapter. Chapter 2 presents three real-world optimization problems from engineering in which BO and PBO can be applied. The choice of the most suitable algorithm directly depends on the characteristics of the tackled problem. Through these three real-world problems, we give an overview of the scope of BO algorithms and mostly we identify associated limitations and some insights for their design.

In Section 2.1, we tackle an inverse identification problem in mechanical engineering. The objective of this first problem is to fit the parameters of a cutting model so that the model accurately describes the cutting process. The model takes 5 parameters which are our design variables. Due to complex numerical computations, the evaluation of a set of design variables is extremely time-consuming. However, parallel computing can be used to reduce the time of a single simulation. In this situation, the parallel evaluation of several candidate points might not be the best option. Consequently, we apply the sequential EGO algorithm.

One parallel version of the EGO algorithm is applied in Section 2.2, where we deal with an electrical engineering problem whose objective is to find the best bidding strategy for an operator to participate in the energy market. The particularity of this problem is that the decision must be taken in a restricted time so that the solution takes part in the daily energy market. The budget is then very limited. In this situation, the simulator is time-consuming in the sense that the number of calls to the simulator must be limited. The simulator considers 3 decision variables and the PBO is compared to an *offline* strategy where the GP model is fit before being used in an Evolutionary Algorithm (EA). The model is not updated at each cycle. In comparison, BO is said *online* as it is updated with the incoming information.

We have seen in recent years that being able to predict the evolution of epidemics and evaluate the impact of certain decisions is important in healthcare. Section 2.3 deals with an optimization problem applied to tuberculosis transmission control. The application is designed to find the best vaccination strategy to minimize the prevalence of the disease. This problem involves 6 design variables, and parallel computing can be leveraged to improve the optimization process. We use the qEGO algorithm with the KB heuristic and compare the BO approach to population-based algorithms in terms of batch effectiveness and scalability.

The observations from the latter real-world problems offer a better characterization of the challenges we face in improving the parallelization of BOAs. First, we observe that the batch acquisition does not preserve the quality of its sequential counterpart. Actually, even considering the increased number of simulations offered by the parallelization of the simulations, large batches degrade the final outcome. Second, we denote a bad scalability caused by the increasingly time-consuming sequential part of the algorithm.

The contributions presented in this chapter has been published in the four following articles:

- *In academic journals*:
  - Kugalur Palanisamy, N., Rivière Lorphèvre, E., Gobert, M., Briffoteaux, G., *et al.* Identification of the Parameter Values of the Constitutive and Friction Models in Machining Using EGO Algorithm: Application to Ti6Al4V. *Metals* **12.** ISSN: 2075-4701. https://www.mdpi.com/2075-4701/12/6/976 (2022)
  - Ducobu, F., Kugalur Palanisamy, N., Briffoteaux, G., Gobert, M., *et al.* Identification of the Constitutive and Friction Models Parameters via a Multi-Objective Surrogate-Assisted Algorithm for the Modeling of Machining - Application to ALE orthogonal cutting of Ti6Al4V. *Journal of Manufacturing Science and Engineering*, 1–54. ISSN: 1087-1357. eprint: https://asmedigitalcollection.asme.org/manufacturingscience/article-pdf/doi/10.1115/1.4065223/7324066/manu-23-1749.pdf. https://doi.org/10.1115/1.4065223 (2024)
  - Briffoteaux, G., Gobert, M., Ragonnet, R., Gmys, J., *et al.* Parallel surrogate-assisted optimization: Batched Bayesian Neural Network-assisted GA versus q-EGO. *Swarm and Evolutionary Computation* **57,** 100717. ISSN: 2210-6502. http://www.sciencedirect.com/science/article/pii/S2210650220303709 (2020)
- *In conference proceedings, with peer reviewing, and presentation*:

– Gobert, M., Gmys, J., Toubeau, J.-F., Vallée, F., Melab, N. & Tuyttens, D. Surrogate-Assisted Optimization for Multi-stage Optimal Scheduling of Virtual Power Plants. in *2019 International Conference on High Performance Computing Simulation (HPCS)* (2019), 113–120

2.1 Potential of EGO in Solving Expensive Simulation-Driven Problems

The advancement in computer science enables the development of numerical models that emulate real experiments. Those models generally require to be tuned for specific applications by fixing the hyper-parameters of the model and accurately describe the reality. The numerical modeling of engineering processes usually result in computationally-intensive simulations which makes the optimization of the hyper-parameters challenging. This process is called *inverse identification*. It is applied to an orthogonal cutting problem and solved using the Efficient Global Optimization (EGO) algorithm [1].

2.1.1 Inverse Identification in Mechanical Engineering

The problem introduced in this section consists in finding the parameters of a model that describes the machining of an alloy. The Ti6Al4V alloy is considered for an orthogonal cutting process, as illustrated in Figure 2.1. The yellow cutting tool enters the alloy in rotation when moving forward following the red axis (orthogonal to the rotation axis). This alloy is commonly used in aerospace, biomedical, and marine fields for its excellent properties. It is considered as a major production industry concern because of its poor machinability characteristics [79–81].

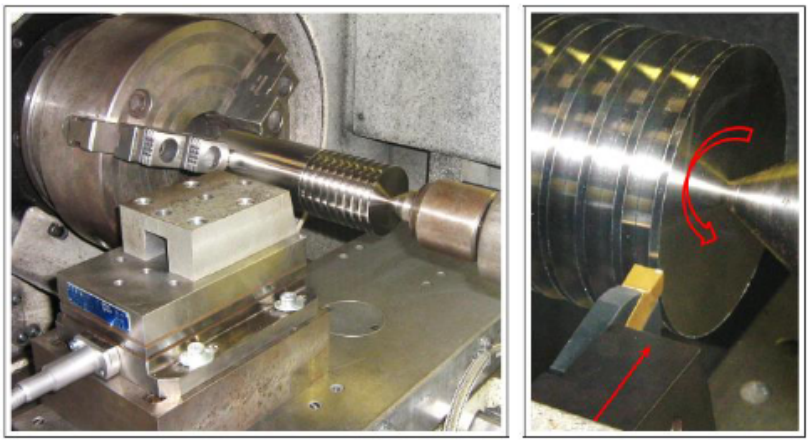


Figure 2.1: Illustration of the orthogonal cutting process [82]

Finite element modeling is generally used for the simulation of the metal-cutting process [83]. This finite element model is used in place of the experimental cutting and is referred to as the simulator in the following. It makes it possible to execute a larger set of tests and predict quantities that are difficult to obtain experimentally [84]. However, modeling the machining process using finite element methods is challenging in several aspects. The accuracy of the model depends on the model's choices such as the formulation type, the quality of the mesh [85], the boundary conditions, the material constitutive and friction models [86–89].

In particular, material model and friction conditions between tool and chip are essential to obtain accurate and reliable results from the simulation [87]. In this present work, the Johnson–Cook constitutive model and Coulomb's friction coefficient are implemented in the simulator that approximates the cutting process. The objective is then to jointly calibrate the parameters of the two latter using an inverse identification procedure to minimize the prediction error. But first, we describe the simulator and its relation with the objective function.

The simulator

The formulation of the objective function considers the Johnson-Cook model and the Coulomb friction for the finite element method. For the sake of clarity, we adopt here the *black-box* point of view and only consider the optimization aspects of this work. The full justification and motivations behind those choices are given in [21]. We summarize in Figure 2.2 the constitutive elements of the simulator and provide a succinct description of the variable of interest.

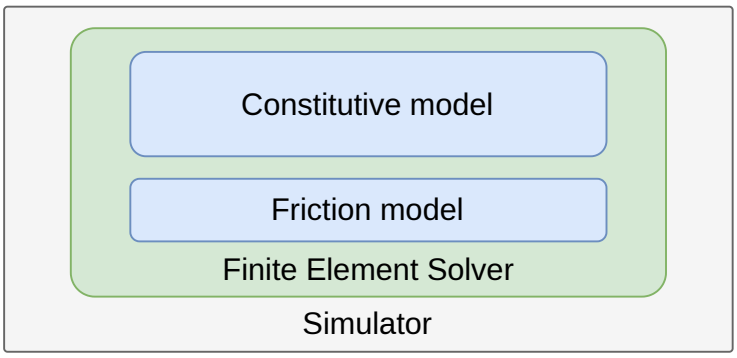


Figure 2.2: Constitution of the simulator

The Johnson-Cook model is a constitutive model which relates the flow stress (σ) to strain (ϵ), strain rate ($\dot{\epsilon}$) and temperature (T). A constitutive model usually writes $\sigma = \sigma(\epsilon, \dot{\epsilon}, T)$. It is widely used and possesses five material parameters (A, B, n, m, C) to describe the flow stress σ during the orthogonal cutting process. The formula states:

$$\sigma = [A + B\epsilon^n] \times \left[1 + C \ln\left(\frac{\dot{\epsilon}}{\dot{\epsilon}_0}\right)\right] \times \left[1 - \left(\frac{T - T_{room}}{T_{melt} - T_{room}}\right)^m\right] \tag{2.1}$$

The yield stress of the material at a reference temperature is given by the A parameter, B gives the modules of strain hardening, n the strain-hardening exponent, C the strain rate sensitivity, and m is the thermal softening exponent. The temperature parameters are identified by the letter T, where T is the current temperature, T_{melt} and T_{room} are respectively the melting and the room temperatures. Last, $\dot{\epsilon}_0 = 1$ is the reference strain rate.

The friction model is also an important factor influencing the accuracy of the simulation. The Coulomb's friction model is used to explain the friction conditions at the tool-chip interface. Coulomb's model stated in Equation 2.2 assumes that the frictional sliding force (τ) is proportional to the applied normal load (σ). The μ parameter is called the friction coefficient and is constant in all the contact length.

$$\tau = \mu\sigma \tag{2.2}$$

The set of design variables is composed of (B, C, n, m) from the Johnson Cook constitutive model and μ from the Coulomb friction model. The yield stress parameter A of Equation 2.1 is attributed 997.9 MPa in accordance with [90, 91] and is not a design variable in our study. Hence, the decision vector of the optimization problem is $\boldsymbol{x} = (B, C, m, n, \mu) \in \mathbb{R}^5$.

To achieve an efficient cutting process, the knowledge of the forces experienced during the cutting of a material is essential. It affects the tool wear, the workpiece surface quality, etc. An accurate prediction of forces during the simulation process is then also essential. The objective function is built on three quantities provided by the finite element model: the cutting force F_c, the feed force F_f and the chip thickness h'.

Figure 2.3 illustrates these quantities in the machining process. The uncut chip thickness h is a user-defined parameter that is subject to change. Ideally, the model parameters identified with the optimization process are well suited for any h.

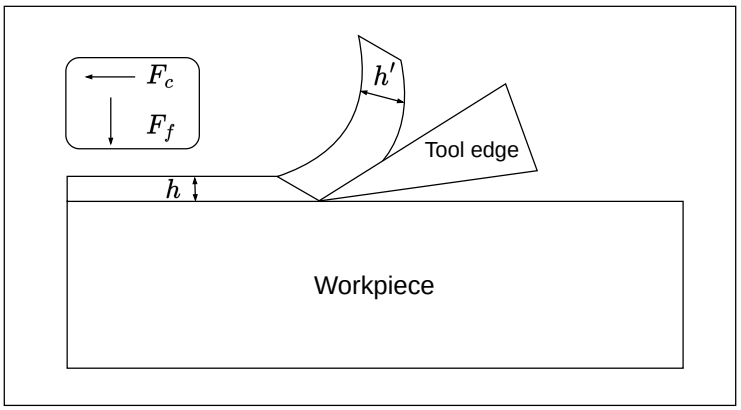


Figure 2.3: Illustration of the mechanical cutting process and the monitored quantities

Adopting the black-box point of view, we summarize the simulator in Figure 2.4. The set of design variables are the inputs of the simulator, which returns the simulated F_c, F_f, and h'.

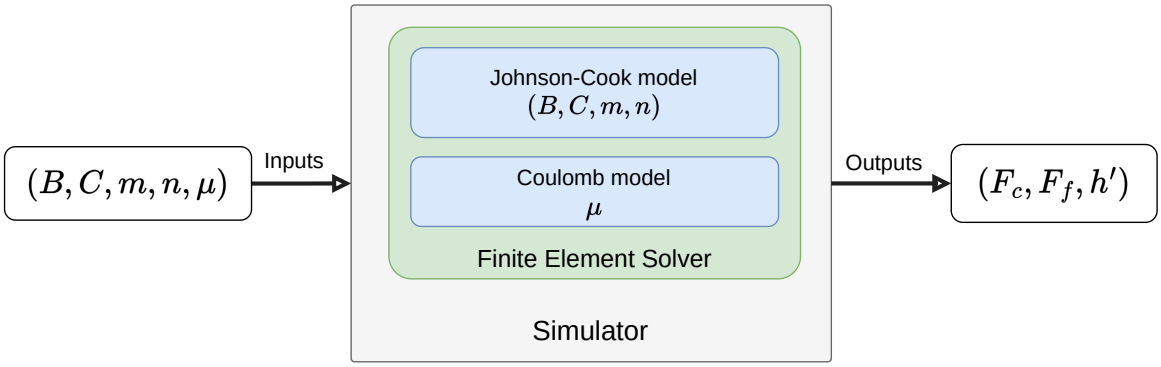


Figure 2.4: Representation of the simulator for inverse identification problem

The numerical simulator (written with Abaqus) is computationally expensive. The computation time for an uncut chip thickness $h = 0.1$ mm is about 42 minutes using the eight cores of an Intel® Core™ i7-10700 computing unit with 16GB memory.

Objective function

The objective function is defined as a measure of the error between the real experiments and the simulated ones. We denote with (exp) and (sim) exponents the quantities coming from respectively experiments and simulations.

A weighted sum method is used to evaluate the combined error of the three monitored quantities. This allows us to define the real-value objective function (single objective) given in Equation 2.3.

$$\begin{aligned} f(\boldsymbol{x}) = w_{F_c} & \frac{|{F_c}^{(sim)} - {F_c}^{(exp)}|}{\max|{F_c}^{(sim)} - {F_c}^{(exp)}|} \\ + w_{F_f} & \frac{|{F_f}^{(sim)} - {F_f}^{(exp)}|}{\max|{F_f}^{(sim)} - {F_f}^{(exp)}|} \\ + w_{h'} & \frac{|{h'}^{(sim)} - {h'}^{(exp)}|}{\max|{h'}^{(sim)} - {h'}^{(exp)}|} \end{aligned} \tag{2.3}$$

where $(w_{F_c}, w_{F_f}, w_{h'})$ is a vector of weights giving the importance of each objective with $\sum w_* = 1$.

We consider two sets of weights: the first one is a uniform distribution, $\boldsymbol{w}^{(1)}$, giving equal importance to the three outputs (F_c, F_f, h'). The second set named $\boldsymbol{w}^{(2)}$ is chosen to be more representative of the importance of the three quantities in the industrial context. The two sets of weights are given in Table 2.1.

Table 2.1: Two sets of weights for the objective function of Equation 2.3

w	w_{F_c}	w_{F_f}	$w_{h'}$
$\boldsymbol{w}^{(1)}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
$\boldsymbol{w}^{(2)}$	0.40	0.35	0.20

Ti6Al4V alloy test case

The experimental values are extracted from the work of Ducobu *et al.* [92]. The orthogonal cutting experiments are performed using Ti6Al4V with the same cutting conditions as our model for an uncut chip thickness of 0.1 mm. The results are displayed in Table 2.2.

Table 2.2: Mean objective values observed experimentally with 6 repetitions

h (mm)	${F_c}^{(exp)}$ (N/mm)	${F_f}^{(exp)}$ (N/mm)	${h'}^{(exp)}$ (mm)
0.1	173 ± 2	51 ± 1	0.135 ± 0.006

The design variables are restricted within boundaries chosen according to the literature. Table 2.3 presents the boundaries chosen in accordance with the work of Ducobu *et al.* [93], where the authors extensively investigated the literature for Ti6Al4V alloy.

Table 2.3: Boundaries of the design variables for the inverse identification problem

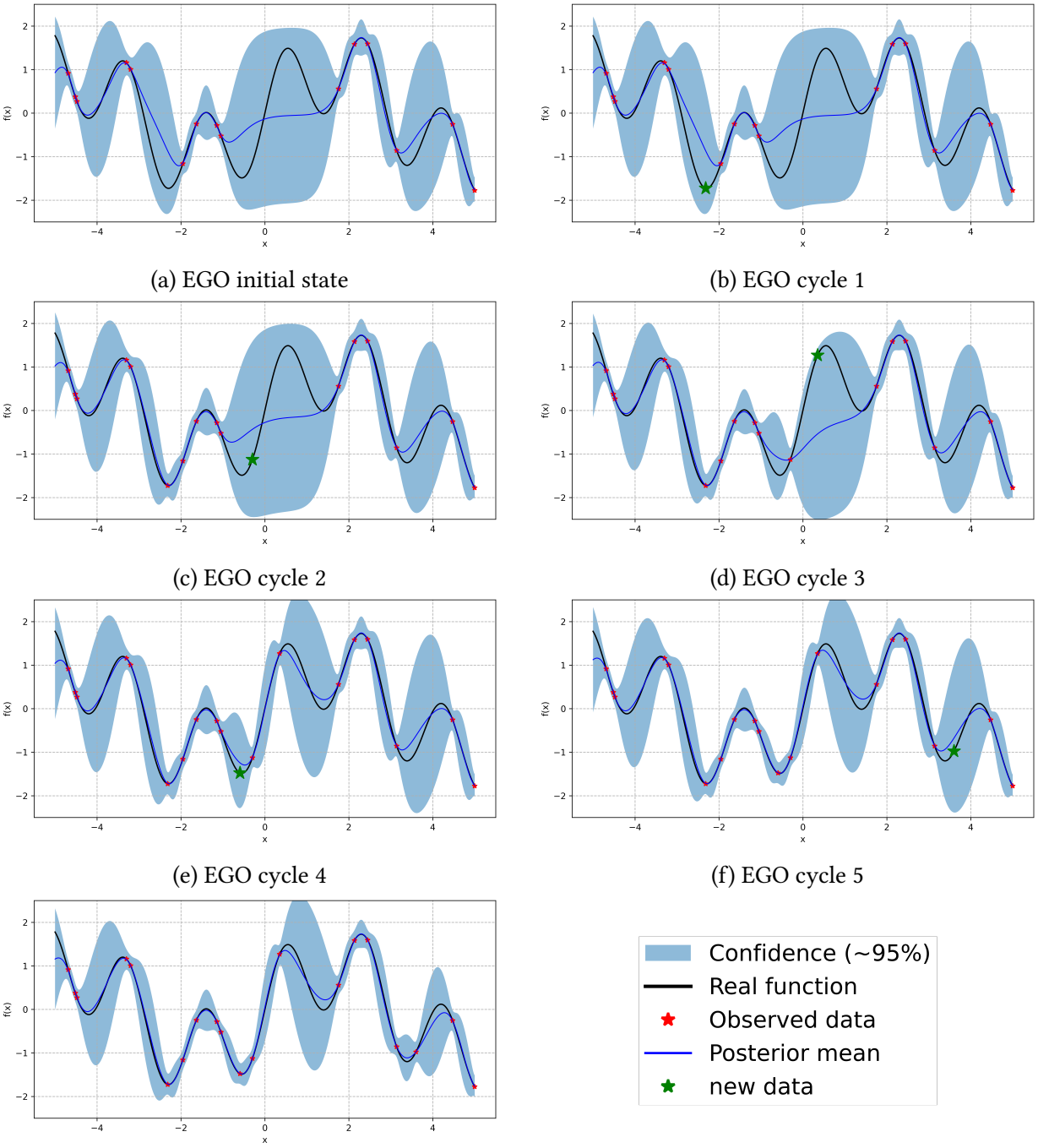
Parameters	lower bound	upper bound
B	331.2	1092
C	0.000022	0.05
m	0.6437	1.51
n	0.122	1.01
μ	0	1

2.1.2 The Efficient Global Optimization Algorithm

The objective function of Equation 2.3 evaluates the error between numerical simulation and experimental measures. The simulator is time-consuming, and the Abaqus software used for the finite element implementation of the simulator already leverages parallel computing to reduce the wall-clock time. Consequently, a sequential BO algorithm such as EGO is perfectly appropriate.

EGO is a well-known BO algorithm with proven efficiency for time-consuming black-box optimization problems [1]. We first illustrate the working principle of the algorithm through a toy problem, before applying it to the inverse identification problem.

Figure 2.5 depicts the three steps of EGO that are also presented in Algorithm 1. First, as shown in Figure 2.5a, a surrogate model is built based on the observations. Then, the first new candidate is selected by optimizing the AF, *i.e.* Expected Improvement (EI) in our case. The new candidate is marked with a green star in Figure 2.5b. This candidate is evaluated and integrated into the data set, and a new cycle can begin as represented in Figure 2.5c to Figure 2.5f. Finally, the algorithm returns the best-found outcome.



(a) EGO initial state

(b) EGO cycle 1

(c) EGO cycle 2

(d) EGO cycle 3

(e) EGO cycle 4

(f) EGO cycle 5

(g) EGO final state

(h) legend

Figure 2.5: Five cycles of EGO. A cycle is composed of a surrogate model fitting, an AP, and a simulation

2.1.3 Experimental Results

Experimental protocol

The objective function is defined according to the experimental values of Table 2.2, obtained for an uncut chip thickness of $h = 0.1$. Ideally, the obtained optimized parameters should also be valid for other values of h. Consequently, our protocol also assesses the quality of the obtained results for uncut chip thicknesses of 0.06mm and 0.04mm. The obtained results are compared with values from Seo *et al.* [91]. The total computation budget is fixed to 300 simulations, corresponding to 300 EGO cycles, which take about 8 days to complete. The algorithm is used with the EI criterion, given in Equation 1.23. Two distinct optimization runs are performed with the two sets of weights given in Table 2.1.

Implementation of EGO

Following standard recommendations [1], an initial data set is built with 60 ($12 \times d$, with the problem dimension $d = 5$) design points. The initial sample is created with the Latin Hypercube Sampling (LHS) method [94] in order to ensure a good space filling. Several GPs are built with the initial data set following a grid search pattern in order to select the best hyperparameters for the problem. The grid search involves several kernels and mean functions for the GP model, as well as various learning rates for the optimization of the model's parameters (cf Section 1.2.2). The Leave-One-Out Cross-Validation score [95] is used to compare the different combinations. All the experiments are performed using the `GPyTorch` and `BOTorch` frameworks in Python [96, 97], and revealed that the constant mean and spectral mixture kernel allow one to achieve a good accuracy with a learning rate of 0.1.

Experimental results

The results from the two runs with the two sets of weights are displayed in Table 2.4 and referred to as $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ in accordance with $\boldsymbol{w}^{(1)}$ and $\boldsymbol{w}^{(2)}$. The obtained results in terms of cutting force (F_c), feed force (F_f), and chip thickness (h') are compared to the experimental values and to the work of Seo *et al.* [91]. We observe a global improvement for both runs compared to the reference work.

Table 2.4: Cutting force (F_c), feed force (F_f), chip thickness (h'), and their differences (Δ) with the experimental results for h = 0.1 mm.

h	source	F_c (N/mm)	ΔF_c (%)	F_f (N/mm)	ΔF_f (%)	h' (mm)	$\Delta h'$ (%)
0.1	Experiment	173 ± 2	-	51 ± 1	-	0.135 ± 0.006	-
	Seo *et al.* [91]	177	2	41	22	0.177	27
	$\mathcal{R}^{(1)}$	163	6	53	4	0.147	9
	$\mathcal{R}^{(2)}$	169	2	55	7	0.150	11

The results from $\mathcal{R}^{(1)}$ provide homogeneous errors, consistently with the weights given by $\boldsymbol{w}^{(1)}$. Despite an improved cumulative error, the one associated to F_c is higher compared to the reference work. Considering the second set of weights, $\boldsymbol{w}^{(2)}$, it gives a higher importance to F_c, slightly less to F_f, and far less to h'. This results in a reduced error for F_c and a slightly increased error for F_f and h' compared to $\mathcal{R}^{(1)}$. However, in comparison with Seo *et al.*, $\mathcal{R}^{(2)}$ is still better for h' and F_f, and equivalent regarding F_c.

Validation of the results

Two additional uncut chip thickness values are considered for validating the previous results: $h = 0.04$ mm and $h = 0.06$ mm. The results of the numerical simulation as well as the experimental values and reference values are provided in Table 2.5.

Both $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ provide a quite accurate cutting force (deviation lesser than 4%) for $h = 0.04$ mm and $h = 0.06$ mm. However, the feed force F_f seems overestimated (16% to 33%) compared to experiments and is less accurate than the reference work. Regarding the chip thickness h', the prediction shows some improvement when compared with the reference values, even though they are also overestimated (14% to 19%) compared to the experimental measures.

The accuracy for the new cutting conditions is lower but still quite good, especially for the cutting force. The prediction of the latter has an important impact on tool design, and also for tool wear/life prediction [98].

Table 2.5: Cutting force (F_c), feed force (F_f), chip thickness (h'), and their differences (Δ) with the experimental results for $h = 0.04$ mm and $h = 0.06$ mm.

h	source	F_c (N/mm)	ΔF_c (%)	F_f (N/mm)	ΔF_f (%)	h' (mm)	$\Delta h'$ (%)
0.06	Experiment	112 ± 2	-	45 ± 1	-	0.080 ± 0.004	-
	Seo et al. [91]	120	7	41	9	0.112	33
	$\mathcal{R}^{(1)}$	112	0	56	22	0.093	15
	$\mathcal{R}^{(2)}$	116	4	53	16	0.097	19
0.04	Experiment	86 ± 2	-	41 ± 1	-	0.059 ± 0.005	-
	Seo et al. [91]	92	7	35	16	0.083	34
	$\mathcal{R}^{(1)}$	86	0	57	33	0.068	14
	$\mathcal{R}^{(2)}$	88	2	52	24	0.071	18

Nevertheless, the lesser accuracy for the new cutting conditions shows the limits of optimizing the parameters on a single cutting condition ($h = 0.1$ mm). A multi-objective formulation is recommended to simultaneously optimize the parameters on different cutting conditions.

Insights from the observations

This first application in mechanical engineering illustrates the efficiency of sequential BO methods such as EGO. This study does not leverage batched-BO since the simulator is already executed using parallel computing, and larger scale parallelization is not possible due to proprietary software. In this situation, it is preferable to choose the next candidate point with maximum information, *i.e.* sequentially, and use the computing power to alleviate the time cost of a simulation. Through this application, we demonstrated that BO is efficient in real-world applications and that it contributed to significant improvements for this mechanical engineering application.

The BO framework constitutes a novel approach for the inverse joint identification of the optimal parameters of the constitutive and friction models during an orthogonal cutting finite element modeling of Ti6Al4V. The identified parameters predict the forces and chip thicknesses with a better overall accuracy than the best set of parameters identified and stated in the literature.

Multi-objective formulation

Even though not being in the scope of this manuscript, an Adaptive Bayesian Multi-objective Evolutionary Algorithm has also been considered to simultaneously handle the three cutting conditions. The obtained results show a great improvement compared to the single-objective framework as it allows us to predict the cutting and feed forces with a deviation of less than 4% from the experiments for the three considered cutting conditions. We published this work in [23].

2.2 Impact of the Batched Parallelism in EGO

Our next application takes place in the electrical engineering field, and more precisely in the electricity market. We adopt the point of view of an actor in the electricity market taking part in the medium-term and short-term markets. This actor operates on an entity, called a virtual power plant, aggregating heterogeneous assets (*e.g.* thermal power plant, hydroelectric stations, etc.) whose decision to participate in the different energy markets will impact its profit. We investigate the parallel KB-qEGO [16] on this problem to find the optimal bidding strategy for maximizing the profit of the operator. This approach is compared to an *offline* Surrogate-Assisted Evolutionary Algorithm (SAEA), where the optimization is performed only using the surrogate model.

2.2.1 Optimal Commitment of Virtual Power Plants

Virtual power plants are aggregators whose initial goal is to make some profit by helping the transmission system operator maintain its frequency. Indeed, to ensure the electricity distribution grid stability, the transmission system operation must balance energy generation and consumption to ensure a 50 Hz-frequency constant within the grid. Any failure to maintain

this balance can result in severe consequences, including congestion or blackouts within the electricity system [99]. The transmission system operator is responsible for this constraint and has to contract energy reserves to be triggered when needed.

Practically, virtual power plants can be seen as single actors combining different generation, storage, and load management technologies. Their operators can either participate in energy markets (selling/buying energy to other actors) or offer ancillary services to the transmission system operator to help him maintain the grid stability. Such services are typically contracted in mid-term (*i.e.* week- or month-ahead) and consist of amounts of energy kept available in case the system operator needs to balance the electrical supply and demand. The optimized decision of the joint operation of the different assets allows the operator to maximize its expected profit. A simulator is designed to compute the expected profit the virtual power plant operator can hope for, by making a decision. The objective of this work is then to propose an efficient decision tool to maximize the expected profit of the market operator.

Similarly to the previous application, we adopt a black-box point of view and only succinctly present the simulator without getting into details of its conception, for which we refer to the work of J-F. Toubeau *et al.* [100].

A two-stage formulation

We consider a virtual power plant operator participating in medium-term reserves (week-ahead) as well as day-ahead and real-time energy markets. The decision process involves different embedded time horizons and each market is characterized by a decision to allocate a certain amount of resources. Consequently, the problem formulation takes the form of a two-stage algorithm.

The first stage is called medium-term optimization and concerns the reserve market decisions. The second stage is called short-term optimization and refers to decisions taken on the day-ahead and real-time markets. This second stage depends on the first stage since the amount of energy allocated to the reserve market cannot be used in the other markets, at the risk of not being able to fulfill the commitment and being penalized. This situation establishes a mutual dependency between the electricity market and the reserve market. Therefore, it is highly preferable to address both problems simultaneously as a two-stage optimization problem.

Furthermore, the short-term optimization has to be carried out daily and some moderate risk attitude can thus be envisaged here. Two-stage scenario-based optimization is therefore favored over techniques such as robust or interval optimization techniques that are known to yield conservative (and thus sub-optimal) solutions [101]. Furthermore, the optimization tool must take into account the prediction uncertainty due to unexpected load or renewable production to contract energy reserves and be able to contribute to the grid safe state restoration in case of unexpected dizziness on the electricity transmission system. This is the reason why it is treated using a stochastic algorithm, where several scenarios of possible short-term realizations are generated [102]. The mid-term uncertainty is addressed by defining representative days of wind and solar generation as well as total consumption within the portfolio. Afterwards, the day-ahead scenarios are generated.

The simulator

The aggregator's decision involves the two dependent stages, however, our decision variables only concern the reserve procurement done at the medium-term level. In practice, the operator can act on three products depending on the required responsiveness. They are:

- R_1, to be activated within 30 seconds to bring the frequency shift back to 50 Hz,
- R_2, to be fully activated between 7.5 and 15 minutes to help restoring the frequency,
- R_3, to be required at longer-term if the use of R_1 and R_2 are not sufficient.

Let us call $X_u = (R_1, R_2, R_3)$ the design variables of the first (upper) stage decision. Following the mid-term commitment, the contribution to day-ahead and real-time markets must be decided. We denote as X_l the second (lower) stage variables, on which we cannot act. X_l is a mixed-integer vector, decided through a dedicated optimization process, also depending on X_u. Consequently, the simulator can be represented as shown in Figure 2.6, where P_t represents the total profit realized at both stages.

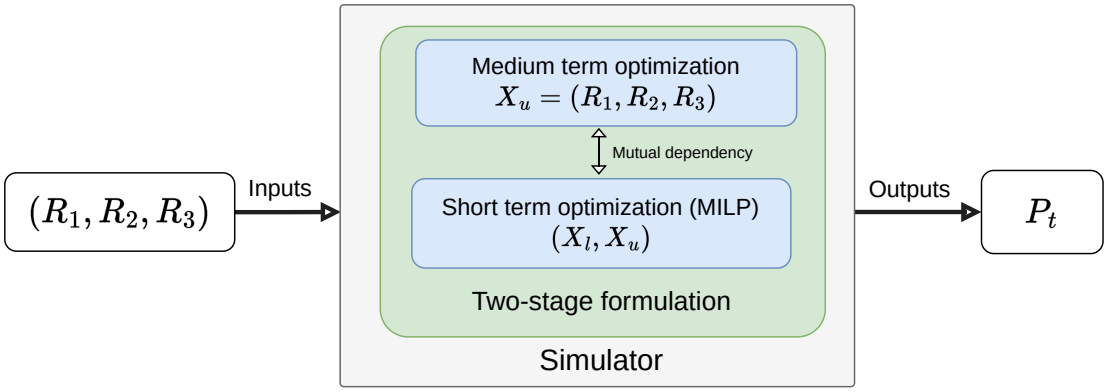


Figure 2.6: Representation of the simulator for the virtual power plant optimal commitment problem

The short-term optimization is an optimal scheduling problem and is itself a two-stage optimization problem (day-ahead and real-time markets). The decisions must consider constraints such as the physical limits of the various means of production, or the necessity to be balanced in real-time (*i.e.* what has been procured in previous market floors has to be actually delivered). The short-term optimization constitutes the non-linear part of the objective function: it gives a variable profit by optimizing resource management in day-ahead and real-time markets.

Objective function

One could formulate the problem as follows:

$$\max_{\boldsymbol{X}_u \in \Omega} \; P_t(X_u, X_l) = \max_{\boldsymbol{X}_u \in \Omega} \; P_f(X_u) + \mathbb{E}\left[P_v(X_u, X_l)\right] \tag{2.4}$$

$$s.t \; X_l = \arg\max\left(\mathbb{E}\left[P_v(X_u, X_l)\right]\right) \tag{2.5}$$

The total profit P_t of Equation 2.4 is the result of two contributions, P_f and P_v. They respectively correspond to the fixed profit and the variable profit. P_f is directly computed with a linear combination of the X_u decisions, while P_v depends on a complex mixed integer linear formulation [100] and is the result of the short-term optimization. The short-term optimization is formulated as a Mixed-Integer Linear Programming (MILP) following the model from [103].

As pointed out above, the optimizations of the two temporal horizons are intrinsically linked. The resources allocated in mid-term must remain available if they are required in real-time. The simulator gives the expected best variable profit according to the reserves commitment that guaranty a fixed profit. High commitment to the reserve is then less risky, but high resources allocated to the short-time market are potentially more profitable.

This stochastic programming approach resulting from the mentioned uncertainties has two main challenges. The first one consists of modeling the uncertainty through a set of time-varying predictive scenarios that represent time trajectories of all uncertain variables. The second one is to overcome the computational burden associated with the resulting formulation dealing with uncertainties. Both are treated in Toubeau *et al.* [100]. This is why the use of a metamodel is mandatory to complete the optimization in the dedicated time.

From the optimization point of view, we optimize the upper stage design variables X_u, trying to maximize the total profit P_t.

Description of the test case

The virtual power plant is composed of 3 conventional power plants, with a maximum output power of respectively 130, 80, and 55 MW as well as 2 pump storage units, both characterized by an output power of 15MW. In addition, there are also renewable generation devices such as wind farms and domestic rooftop photovoltaic installations for a total power of 120MW.

The prices for the reserve capacity are fixed at 16€/MW for R_1, 4€/MW for R_2, and 1€/MW for R_3. The activation prices reflect the technology-specific operation costs. The portfolio is created based on real data from the Belgian system in order to represent a typical actor. The generation and consumption patterns are realized regarding aggregated data for a typical month of July.

2.2.2 *Offline* SAO *versus* qEGO

In [100], Toubeau *et. al* adopt an *offline* strategy where the simulator is replaced by a surrogate model to spare the computational cost of the short-term optimization. Then, an evolutionary optimization is carried out using the metamodel instead of the simulator, and the best obtained X_u is evaluated afterwards with the real simulator to return the real expected profit. Following the definitions of Section 1.1.2, this method is an *offline* SAEA. The term *offline* is opposed to *online* where the surrogate model is dynamically updated with incoming information, such as in BO.

The modeling through GP is reasonable since the variation expected from small variations in the decision X_u should result in a small variation of the total profit. Consequently, the simulator is assumed stationary. The optimization must be carried out in a limited time window since it is included in a decision-making process in the energy market. Consequently, this application will be treated using batched-BO algorithms, and the approach will be compared to the initial *offline* strategy. Then, we oppose the *offline* strategy, consisting of a GP model coupled to an Evolutionary Algorithm (EA), to the *online* strategy provided by BO and more precisely to the qEGO algorithm.

Evolutionary algorithm with GP model

The short-term optimization result is replaced by a GP model, trained over a data set previously generated. As for the mid-term optimization, it is carried by an EA operating on the surrogate model that approximates the short-term management. An EA consists in evolving a population of candidate solutions through generations of individuals trying to perpetuate the *good* genes (*i.e.* design variables) that produce good values from the objective function's point of view. Classical operations involved in an EA are the selection, reproduction, and replacement [32]. The three steps are used to evolve the populations through successive generations of individuals. For a given population, the selection operator chooses pairs of individuals, called parents, to be used in the reproduction step. The newly produced individuals are called offspring and are the result of the crossover and mutation operators. The crossover selects genes from both parents to produce the offspring, while the mutation alters the offspring with a given probability after the crossover. The new individuals are evaluated with the fitness function, which is actually the surrogate model in our situation. Finally, the replacement operator is used to decide whether the newly generated individuals replace the parents or not. The algorithm loops over these previous steps until the stopping criterion is met, as described in Algorithm 3.

Algorithm 3 Evolutionary Algorithm

Input

f: objective function
Ω: design space
p_{size}: population size

$\mathcal{P} = sampling(f, \Omega, p_{size})$
while Budget available **do**
  $\mathcal{P}_{par} =$ selection$(\mathcal{P})$
  $\mathcal{P}_{off} =$ reproduction$(\mathcal{P}_{par})$ ▷ crossover and mutation
  $\mathcal{P}_{off} =$ evaluation(f, P_{off})
  $\mathcal{P} =$ replacement$(\mathcal{P}, \mathcal{P}_{off})$
end while
return best_individual$(\mathcal{P})$

EAs are very popular in solving *black-box* optimization problems [104–106], however, they usually require a large number of objective function evaluations. This motivates the introduction of the surrogate model to replace the time-consuming evaluation of the simulator.

Batch-parallel EGO

The EGO and qEGO are proposed to challenge the previously described approach. Both algorithms are introduced in Section 1.3, and the qEGO pseudo-code is given in Algorithm 2. The latter offers the possibility to evaluate the q candidates coming from the acquisition process in parallel. Since only the evaluation of the objective function is executed in parallel, it is often referred to as a batch-parallel algorithm, and q is called the batch size. Setting q to 1 is equivalent to running the standard EGO algorithm.

2.2.3 Experimental Results

Experimental protocol

The total computational budget is set to 48 simulations. The *offline* model is learned over a set of initial points generated through Latin Hypercube Sampling (LHS), supplemented with the 8 corner points for a total of 48 initial design samples. The final obtained point also needs to be evaluated by the simulator, making a total of 49 simulations. As for the EGO algorithm, the first model is set up based on an initial data set composed of 15 points generated using LHS. The remaining budget is used to evaluate points proposed by the qEGO algorithm. The considered batch sizes are $q \in \{1, 2, 4, 8\}$, which means that the algorithm performs respectively $n_{cycles} = 32, 16, 8,$ or 4 optimization cycles for a total of 32 new points. The total number of simulations is slightly less to compensate for the surrogate update during the search. However, assuming a smaller budget, the algorithm operates in an iterative fashion and could be stopped at any cycle, providing the actual best-observed point.

Implementation of the algorithms

Consistently with the initial work, the *offline* model is set up using the SUMO toolbox [107], the kernel is Gaussian, and a linear trend function is chosen to account for the linear part of the total profit. Then, the EA from the Matlab `Global Optimization` toolbox [108] is initialized with a randomly generated population. The hyper-parameters of the method are chosen according to the recommendations of the toolbox. The selection operator is stochastic uniform, attributing a higher probability of being selected to individuals with good properties (*i.e* high objective values). The crossover operator is a scatter function that generates a random binary vector that indicates from which parent the gene is selected. As for the mutation operator, it adds a small perturbation of each design variable of the individual. The perturbation is computed according to a centered Gaussian distribution with a standard deviation set to $\sigma_k = \sigma_{k-1}(1 - \frac{k}{N})$, where k is the current generation, N the maximum number of generations, and $\sigma_0 = 1$. Its probability of appearance is 0.01. Each generation is composed of 50 individuals and the number of generations is set to $100 \times d$ where $d = 3$ is the dimension.

The online approach is the q-points EGO from [16], using the CL heuristic described in Section 1.3.2. The surrogate model is also a Kriging model, with a Matern $\frac{5}{2}$ covariance kernel and a linear trend. The implementation relies on the `DiceOptim` and `DiceKriging` R packages [48].

Results and observations

The results are presented in Figure 2.7 which displays the expected profit of the operator as a function of the number of evaluations. For all the batch sizes, the average outcome (μ) is plotted in solid lines and the standard deviation σ computed over the 10 repetitions is displayed in dotted lines of the same color. The reference result from the *offline* optimization is represented by the constant orange line. First, we can see that the qEGO algorithm consistently performs better than the *offline* method. In a few evaluations, the BO algorithm achieves higher expected profit.

However, noticeable differences are observed between the different batch sizes. Indeed, increasing the batch size usually results in a lower profit for a given number of simulations. It is understandable as the large batches of points are chosen with limited information compared to small batches. Indeed, when $q = 1$, each time the algorithm requires a new candidate, it is selected according to a model fitted with all the previous information. However when $q > 1$, each additional candidate is selected using a model partially fitted on a data set including the *lies* from the CL heuristic. This results in sub-optimal choices for the next candidates and we can observe Figure 2.7 a plateau corresponding to a cycle. For example, evaluations 9 to 16 correspond to the second cycle when $q = 8$. As they are selected sequentially, it indicates that almost no improvement is done after the first sample of the batch.

Figure 2.8 displays a resized window of Figure 2.7 where we can observe more precisely the plateaus. We clearly see that the gain from new candidates is higher when it is selected at the beginning of a cycle. The effectiveness of the batch of candidates decreases when the batch size increases.

Insights from the observations

We demonstrated the superiority of the *online* qEGO algorithm compared to the *offline* coupling of an EA and a GP model on this maximization problem. However, we observed a low batch effectiveness of the qEGO algorithm. It is characterized by a loss of performance regarding the final outcome when the batch size increases, for a fixed number of simulations. This observation needs to be clarified with experiments accounting for the global optimization time. Indeed, the time should be taken into account to compare the algorithms since increasing the number of processing units, therefore of the batch size, hopefully results in a larger number of simulations in a given time. The acquisition of the batch of points draws its utility from the parallel simulations of the obtained candidates, thus a time saving. Accounting only for the number of simulations as it is usually the case withdraws the gain from parallel computing.

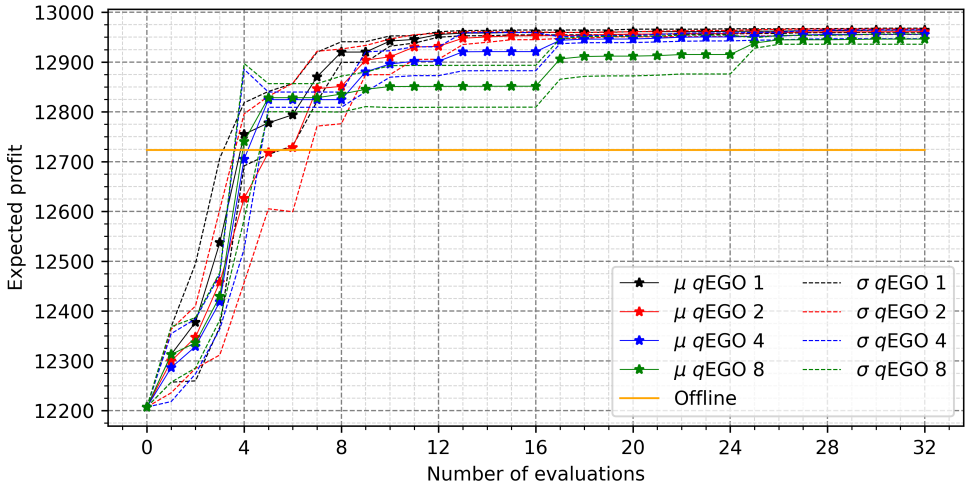


Figure 2.7: Average expected profit (μ, in €) in a function of the number of simulations for batch sizes $q \in \{1, 2, 4, 8\}$. The orange curve represents the average expected profit from the *offline* approach. Dotted lines indicate the standard deviation (σ) for each batch size.

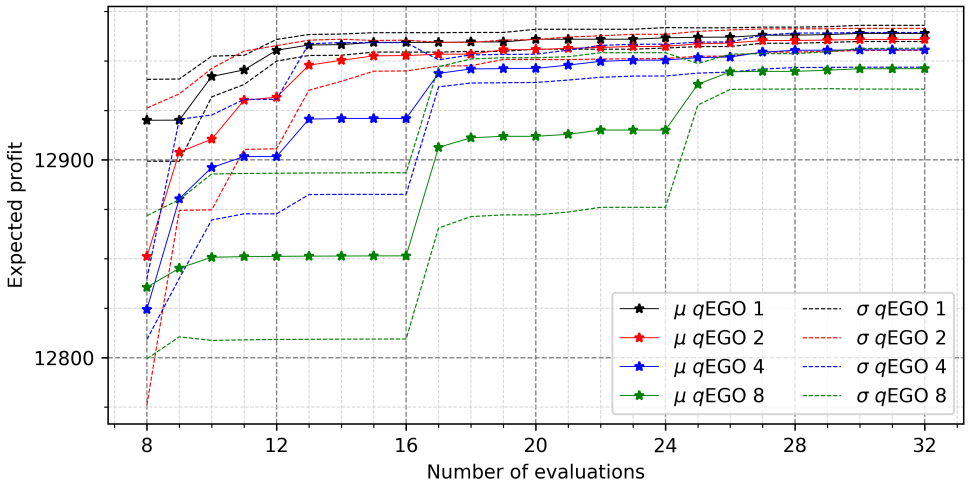


Figure 2.8: Zooming on the average expected profits (in €) in a function of the number of simulations for batch sizes $q \in \{1, 2, 4, 8\}$. Dotted lines indicate the standard deviation (σ) for each batch size.

2.3 qEGO *versus* Surrogate-Assisted EA

Properly calibrated, mathematical models have the ability to simulate the transmission within a population and estimate the impact of control interventions on disease propagation [109]. In particular, the AuTuMN model, used in this work, is an ordinary differential equation-based system modeling the impact and cost of tuberculosis control programs [110]. In this problem, we are searching for the best strategy to minimize the impact of a disease. The time cost of the simulator is smaller than for the precedent applications, allowing a larger number of simulations, even in restricted time. Surrogate-Assisted Evolutionary Algorithm (SAEA) are known to scale efficiently with the number of computing units, consequently we confront this population-based algorithm to the qEGO algorithm on this minimization problem.

2.3.1 Tuberculosis Transmission Control (TBTC)

Tuberculosis is an airborne disease that has been threatening mankind for thousands of years and still affects 10 million individuals each year, killing around 1.7 million of them [111]. Main global health agencies and funders increasingly rely on mathematical modeling to design better tuberculosis control policies. Optimization of resource allocation is increasingly popular in the context of global health. The AuTuMN model [110] is used to create a simulator that estimates the impact of a preventive treatment allocated across different age groups. The objective is then to find the best allocation strategy to limit the prevalence of the disease. The prevalence reports the number of infected people over 100 000 at a given time.

The simulator

The simulator is a black-box computing program built with AuTuMN that computes the expected prevalence of the tuberculosis disease after a period of time, given the initial conditions of a population and a specific control program. The prevalence is used to measure the effectiveness of the control program. Given a limited number of treatments available each year, the treatments are distributed among d age groups. The number of age groups constitutes the number of design variables $\boldsymbol{x} = (x_1, \ldots, x_d)$, which are linked by the relation: $\sum_{i=1}^{d} x_i = n$, where n is the number of available treatments. Consequently, the design variables are integers. However, they are implemented as real numbers in the optimization process. The effectiveness of this strategy is measured by the prevalence returned by the simulator.

The objective function can be stated as follows:

$$\min_{\mathbf{x} \in \mathbb{N}^d} f(\mathbf{x}), \tag{2.6}$$

$$s.t \sum_{i=1}^{d} x_i = n, \tag{2.7}$$

Presentation of the test-case

For this application, we consider the Philippines, a high tuberculosis burden country with a disease prevalence of over 1% measured in 2016. The objective is to determine the allocation of preventive treatments that would minimize the estimated prevalence in 2035. Six age categories are considered and the optimization variables represent the number of treatments to be allocated to each of these sub-groups, considering a total number of preventive treatments of $n = 600\ 000$ per year starting from 2020.

2.3.2 Competing Approaches

Considering the TBTC test case, one evaluation of the objective function lasts up to 20 seconds. In this situation, the simulation time is only moderately expensive. Indeed, it is costly enough to discard evolutionary approaches such as EAs, yet BO might not be suitable either because of the surrogate model time cost. Indeed, it is expected that the data set becomes rapidly substantial because of the 20-second simulation, and the use of parallel computing. In this context, we propose two approaches to tackle this problem: the qEGO algorithm will be used and confronted with a Surrogate-Assisted Evolutionary Algorithm (SAEA) algorithm. SAEA is a SAO algorithm (see Figure 1.4) using an EA to guide the optimization, and a surrogate model to either evaluate the objective function or to decide which offspring are to be exactly simulated.

In [112], the authors proposed a Bayesian neural network-assisted EA that has been applied to another TBTC problem. However, this work relies on either the surrogate prediction, or the surrogate uncertainty, but not both. Inspired by EGO, the present work appraises the introduction of the EI criterion into the EC so that both the prediction and uncertainty are used jointly to select the new batch of points. An extensive comparison of the different ECs is conducted in [19]. In this work, we focus more on the differences between BO algorithms (more exactly qEGO) and SAEAs.

Dealing with the linear constraint in qEGO

The qEGO algorithm is used with the KB heuristic, and referred to as KB-qEGO in the following. To take into account the constraint of Equation 2.7, the AP will be managed by an EA dealing with the constraint directly into the reproduction operator so that all candidates are admissible. The generic EA is described in Algorithm 3 and only the reproduction step needs adjustment to take into account the linear constraint. The algorithm operates as follows: first, the parents are randomly selected without replacement in the population; then the reproduction is carried on by randomly selecting each attribute from one of the parents; and finally, the result is re-scaled to fulfill the constraint. Two offspring are produced for each pair of parents, and they replace these latter if they improve the objective.

Bayesian Neural-Network Assisted EA

In SAEA [113], the model is often an artificial neural network [114] [115] [116]. The used algorithm, further described in [20], proposes to use the EI (see Equation 1.23) as an EC to assist the optimization. Doing so, the optimization can benefit from recommendations taking into account the standard deviation of the prediction. However, unlike Kriging, neural networks are easier to train but do not provide uncertainty information around their predictions [117]. The development of Bayesian neural networks based on Monte Carlo dropout seems to provide a suitable alternative to retain the best of both Kriging and artificial neural networks [118] [119]. The Bayesian neural network with Monte Carlo dropout is adopted as surrogate model to compute the EI into the EC.

Monte Carlo dropout consists of generating n_{subnet} sub-networks by dropping out neurons from the fully connected network. The final prediction and standard deviation are derived from the standard mean and variance estimators, recalled in Appendix A.1.1. The surrogate model is fitted with all the available data of simulated individuals. For each new cycle, which ends with the evaluation of the new batch of points, the surrogate model is updated using the weights from the model built during the previous cycle. Thanks to the incremental fitting, the time needed to fit the network is greatly reduced.

In this SAEA, the number of produced offspring n_{off} is larger than the batch size q so that the offspring are split into two groups. The first one, composed of q points, will be exactly evaluated while the second composed of the remaining points will be only predicted. The surrogate model is then used as an evaluator and as a filter. The selection is done according to the EI value so that the most promising candidates are exactly evaluated (*i.e.* surrogate as a filter) and the other ones, *a priori* less critical are only predicted (*i.e.* surrogate as an evaluator).

Algorithm 4 summarizes the EC steps of the Bayesian neural network-assisted EA, which will be denoted as BNN-GA in the following. Given the current population $\mathcal{P}$, and a model $\mathcal{M}$ fitted on the whole data set $\mathcal{D}$, a population of offspring is created based on the selected parents as state lines 1 and 2. Then, in line 3, the offspring are evaluated by the EI criterion. The ones with higher EI are used to form a sub-population of simulated (evaluated with the real simulator) individuals, while the remaining form another sub-population with predicted individuals. Line 4 refers to the new batch of points to be simulated by X_{new}, consistently with Algorithm 1. This batch is evaluated and added to the data set at respectively lines 6 and 7. As for the second population of predicted individuals, it is evaluated by the surrogate model prediction and designated by $\mathcal{P}_{pred}$ at line 9. Both populations take part in the replacement step at line 10 to update the current population. However, only the simulated one is saved into the data set as shown in line 7.

2.3.3 Experimental Results

Experimental protocol

As stated in Section 2.2, having a budget defined by the number of simulations does not enable us to assess the suitability of a parallel approach since the number of cycles depends on the

Algorithm 4 Evolution Control in Bayesian Neural Network assisted Evolutionary Algorithm

Input

$\mathcal{P}$: current population
$\mathcal{M}$: surrogate model
$\mathcal{D} = (X, \boldsymbol{y})$: current data set
q: batch size

$\mathcal{P}_{par} = \text{selection}(\mathcal{P})$
$\mathcal{P}_{off} = \text{reproduction}(\mathcal{P}_{par})$
$\alpha = \text{EI}(\mathcal{P}_{off}, \mathcal{M}, \min(\boldsymbol{y}))$
$X_{new} = \text{sort}(\mathcal{P}_{off}, \alpha)[:q]$ ▷ select q points according to EI
$\mathcal{P}_{pred} = \mathcal{P}_{off} \backslash X_{new}$
$\boldsymbol{y}_{new} = \text{evaluate}(X_{new}, f)$ ▷ q parallel simulations
$\mathcal{D} \leftarrow \mathcal{D} \cup \{X_{new}, \boldsymbol{y}_{new}\}$
$\mathcal{P}_{sim} = \{X_{new}, \boldsymbol{y}_{new}\}$ ▷ Simulated population
$\mathcal{P}_{pred} = \text{evaluate}(\mathcal{P}_{pred}, \mathcal{M})\}$ ▷ Predicted population
$\mathcal{P} = \text{replacement}(\mathcal{P}_{sim}, \mathcal{P}_{pred})$

batch size. One could fix the budget as a number of cycles so that the total number of simulations is $n_{sim} = q \times n_{cyc}$. However, this does not account for the time required to set up the surrogate model, which increases fast with the data set size in PBO algorithms. Another choice, adopted here, is to rely on a fixed time budget. Actually, time is often the limiting factor of the studies, and this appears as a relevant budget definition in the parallel computing setting. Both algorithms are then evaluated based on their capability to find good objective values in a given time, and their ability to use the computing resources.

The search starts with the generation of a random initial sample (respecting the constraint) composed of 128 points, and a budget of 30 minutes is granted to each algorithm. The considered batch sizes are $q \in \{1, 8, 16, 32\}$ and the optimization runs are repeated 50 times for each. There are 50 initial sets, so the two algorithms start with the same data for the 50 repetitions on 4 different batch sizes. Experiments are run on a cluster composed of 8 compute nodes, involving two 16-core AMD EPYC 7301 CPUs. One evaluation of the simulator lasts 6 to 20 seconds on a single core.

Implementation of the algorithms

The KB-qEGO algorithm is implemented using the R packages `DiceKriging` and `DiceOptim` from Roustant *et al.* [48]. The Kriging model is built with a linear trend, and a Matern$\frac{5}{2}$ covariance kernel as advised in [48]. A small perturbation (*i.e. a jitter*) is added to the metamodel to avoid ill-conditioning of the Kriging matrix. It is automatically estimated during the fitting of the model. The EI maximization is performed with an EA to manage the linear constraint as explained previously. The population size is set to $n_{pop} = 150$ and the number of generations is limited to $n_{gen} = 15$.

The Pagmo library [120] library is used to implement the Bayesian neural network-assisted EA. The neural network hyper-parameters are fixed to 2 hidden layers, 12 neurons per layer, ReLU activation function, a learning rate of 0.3, and a probability of dropping the neuron of $p_{drop} = 0.1$. The EA is configured as follows: the population size is $n_{pop} = 128$, and the number of offspring depends on the batch size q so that $q = 0.25 \times n_{off}$. Consequently, $n_{off} \in \{4, 32, 64, 128\}$. A simulated binary crossover with probability 0.9 and distribution index 10 is used along with a polynomial mutation with probability 0.1 and distribution index 50. The reproduction operator also takes into account the constraint of Equation 2.7 by re-scaling the offspring. The replacement operator is a tournament of size 2. In addition to these parameters, the surrogate model fitting uses an early stopping strategy with a tolerance of 10^{-4} during 56 iterations. The stochastic gradient descent is used with a Nesterov momentum of 0.1 and the training set is normalized to lie into $[0, 1]^d$.

Results and observations

The results of the experiments are reported in Figure 2.9. It shows the average prevalence as a function of the number of simulations. The budget is defined as a fixed time, consequently, the number of simulations differs with the batch sizes. Ideally, with perfect scalability, increasing the batch size proportionally increases the number of simulations. The curves are truncated to the minimum number of evaluations so that each point represents the average over 50 runs. The last point also gives the minimum number of simulations performed by the algorithms for the given budget of 30 minutes. The rhombus-shaped point indicates the common best prevalence from the initial sample. The first overall observation is that KB-qEGO is very sample-efficient. Indeed, after only a few cycles the prevalence is reduced from about 803 to 797. On the contrary, the Bayesian neural network-assisted EA needs a lot more evaluations to achieve equivalent results.

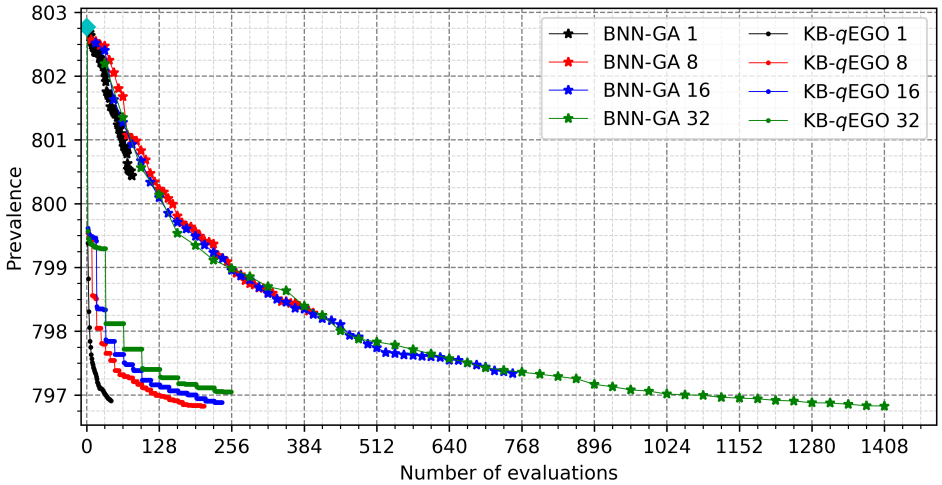


Figure 2.9: Average prevalence according to the number of simulations. The mean is computed over 50 repetitions. The rhombus-shaped point represents the common initial best value.

Regarding the effectiveness of the batch size, we can see that KB-qEGO faces difficulties to profit from the parallel setup despite the sample efficiency pointed out above. When looking at Figure 2.10, which focuses on the KB-qEGO results, we only observe slightly better outcomes when increasing the batch size from $q = 1$ to $q = 8$, and no improvement is visible between $q = 1$ and $q = 16$ despite a larger number of simulations. Increasing the batch size to $q = 32$ even deteriorates the final outcome of the KB-qEGO method. Actually, the same plateaus observed in Figure 2.7 are observed in Figure 2.10 corresponding to the different cycles. For a given number of simulations, it is always preferable to keep a small batch size, as already observed in Section 2.2 with the optimal scheduling problem. As for the SAEA, not all the points are plotted since there is no logical order in the generation of the offspring as there is in KB-qEGO. BNN-GA manages to take advantage of the larger batch size. Indeed, the performance of BNN-GA for a given number of simulations is equivalent for the different batch sizes. Therefore, increasing the batch size allows the algorithms to perform more simulations and achieve better outcomes.

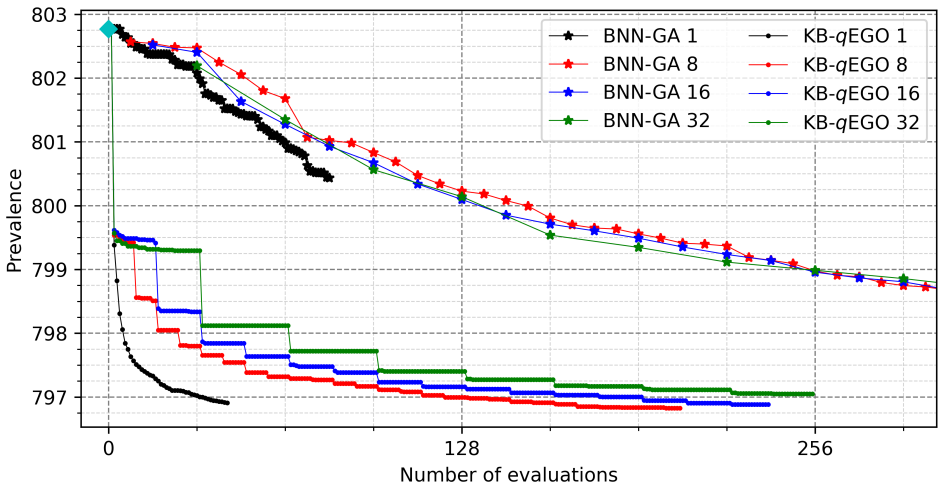


Figure 2.10: Focus on the average prevalence according to the number of simulations for KB-qEGO. The mean is computed over 50 repetitions. The rhombus-shaped point represents the common initial best value.

In addition, doubling the number of processing units and the batch size does not result in a significant increase in terms of simulations for the KB-qEGO algorithm. Going from $q = 1$ to $q = 8$ significantly increases the number of simulations, however the latter does not persist from $q = 8$ to $q = 16$ and $q = 32$. The surrogate model fitting and AP take an important part of the time budget, especially when the batch size is high. Regarding BNN-GA, we can see in Figure 2.9 that the number of simulations is almost proportional to the batch size, which indicates a much better scalability. More precisely, Table 2.6 reports the minimum number of simulations performed over the 50 repetitions by each algorithm for each batch size. It also displays the gain related to the increase of the batch size, and the scaling factor. We can see, as already observed in Figure 2.9, that the gain in terms of simulations does not scale well with q for KB-qEGO. However, the scaling factor decreases much slower for BNN-GA. Having $q = 32$ available computing cores implies being able to execute 17.6 times more simulations. This is quite good considering the surrogate model fitting part is executed sequentially, which becomes

more and more time-consuming as the data set gets bigger. This difference is most likely due to the surrogate model fitting, which is done incrementally for the Bayesian neural network allowing to save considerable time.

	q	1	8	16	32
KB-qEGO	n_q	44	208	240	256
	gain (n_q/n_1)	1	4.7	5.5	5.8
	scaling factor $(n_q/(n_1.q))$	1	0.59	0.34	0.18
BNN-GA	n_q	80	400	752	1408
	gain (n_q/n_1)	1	5	9.4	17.6
	scaling factor $(n_q/(n_1.q))$	1	0.63	0.59	0.55

Table 2.6: Minimum number of simulations over the 50 repetitions of KB-qEGO and BNN-GA according to the batch size. With perfect scalability, $n_q/n_1 = q$, and $n_q/(n_1.q) = 1$.

Regarding the results from an epidemiological point of view, the prevalence reduction from 803 to less than 797, as suggested by the results, represents a significant improvement in the epidemiological situation of the Philippines. Indeed, given that the population of this country is over 100 million, the reduction induced by optimization is equivalent to more than 6,000 cases of tuberculosis prevented in 2035.

Insights from the observations

The observed results for this optimization problem refine the analysis of Section 2.2 by comparing the standard KB-qEGO algorithm to an efficient Bayesian neural network-assisted EA. The KB-qEGO algorithm struggles to exploit the full capacity of the available computing resources, mainly when the batch size q passes 8. The present assertion is supported by the low effectiveness of the batch size (indicated by the plateaus) and the bad scalability regarding the gain in terms of simulations with larger batch sizes.

2.4 Chapter's Conclusions

The conclusions of the precedent studies point out the low parallel potential of the KB-qEGO algorithm, which can be generalized to many BO algorithms. In some situations, characterized by a moderately time-consuming objective function, the data set can become substantial, especially using parallel computing. Consequently, it makes the fitting cost increasingly time-consuming and highly superior to the simulation time. In addition, the AP (excluding model fitting) can also become time-consuming when the batch size is large. The combination of the two aspects leads most BO algorithms to scale poorly with the batch size.

A lot of approaches consider learning and acquisition time negligible compared to the simulation time. However, for many applications, the simulation time might be quite small but still considered high if the computational budget is defined as a restricted time. A pertinent

approach would be to compare the simulation time and the acquisition time (model fitting time included) to choose the adapted family of algorithms. Considering this, the previously stated limitations become even more significant.

Making BO parallel raises new challenges relative to scaling to large-scale parallel computation. Even though we know how to choose a few (≤ 8) candidates efficiently [16, 17], it is still a challenge to propose larger batches of candidates to exactly evaluate [12, 15]. Recent parallel computers involve thousands of computing cores and even general-purpose products involve 16 cores per CPU that can be used in many applications.

The need for a better scalability takes us to reconsider the sequential AP of qEGO by inserting parallelism inside the AP. The low batch effectiveness also reveals a defect in the diversification/intensification trade-off of the batch of candidates. The main leads for improvement are the reduction of the sequential part of the algorithm to better exploit the computing resources, and the use of complementary criteria in the AP. We investigate space decomposition and multi-infill approaches in the following chapters. We rely on the **scalability** and **batch effectiveness** as performance indicators for the considered algorithms.

Part II

Contribution to the Design and Analysis of Parallel Hybrid BOAs

Chapter 3

BSP-EGO: a New Decomposition-based EGO

The contributions of this chapter concern the development of a new parallel algorithm and its performance investigation regarding recent state-of-the-art algorithms on both benchmark

and real-world problems. This algorithm includes an Acquisition Process (AP) fully parallelizing the acquisition of new candidates by partitioning the search space. Indeed, a basic approach to improve the scalability is to split the computational workload between the worker units. This technique is also very common in Global Optimization (GO) to better control the trade-off between local and global search. For example, the DIRECT algorithm [121] has been widely used and derived in various forms [122]. Space decomposition-based algorithms explore methodically the whole search space while being easily parallelizable since the jobs in each partition may be performed concurrently without interfering. Consequently, they add a level of parallelism to BO authorizing us to consider higher evaluation budgets in equivalent time. Actually, they supply another way to balance exploration and exploitation by determining which region is better being explored. Based on that idea of spatial decomposition, we propose Binary Space Partitioning Efficient Global Optimization (BSP-EGO) [27, 123] and Local model-based BSP-EGO (ℓBSP-EGO) [24] using binary trees to manage the sub-spaces. The binary decomposition is investigated with two variants of surrogates: global ones learned over the whole data set, and local models learned over only a subset of the data.

We conduct an extensive comparison between the two-variant proposed approach and several related ones. The contestant algorithms are chosen for their good performances and the fact that they adopt different acquisition strategies so we can analyze their impact on the optimization. The selected algorithms involve the simultaneous use of several Acquisition Functions (AFs), allowing to reduce the acquisition time cost but mostly increasing the batch effectiveness. In addition, multiple criteria are used either cooperatively or competitively. The cooperative way selects candidates finding trade-offs between the considered criteria, while the competitive way involves different criteria without (explicit) interaction. We propose a competitive AF-based algorithm inspired by the qEGO algorithm that illustrates the complementarity of the AFs. Trust region approaches that reduce the search space iteratively are also analyzed, as well as recursive space partitioning techniques. The investigation of the algorithms is performed on classical benchmark functions so that we can conduct exhaustive experimentation in a reasonable time. The experimental setup considers 5 benchmark functions executable in dimensions 6 and 12. Then, the best-performing algorithms are further investigated on another scheduling problem from electrical engineering that has 12 design variables. The results confirm that resorting to multiple AFs is beneficial for the optimization, and that splitting the search space is also very effective. As expected, higher dimensional problems are more time demanding and emphasize the lesser efficiency of some methods compared to multi-infill and partitioning ones.

This chapter starts with the description of the investigated strategies in Section 3.1 and the developed BSP framework in Section 3.2. Then, in Section 3.3 the benchmark analysis is conducted to assess for the efficiency of the different approaches. Finally, the best-performing algorithms are further investigated on the real-world Underground Pumped Hydro-Energy Storage (UPHES) management problem.

The work presented in this chapter relates to the following publications:

- *In academic journals*:
  - Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Batch Acquisition for Parallel Bayesian Optimization; Application to Hydro-Energy Storage

Systems Scheduling. *Algorithms* **15.** ISSN: 1999-4893. https://www.mdpi.com/1999-4893/15/12/446 (2022)

- *In conference proceedings, with peer reviewing and presentation*:
  - Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Adaptive Space Partitioning for Parallel Bayesian Optimization. in *HPCS 2020 - The 18th International Conference on High Performance Computing Simulation* (Barcelona / Virtual, Spain, 2021). https://hal.inria.fr/hal-03121209
  - Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Parallel Bayesian Optimization for Optimal Scheduling of Underground Pumped Hydro-Energy Storage Systems. in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2022), 790–797
- *And as abstracts and presentations in conferences*:
  - Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Towards Adaptive Space Partitioning for Large-scale Parallel Bayesian Optimization. in *OLA'2020 - International Conference on Optimization and Learning* (Cadix, Spain, 2020). https://hal.archives-ouvertes.fr/hal-02898960
  - Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Space Partitioning with multiple models for Parallel Bayesian Optimization. in *OLA 2021 - Optimization and Learning Algorithm* (Sicilia / Virtual, Italy, 2021). https://hal.archives-ouvertes.fr/hal-03324642

3.1 Improving the Scalability and the Batch Effectiveness

As already mentioned in Section 1.3, it might be difficult to choose the best AF for a given problem since none is consistently better [11]. Actually, it has been noticed that AFs performing well in small and high-dimensional spaces are not the same. For example, Rehbach *et al.* claim in [124] that the predicted value is a better AF than EI in some cases ($d > 10$). The variance provided by the GP model is less reliable when the dimension increases, misleading the search.

Many approaches use multiple criteria at the same time in order to maximize the effectiveness of the batch of candidates. The advantage is that it is easy to massively sample candidates, and we do not have to choose which AF to use [74]. However, it can result in a waste of budget. This question will be addressed in this chapter and we identify two ways of using multiple AFs in PBO. We differentiate APs operating either competitively or cooperatively whose common objective is to improve the batch effectiveness of the AP.

Actually, resorting to multiple AFs can also be useful regarding the scalability as it would not require to update the model as in qEGO. The decomposition of the design space into smaller sub-regions is another valuable alternative for improving the optimization process and its scalability since it allows the management of the exploration and exploitation trade-off while adding a level of parallelism to the algorithm. Indeed, in some cases, distinct sub-regions can be considered so that multiple APs can be conducted independently. The decomposition can also be used to re-center the design space around good regions so that it compensates for the over-exploration of classical BOAs [125].

3.1.1 Multi-Criteria Algorithms

Competitive approaches

In order to maximize the promisingness of a batch of candidates, competitive approaches optimize multiple AFs to independently select multiple candidates. Each AF is optimized using the same surrogate model, saving the partial updates compared to the previously used KB-qEGO algorithm. Consequently, the different AF optimizations can be conducted in parallel, limiting the increasing time required for the AP. To represent this class of algorithms, we propose Multi-Infill Criteria qEGO (MIC-qEGO). This new variant of qEGO is a combination of this latter and multi-infill approaches such as in De Palma *et al.* [14]. It has been observed that resorting to different AFs can impact favorably the objective value, especially when the batch size is high. Indeed, the relevance of candidates after several partial updates of $\mathcal{M}$ may be discussed.

The process is described in Algorithm 5. First, given a list of AFs, a number of candidates is attributed to each of them. For a given AF α, $n_{cand}[\alpha]$ gives the number of candidates selected according to α. A counter is initialized to 0 at line 1 and increments each time the algorithm adds a candidate to $\boldsymbol{X}_{new}$. The algorithm goes through the list of AFs and proceeds to the optimization of the latter if α is allocated a candidate, as indicated in lines 4 and 6 respectively. Each time a candidate is selected from the AF α, the number of selected candidates is incremented by 1 as shown in line 9 and the number of remaining candidates relative to α is decremented as

displayed in line 7. Afterward, if more candidates are needed, a surrogate update must be done in order to repeat the previous operations. Nevertheless, this intermediate step must remain fast to not penalize the overall time of the optimization. The temporary model is fitted with a smaller budget than the initial model. It is named *partial_fit* at line 12 and it is realized with the predicted value of the surrogate model, consistently with the KB heuristic. The loop continues until the algorithm completes the batch of candidates.

Algorithm 5 Acquisition Process of the Multi-Infill Criteria qEGO (MIC-qEGO) Algorithm

Input

$\mathcal{M}$: Surrogate model

n_{crit}: number of chosen AF

n_{cand}: vector of size n_{crit}

$n_{cand}[\alpha]$: number of candidates for AF α, $\sum n_{cand} = q$

1: $ct = 0$: initialize counter
2: $X_{new} = \{\},\ \boldsymbol{y}_{new} = \{\}$
3: **while** ct < q **do**
4: **for** α in AF list **do**
5: **if** $n_{cand}[\alpha] \neq 0$ **then**
6: $\boldsymbol{x}_{new} = argmax_{\mathcal{D}}(\alpha(\boldsymbol{x}), \mathcal{M})$
7: $n_{cand}[\alpha] \leftarrow n_{cand}[\alpha] - 1$
8: $X_{new}, \boldsymbol{y}_{new} = X_{new} \cup \boldsymbol{x}_{new}, \boldsymbol{y}_{new} \cup y_{PV}$
9: $ct \leftarrow ct + 1$
10: **end if**
11: **end for**
12: $\mathcal{M} \leftarrow$ partial_fit$(X \cup X_{new}, \boldsymbol{y} \cup \boldsymbol{y}_{new})$
13: **end while**
14: **return** $\boldsymbol{X}_{new}$

Cooperative approaches

With the same idea of resorting to multiple AFs to maximize the promisingness of the batch issued per cycle, another way to proceed is to consider a trade-off between all chosen candidates. To do so, we resort to multi-objective optimization[1] algorithms that provide a set of non-dominated solutions. This set is called the Pareto set. Loosely speaking, this set includes the best trade-off between all the objectives. Usually, there are no candidates that give the best possible outcome for every objective, and multiple choices can be made regarding the importance of the considered objectives. One advantage regarding the execution time is that we can choose as many candidates as needed with a single surrogate model. However, multi-objective optimization is generally much more time-consuming.

[1]Notes on multi-objective optimization are given in Appendix B

The Multi ACquisition Ensemble (MACE) algorithm [75] uses this principle to create a set of points resulting from the simultaneous maximization of EI, minimization of LCB, and maximization of PI. Then, q candidate points are randomly sampled from the Pareto set resulting from the multi-objective optimization. This set of AFs also appears to give good results in [13]. The multi-objective optimization of MACE is performed by NSGA-II [126]. Details about NSGA-II are provided in Appendix B.2. MACE has been experimented on 8 benchmark functions whose dimension varies between 2 and 10 and 2 real-world problems. The comparison involves several popular BOAs such as qEGO [17] and Local Penalization EGO with EI [69] and is conducted for a maximum batch size of 4 simulations per iteration. We then extend the study up to the acquisition of 32 points per cycle.

Algorithm 6 describes the operation of MACE for a user-defined set of AFs, designated by $\boldsymbol{\alpha}$ in line 6. After building the surrogate model at line 2 that is needed to evaluate the AFs, a multi-objective optimization is carried out using NSGA-II. The obtained Pareto set of line 4 represents the best trade-offs between all AFs. A subset of size q is sampled to create the batch of candidates required for the parallel evaluation of line 6.

Algorithm 6 Multi ACquisition Ensemble (MACE) Algorithm

Input

Ω: Design space
$\mathcal{D} = (X, \boldsymbol{y})$: initial data set
$\mathcal{M}$: surrogate model
q: batch size
$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)$: chosen AFs

1: **while** budget available **do**
2: $\quad \mathcal{M} = \mathcal{GP}(\mathcal{D})$
3: $\quad \mathcal{P} = \text{NSGA-II}(\boldsymbol{\alpha}, \mathcal{M})$
4: $\quad \mathcal{P}_{ps} = \text{Pareto\_set}(\mathcal{P})$
5: $\quad X_{new} = \text{random\_selection}(\mathcal{P}_{ps})$
6: $\quad \boldsymbol{y}_{new} = f(X_{new})$
7: $\quad \mathcal{D} = \mathcal{D} \cup (X_{new}, \boldsymbol{y}_{new})$
8: **end while**
9: **return** $\min_y \mathcal{D}$

The second considered approach for improving the batch effectiveness and scalability of BOAs is to split the design space into several smaller spaces in order to split the tasks and ensure diversity in the AP. The decomposition approach is detailed in the following section.

3.1.2 Space Partitioning in Optimization

Divide and conquer approaches

The *divide and conquer* strategy is quite common in optimization schemes. For example, the recursive decomposition of the design space has been extensively used in optimization algorithms such as DIRECT [122] and possesses the advantage of being easily parallelizable. Some other methods use the *divide and conquer* scheme differently such as Villanueva *et al.* [127], where an agent is allocated to a sub-domain and performs a surrogate-based optimization on its own. Sub-domains are created using k-mean clustering, they also have the possibility to merge or to be removed. Wang *et al.* [128] also use clustering techniques to create sub-regions into which we can zoom. Li *et al.* [129] proposed a decomposition approach based on principal component analysis and split until each sub-domain possesses approximately the same sample size to create local metamodels. Wang *et al.* [130] used a partitioning scheme inspired by DIRECT [131] to lower the computing cost and guide the optimization.

However, few works associate spatial decomposition with organized structures, such as space partitioning trees, and surrogate-based optimization. The only found reference is the Treed GP algorithm from Gramacy *et al.* [50, 51], developed in the context of non-stationary processes. In this case, local models are used to better approximate the landscape of the non-stationary objective function. We propose a new algorithm that is able to select large batches of points to evaluate in parallel in a moderate timing while keeping a balance between exploration and exploitation. We use a space partitioning managed by a self-organizing binary tree in order to perform simultaneously different local APs in each sub-domain. The purpose of the tree is to structure the design space and decompose the global AP into several ones. It also guides the optimization process using a decision heuristic dealing with where to intensify the decomposition process, and where to sample less frequently. This type of partitioning does not reduce the dimension of the problem, each sub-domain remains of dimension d. Dimension reduction methods like feature extraction or differential grouping are out of the scope of this thesis since we do not consider any additional property regarding the objective function. The developed algorithm is exhaustively described in Section 3.2.

Other approaches, more focused on higher dimensional problems, use trust regions. They represent a sub-space of the design space, where we choose to intensify the search. The primary idea is to compensate for the over-exploration of classical BO.

Trust Region-based acquisition

Trust region approaches consider a sub-region that characterises the best current outcome, plus a hyper-volume around it in which to intensify the search. It is especially valuable when the dimension increases and the models are less accurate, often overestimating the uncertainty, which results in misleading the sampling. Among them, differences exist in how and when to trigger the trust regions. For example, in TREGO [132], the algorithm operates at both local and global levels. If the global optimization fails, the local optimization is performed. Given $\boldsymbol{x}_k^*$,

the best current point at cycle k, the trust region is defined as:

$$\Omega_k = \{\boldsymbol{x} \in \Omega | d_{\min}\sigma_k \leq \parallel \boldsymbol{x} - \boldsymbol{x}_k^* \parallel \leq d_{\max}\sigma_k\},$$

where $d_{\min}$ and $d_{\max}$ are two strictly positive constants, and σ is a parameter increasing or decreasing according to success or not.

The TrUst Region Bayesian Optimization (TuRBO) algorithm [18] also uses a local strategy by performing local optimization in a trust region. It aims at compensating for the over-exploration resulting from a global acquisition process. The trust region is a hyper-rectangle centered on the best solution found so far. The edges of the hyper-rectangle are scaled according to the length scale from the GP model (see Equation 1.21). The side length for each dimension of the hyper-rectangle is scaled according to the length scale θ_i, while maintaining a total volume of L^d. The trust region evolves at each iteration, either decreasing its volume when the algorithm fails to improve the target for a certain number of iterations and needs more exploitation or, conversely, increasing it when more exploration is needed. Li *et al.* [133] propose the TRLBO algorithm, inspired from TuRBO, where the models are locally fitted inside the trust region. This results in a faster AP and faster algorithm.

Algorithm 7 TrUst Region Bayesian Optimization (TuRBO) Algorithm

Input

Ω: Design space
$\mathcal{D} = (X, \boldsymbol{y})$: initial data set
$\mathcal{M}$: surrogate model
q: batch size
$\mathcal{T}, L$: trust region and its length

1: **while** budget available **do**
2: $\quad \mathcal{M} = \mathcal{GP}(\mathcal{D})$
3: $\quad \boldsymbol{\theta} = \text{length\_scale}(\mathcal{M})$
4: $\quad \mathcal{T} = \boldsymbol{x}_{min} \pm L\boldsymbol{\theta}/2\,(\prod \boldsymbol{\theta})^{\frac{1}{d}}$ ▷ Total volume of L^d
5: $\quad X_{new} = argmax_{\mathcal{T}}(q\text{EI}(x))$
6: $\quad \boldsymbol{y}_{new} = f(X_{new})$
7: $\quad \mathcal{D} = \mathcal{D} \cup (X_{new}, \boldsymbol{y}_{new})$
8: $\quad \text{update\_length}(L)$
9: **end while**
10: **return** $\min_y \mathcal{D}$

The considered version of TuRBO for the following benchmark uses a single trust region, with the EI criterion, but can be generalized to k sub-regions characterized by their own trust region. Algorithm 7 displays the operation of TuRBO. The main difference with standard BO is the management of the trust region length, denoted L. It is used to compute the trust region $\mathcal{T}$ of line 4. The size of the trust region along the dimensions is proportional to the associated length scale from the covariance matrix of the model (line 3), giving a wider space to influential variables. Then, as indicated in line 5, the maximization of the qEI criterion is performed inside the trust region. The selected candidates are evaluated and integrated into the data set, and

the length of the trust region is updated according to the following rule: after τ_{succ} consecutive successes, $L \leftarrow \min(2L, L_{max})$; while after τ_{fail} consecutive failures, $L \leftarrow L/2$. If L becomes too small (*i.e.* if the algorithm has converged) the trust region is reset to the initial value. This feature is added to avoid repetitive sampling which could conduct to singular covariance matrices, but also to limit the traps of local optima. The single trust region algorithm is presented as TuRBO-1 in [18], where it gives good results on various problems (especially those with dimensions 12 and 14) as well as a good efficiency considering the batch sizes up to $n_{cores} = 64$.

3.1.3 A Taxonomy of Bayesian Optimization Algorithms

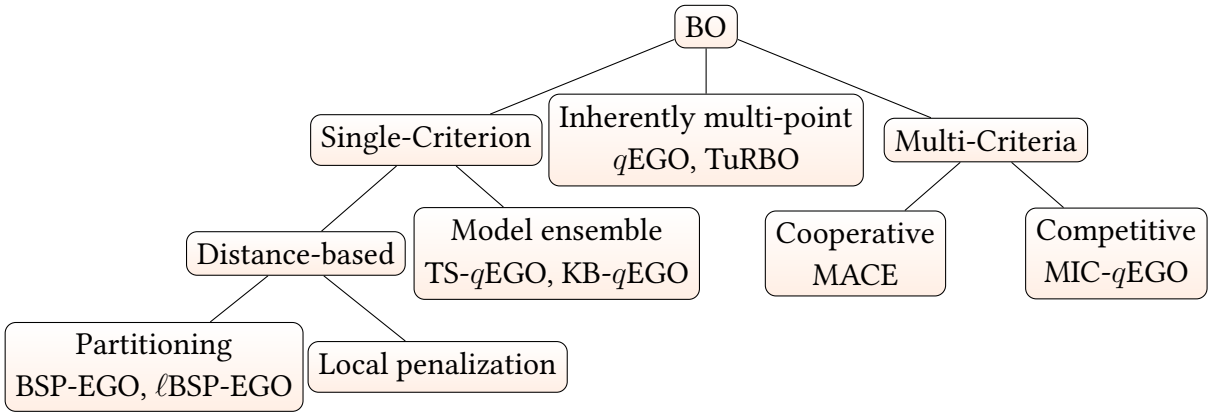


Figure 3.1: A taxonomy of batch-parallel Bayesian Optimization Algorithms

Figure 3.1 represents a taxonomy of existing methods in BO. We first distinguish the single-criterion approaches, the inherently multi-point criteria (*e.g.*, qEI) and their approximated forms, and the multi-criteria ones.

In the first set, the acquisition of new candidates requires to either impose a distance between candidates, or modify the model (or resort to several ones). The *distance-based* class includes methods using for instance space partitions, or local penalization, or even niching strategies (not displayed in Figure 3.1). The developed BSP-EGO and ℓBSP-EGO algorithms are representative of this latter class. Among methods using model ensembles, we can mention TS-qEGO, which samples candidates from different draws from the GP model, and KB-qEGO which resorts to partial updates of the surrogate model to change the landscape of the single-point AF. The single-criterion category also includes multi-model sampling methods, where each candidate is provided with the help of a different model.

qEGO and TuRBO are two algorithms representative of the second set but are quite different since TuRBO acts on trust regions, restricting the search space. The last mentioned set involves cooperative and competitive approaches. For instance, MACE is said cooperative since its AP considers simultaneously the different AFs, trying to find the best trade-off between them. On the other side, MIC-qEGO operates in a more competitive way by optimizing each criterion independently.

The algorithms are sorted with their main characteristic features. Each class is representative of a feature that is supposed to improve the BO framework, and that is being investigated

in the following. However, this taxonomy is obviously not exhaustive, and the classes are not mutually exclusive. For instance, ℓBSP-EGO uses local models, therefore an ensemble of models, but the main feature is the local domains generated through the partitioning of the design space. It does not take into account either the global or local aspect of the surrogate models which could be applied to any algorithm. Trust region algorithms such as TuRBO focus on a single sub-space while qEGO acts globally, yet they belong to the same class. Likewise, BSP-EGO and ℓBSP-EGO are mainly built with a partition structure but have two distinct ways of using the partition.

The developed algorithms, namely BSP-EGO and ℓBSP-EGO, are compared to different algorithms representative of the above classes in Section 3.3.

3.2 Binary Space Partitioning EGO (BSP-EGO)

Parallel versions of EGO-like algorithms often suffer from time-consuming APs or costly metamodel updates, and a major difficulty lies in balancing the optimization process [20]. Despite a fast improvement of the outcome at early stages of the optimization, we observe a tendency to over-exploration [125] and thus a stagnation at later stages [20]. We then propose to partition the design space into smaller sub-spaces in order to perform simultaneously different local APs in each sub-domain. The partitioning is managed by a self-organizing binary tree. It allows the decomposition of the global AP into several ones but also balances the optimization process by using a decision heuristic choosing where to intensify the decomposition process, and where to sample less frequently. However, the algorithm remains able to sample in any sub-region, continuing to explore according to a decision criterion to be defined. The strength of this method lies in its adaptability. Actually, it is able to provide as many candidates as needed, remains fast to execute, and is parallelizable (sub-APs are independent of each other). Finally, the partitioning tree is automatically adapted for the next cycle.

3.2.1 A New Acquisition Strategy for Large Batch Sizes

The particularity of this new AP is that it is decomposed into several sub-processes responsible for providing one candidate each. Let us suppose that the whole domain is Ω_1. The partitions are managed by a binary tree, where the root node (at tree level 1) contains Ω_1. The next level (level 2) has two nodes (node 2 and node 3) that contain Ω_2 and Ω_3 such that $\Omega_1 = \Omega_2 \cup \Omega_3$. For each node k, the property is respected such that $\Omega_k = \Omega_{2k} \cup \Omega_{2k+1}$ and $\Omega_{2k} \cap \Omega_{2k+1} = \emptyset$. Therefore, for each level of the tree, the union preserves the entire domain, without overlaps.

An example is presented in Figure 3.2, where the domain is split into four sub-domains. Let us call $\mathcal{F}_n$ the family of sub-domain indices at cycle n, such that $\bigcup_{k\in\mathcal{F}_n}, \ \Omega_k = \Omega$ and $\forall i, j \in \mathcal{F}_n, \ \Omega_i \cap \Omega_j = \emptyset$. Following the example from Figure 3.2, $\mathcal{F}_n = \{2, 6, 7\}$ and $\mathcal{F}_n = \{3, 4, 5\}$ are acceptable sets. Thanks to this kind of decomposition, it is easy to perform one sub-AP in each sub-domain, while keeping knowledge of the entire domain. As soon as all the candidates are selected, they are collected and sorted according to the chosen figure of merit. A subset of them, corresponding to the batch size q will be exactly evaluated while the rest is discarded.

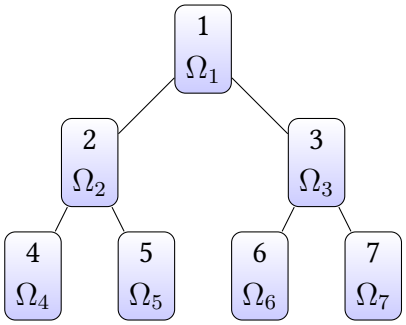


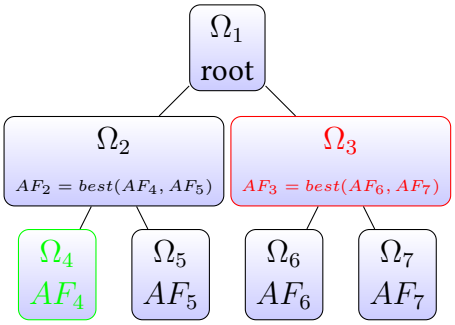
Figure 3.2: Partitioning of Ω through the binary tree

This strategy intends to reinforce the global aspect of optimization by continuing the sampling in *a priori* less interesting areas of Ω (from the AF point of view). Nevertheless, in order to avoid sampling with clearly no gain, and thus waste the computational budget, the batch size is smaller than the total number of candidates. Furthermore, to balance the exploration and exploitation processes, the tree evolves by splitting further the most promising nodes to intensify the search into the best sub-domain - always in terms of the chosen AF. Even though the sub-domains are distinct, it may happen that several candidates are very close to a shared boundary, and thus to each other. In that scenario, the candidates receive a small random perturbation so that the area is still sampled twice but not redundantly.

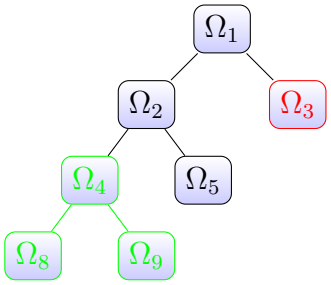
The number of candidates provided by the AP before the selection phase is chosen as a multiple of the batch size to balance the computational load between workers. Indeed, the batch size is fixed equal to the number of available cores (1 evaluation per core), thus each computing unit performs the same number of sub-APs. As stated in the previous paragraph, one candidate is chosen in each sub-domain, thus we have as many candidates as leaves in the tree. Let us call n_{leaves} that number, and still q the batch size. Consequently, $n_{leaves} = r * q$ where $r \in \mathbb{N}_{\setminus\{0\}}$.

The tree is updated once per cycle to take into account the new information. The supposed best sub-domain, according to the AF, is decomposed further to intensify the search in that area. Nevertheless, as we decide to keep n_{leaves} constant, this splitting step is only performed if two domains are merged. In terms of the BSP-tree, the leaf with the highest figure of merit is split, and the parent node with the lowest one loses its leaves to become a leaf itself. This process is illustrated in Figure 3.3.

This example directly follows the one from Figure 3.2, one candidate is chosen inside each leaf (*i.e.*, the ones indexed by $\mathcal{F}_0 = \{4, 5, 6, 7\}$). Each node is attributed the best figure of merit of its children, this number is denoted in Figure 3.3 by AF_{node}. In Figure 3.3a, Ω_4 possesses the best value among the nodes indexed by $\mathcal{F}_0$. Consequently, the node will be split if the merge operation can be performed. Regarding the parents of the leaves, Ω_3 possesses the worst value, meaning that the area does not need as much attention and thus it will be merged. Eventually, we end up with $\mathcal{F}_1 = \{3, 5, 8, 9\}$. In case of a non-allowed operation, the tree is kept identical for the next cycle. For instance, regarding Figure 3.3b, it may happen that for the next cycle Ω_3 is still the worst sub-domain and can't be merged with Ω_2. However, this kind of exception is relatively rare when dealing with large trees.



(a) Step 1: selection of node to be split or merged



(b) Step 2: operation allowed, the decomposition evolves

Figure 3.3: Illustration of one tree update

Without loss of generality, let us set $\Omega_1 = [0, 1]^d$. We must decide how to split the hypercube. For practical reasons, the choice is made for now to keep hyper-rectangular domains, so that a splitting operation is characterized by the axis/dimension to be split, and the range at which the section is done. For example, if $\Omega_1 = [0, 1]^2$ is split according to the first axis in the middle of the segment, it comes to $\Omega_2 = [0, 0.5] \times [0, 1]$ and $\Omega_3 = [0.5, 1] \times [0, 1]$. The heuristic for this study is arbitrary splitting: based on the idea that it is preferable to have dimensions of the same order of magnitude, axes are split one after another in a cyclic way. The chosen axis is determined by the level of depth of the tree node, *i.e.*, the axis along which a sub-domain is split is given by: $d_{split} = depth(node) \bmod(d)$. The initial tree is formed using this heuristic until it reaches the desired depth.

3.2.2 Global Model-based BSP-EGO

Named after the EGO algorithm, this method is called Binary Space Partitioning Efficient Global Optimization (BSP-EGO). It uses a global model learned over the whole data set, and the same surrogate model is used in each sub-region for the local AP.

BSP-EGO is outlined in Algorithm 8. The tree $\mathcal{T}$ is initialized at a predefined depth, deduced from the user-defined hyper-parameter n_{leaves}. At the beginning of each cycle, starting at line 3, the surrogate model is created before the AP begins. The global AP is composed of several sub-APs performed independently in each leaf of the tree, as indicated in lines 6 and 7. In each leaf of the tree, marked by $\mathcal{F}$, a candidate is proposed by maximization of the chosen AF. As each leaf is independent (non-overlapping), the local APs can be performed in parallel. The candidates are gathered and the q most promising ones, according to the chosen AF, are selected (line 9). Then, the tree is updated according to the previously described rule, and the leaves are indexed into $\mathcal{F}$. As for other BO algorithms, the cycle ends with the parallel evaluation of the selected candidates and their insertion into the data set as shown lines 11 and 12.

The present method still relies on a model learned over the whole data set and we have seen that it becomes quickly very time-consuming, especially in the context of time-constrained application. In addition, the evaluation of the tree described in Figure 3.3b requires pruning the tree by cutting the leaves associated with the worst parent. However, this also results in

Algorithm 8 Binary Space Partitioning Efficient Global Optimization (BSP-EGO) Algorithm

Input
f: objective function
Ω: design space
$\mathcal{D} = (X, \boldsymbol{y})$: initial data
d_{tree}: depth of the tree

$\mathcal{T} \leftarrow$ build_tree(d_{tree})
$\mathcal{F} \leftarrow$ get_leaves$(\mathcal{T})$
while budget available **do**
  $\mathcal{M} \leftarrow \mathcal{GP}(\mathcal{D})$
  $\mathcal{B} \leftarrow \emptyset$
  for $leaf$ in $\mathcal{F}$ **do** ▷ parallelizable loop
    $\mathcal{B} \leftarrow \mathcal{B} \cup \arg\max_{\Omega_{leaf}}(\mathcal{M}, \alpha(x))$
  end for
  $X_{new} \leftarrow$ selection$(\mathcal{B})$
  $(\mathcal{T}, \mathcal{F}) \leftarrow$ update_tree$(\mathcal{T}, \mathcal{B}, \mathcal{F})$
  $\boldsymbol{y}_{new} = f(X_{new})$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{X_{new}, \boldsymbol{y}_{new}\}$
end while
return $\min_y \mathcal{D}$

limiting the maximum depth of the tree. Depending on the evolution of the tree and its depth, it can be quite limiting in terms of intensification. For the latter reasons, we develop another version of BSP-EGO that involves local models, and a deeper exploration of the tree.

3.2.3 Local Model-based BSP-EGO Variant

A second approach, named Local-model Binary Space Partitioning Efficient Global Optimization (ℓBSP-EGO), is developed to select a batch of candidates using local models. The local surrogate models are fitted on a subset of points selected from $\mathcal{D}$. As a result, ℓBSP-EGO offers a way to control the time allocated to the metamodel fitting by operating on a subset of data, in a local way. Indeed, the model fitting and the AP can also be computationally expensive. Adjusting the size of the subset helps to better fit the time constraint. It speeds up the model fitting step, and the local models can be fit in parallel, prior to the local APs.

The local data set is chosen according to the distance from the center of the sub-region. The n_{learn} points the closest are chosen, even if they lie outside of the considered sub-region. This basic clustering technique seems to be a reasonable choice since the GP predictor is mainly influenced by the value of the closest points. Consequently, the parallelization does not only concern the evaluations but also the whole AP, including the model fitting (contrary to BSP-EGO). Being able to adjust the learning time of the model and acting locally allows us to perform

many more optimization cycles. However, the ratio between the time allocated to the AP compared to the evaluation time must be tuned carefully. Fast AP and model training allows one to sample more, but the promisingness of each new candidate might be lower.

BSP-EGO relies on the EI to sort the leaves and decide which one is more valuable than the others. However, in the context of local models, the EI coming from different models is less relevant. For instance, one might preferably rely on the predicted value or on the LCB. In addition, and in order to promote intensification, the number of leaves is not kept constant. The tree still evolves at each cycle, but performs only the splitting step allowing it to dig much deeper and improve the intensification. Of course, it implies making one more choice about which leaf (*i.e.* sub-region) to activate. Indeed, exploring all the leaves would be excessively time-consuming if the tree becomes large, which is expected. The number of selected leaves is ideally a multiple of the number of processing units, which is also usually equal to the batch size q, to balance the computational load. For both the selection strategy of the leaves and the evolution of the tree, a choice is made between two options: either relying on the best outcome of the leaf (*i.e.* the best objective value) or on the best potential improvement determined by the chosen AF value of the previous cycle. Since the leaves are not all activated at each cycle, it requires to keep track of each AP conducted in previous cycles. A dynamic strategy is adopted regarding this choice: the probability is computed according to the remaining time budget. The probability of selecting the AF as leaf selection criteria is given by $p = 1 - t_{current}/t_{total}$. At the beginning of the search, exploration is favored by giving more probability to select the AF option, while at the end the probability is reversed favoring exploitation by using the objective value. Furthermore, with a fixed low probability the criterion might be chosen as the index of the leaf, forcing the exploration of large regions - a low index indicates bigger sub-regions.

Algorithm 9 shows the important steps of ℓBSP-EGO. First, the leaves to be activated need to be chosen. According to the dynamic criterion described above and named α in line 4, the most promising leaves are activated as shown at line 5. For each selected leaf of the binary tree, a GP model is fitted with the n_{learn} points closest to the center of the sub-domain. The model is then used to proceed to the local AP using the LCB. The previous steps are described from lines 7 to 10. The remaining operations involved in ℓBSP-EGO, described between lines 12 and 15 are similar to the ones of BSP-EGO. Among the gathered batch of candidates, q ones are selected to be simulated in parallel and the tree is updated.

Even though not the primary objective in our context, resorting to local models also contributes to dealing with non-stationarity in the search space [11, 51].

3.2.4 Software Implementation and Packaging

All the implementations of the methods are based on `GPyTorch` [96] and `BOTorch` [97]. The surrogate models are built using `GPyTorch` and `BOTorch` is used for all that relates to the APs. The implementations of TuRBO[2] and MACE[3] are extracted and adapted from their respective GitHub repositories. MIC-qEGO, BSP-EGO, and ℓBSP-EGO are our own implementations since they are our own contributions.

[2] https://botorch.org/tutorials/turbo_1
[3] https://github.com/Alaya-in-Matrix/pyMACE

Algorithm 9 Local-model Binary Space Partitioning Efficient Global Optimization (ℓBSP-EGO) Algorithm

Input

f: objective function
Ω: design space
$\mathcal{D} = (X, \boldsymbol{y})$: initial data
d_{tree}: depth of the tree
n_{learn}: size of the learning sample

$\mathcal{T} \leftarrow$ build_tree(d_{tree})
$\mathcal{F} \leftarrow$ get_leaves$(\mathcal{T})$
while $t_{current} < t_{total}$ **do**
  $\alpha \leftarrow \alpha$(remaining_budget) ▷ dynamic criterion
  $\mathcal{L} \leftarrow$ select_leaves$(\mathcal{F}, \alpha)$
  $\mathcal{B} \leftarrow \emptyset$
  for $leaf$ in $\mathcal{L}$ **do** ▷ parallelizable loop
    $\mathcal{D}_{leaf} =$ create_subsets$(\mathcal{D}, leaf)$
    $\mathcal{M}_{leaf} \leftarrow \mathcal{GP}(\mathcal{D}_{leaf})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \arg\max_{\Omega_{leaf}}(\mathcal{M}_{leaf}, \text{LCB})$
  end for
  $X_{new} \leftarrow$ selection$(\mathcal{B})$
  $(\mathcal{T}, \mathcal{F}) \leftarrow$ update_tree$(\mathcal{T}, \mathcal{B}, \mathcal{F})$
  $\boldsymbol{y}_{new} = f(X_{new})$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{X_{new}, \boldsymbol{y}_{new}\}$
end while
return $\min_y \mathcal{D}$

The source code of the whole experimental framework is available on GitHub at this address:

https://github.com/MaGbrt/pySBO.git.

It is built upon the pySBO[4] library [134] from which the mentioned GitHub repository is a fork. The repository includes all the experiments conducted within the scope of the thesis.

The tree-based algorithms were initially implemented in C++ using the BayesOpt [135] library, before being ported to Python. Consequently, some differences might appear when comparing with the related publications.

3.3 Benchmarking BSP-EGO against state-of-the-art BOAs

The simulators used in Chapter 2 are subject to software dependencies or proprietary software making large-scale experiments impossible and difficult to replicate. In addition, the time cost implied by the real-world simulators makes it difficult to reach higher simulation budgets. For the aforementioned reasons, the choice is made to experiment with benchmark functions.

The main objective of this section is to investigate the two identified leads to improve the performances of BOAs. Mostly we confront the developed algorithms BSP-EGO and ℓBSP-EGO to state-of-the-art algorithms and relate the most valuable features of the considered BO algorithms and their field of application. The study identifies some remaining limitations and possible improvements for future research directions.

3.3.1 Objective and Experimental Framework

Objective of the study

In this study, we investigate our two proposed algorithms leveraging a recursive decomposition of the search space, namely BSP-EGO and ℓBSP-EGO. Along with the proposed new approaches, the investigation involves several recently designed well-performing algorithms from the literature. In accordance with the established taxonomy shown in Figure 3.1, we classify them into two categories of methods. The first one concerns the multi-criteria methods, such as MIC-qEGO and MACE. The second one involves space-partitioning methods such as BSP-EGO and ℓBSP-EGO to which we add TuRBO, which also deals with the design space. We aim to assess the gain of such algorithms regarding the identified low scalability and low batch effectiveness of classical BO algorithms.

To do so, the following algorithms are selected for investigation:

- MIC-qEGO and MACE, presented in Algorithm 5 and Algorithm 6 respectively, are used in this study to represent the multi-criteria approaches. MIC-qEGO embodies the competitive way and MACE the cooperative way of dealing with multiple criteria.

[4] https://pysbo.readthedocs.io

- BSP-EGO, ℓBSP-EGO, and TuRBO, presented in Algorithm 8, Algorithm 9 and Algorithm 7 respectively are chosen to represent the partitioning approach. Both tree-based algorithms are representative of recursive decomposition-based approaches, while TuRBO embodies the trust region methods.

Baseline algorithms

In addition to the five previously mentioned algorithms, we add three frequently used BO algorithms. First, qEGO is used with the multi-point EI. To speed up the AP, the reparameterization trick [136] is used with a fast Monte Carlo approximation of the qEI surface. Second, we retain the KB-qEGO algorithm, already described in Algorithm 2, which offers another alternative to the time-consuming qEI optimization. And last, we add TS-qEGO, which relies on Thompson Sampling (TS) to propose the batch of q candidates. TS samples from the posterior distribution and minimizes the sampled GP so that it gives the best point to evaluate, according to that posterior sample. It is a fast and scalable method for parallel BO [137].

Potential of improvement

Our investigation focuses on two aspects:

- The performance of the method in terms of the outcome in proportion to the batch size, *i.e.*, the batch effectiveness. Ideally, the method obtains equivalent outcome quality for an equivalent number of simulations, whatever the batch size is. Thus, for a given number of cycles, we achieve better results when increasing the batch size.
- The scalability of the method, studied with the number of total cycles/simulations performed in the fixed time. The expected ideal behavior is that the time cost arising from optimization methods (outside simulation time) remains short enough not to hamper the optimization. Therefore, increasing n_{cores} also increases the number of simulations by the end of the time budget. Additionally, if the previous point is respected, it should also improve the quality of the final result.

3.3.2 Experimental Protocol

Benchmarking procedure

For the same reasons put forward in Section 2.3.3, we rely on a time budget rather than on the number of evaluations or cycles. This aims at visualizing the impact of the AP time in the optimization process and better representing the time constraints of a given optimization problem. With the very small time cost of the benchmark functions, a 20-minute budget is considered as it is found to be sufficient to perform a large enough number of cycles for any algorithm to observe significant divergences between them.

The test-bed includes 5 benchmark functions representative of known problems in optimization (*e.g.*, flat regions, several local optima, noisy landscape, etc.). Details about the functions are given in Appendix C.1. The analysis is conducted in dimensions 6, and 12 so that we can observe the impact of the dimension on the search strategies. Each algorithm performs 20 distinct optimization runs, on each benchmark function, for the following batch sizes $q \in \{2, 4, 8, 16, 32\}$. For each of the 20 runs of each function, an initial sample is randomly generated with n_{init} design points according to the dimension. The same initial set is used for all batch sizes and every method.

The whole experimental setup represents $\#dimensions \times \#batch\_sizes \times \#functions \times \#algorithms \times n_{rep} = 8000$ optimization runs. Each run lasts 20 minutes, this is equivalent to approximately 112 days of computation in a single machine (assuming the number of cores is at least q). This type of benchmark is only made possible with the help of the Grid5000[5] computational test-bed [5].

As stated in [138], it is generally preferable to begin with a small sample to achieve the best performances. It matches our observations that with a fast-increasing model fitting time, the initial design should remain restricted. For this reason, we adopt the quite common rule of considering an initial design of size $n_{init} \approx 10 \times d$, where d is the dimension. Therefore, the initial size is set to 64 and 128 respectively for dimensions 6 and 12, consistently with the batch sizes in a parallel execution perspective. Indeed, 64 and 128 are multiple of any $q \in \{2, 4, 8, 16, 32\}$. The global experimental setup is summarized in Table 3.1.

Table 3.1: Summary of the experimental setup

Dimension	$d \in \{6, 12\}$
Batch size	$q \in \{2, 4, 8, 16, 32\}$
Functions	Rosenbrock, Ackley, Schwefel, Alpine02, Rastrigin
Algorithms	qEGO, TS-qEGO, KB-qEGO, MIC-qEGO, MACE, BSP-EGO, ℓBSP-EGO
Number of repetitions	$n_{rep} = 20$

Hyper-parameter Settings

For all the algorithms of this study, the surrogate model is a GP model. The trend is assumed constant but unknown, the covariance kernel follows the Matérn $\frac{5}{2}$ model and is fitted through MLE. The model is considered with a homoskedastic noise level and the kernel is fit with automatic relevance determination [139]. The way to fit the GP model can be seen as a hyper-parameter as well since it is an implementation choice, and it impacts the behavior of the algorithms, especially in a time-constrained context. As it is not investigated here, an identical routine is used for all methods for a fair comparison. Those parameters are identified from previous works and literature as good default choices [7, 97].

The training set is composed of all the available data, except for ℓBSP-EGO where $n_{learn} = \min(128; |\mathcal{D}|)$. MIC-$q$EGO uses 2 criteria, namely EI and LCB. This choice is arbitrary and could be improved by investigating the impact of the set of AFs. However, it is found to give good

[5] https://www.grid5000.fr

results in preceding experiments. As for MACE, the set from the original paper is used, it is composed of PI, EI, and LCB. TuRBO uses the same AP as qEGO, but in a trust region. Only one trust region is considered in this study. BSP-EGO uses $n_{leaves} = 4 \times q$ that are all explored by an AP so that a selection can be done between all the gathered candidates. In ℓBSP-EGO, the number of leaves of the binary tree is initialized to $n_{leaves} = 2 \times q$, but is not limited. However, the number of considered leaves per cycle remains constant after the selection phase (see Algorithm 9) and is equal to q. Those parameters allow a fast training of the local models while ensuring local accuracy.

3.3.3 Results and Analysis

We analyze the results regarding the following three aspects. First, we evaluate the performance of each algorithm regarding the outcome for different timestamps. Then, we study the scalability in terms of number of performed evaluations within the time budget. Finally, we assess the batch effectiveness.

Overall observations on the outcomes

Table 3.2 and Table 3.3 present a synthetic overview of the results. For each algorithm and batch size, we save the outcome for given timestamps $(0, 30, 60, 120, 300, 600, 900, 1200)$. The outcome for each function is scaled between 0 and 1, where 0 indicates the best possible outcome and 1 the initial best sample (common to all runs). A score between 0 and 1 is then obtained for each function that allows to aggregate the 5 benchmark functions into a single indicator by averaging the scores of each function (for all algorithms, batch sizes, and timestamps). The best values, close to 0, are highlighted in blue shades for a fast visual identification of the well-performing methods.

First, looking at Table 3.2 displaying the **results for the 6d-benchmark** functions, we can see a clear dominance of the investigated approaches compared to the baseline algorithms. Consistently with what was observed in previous sections, qEGO and KB-qEGO do not achieve better performances in terms of final outcome with higher computational power. The ideal batch size for these algorithms is between 4 or 8 simultaneous candidates. In addition, we observe a fast improvement at the beginning, but after 300 seconds it is very limited. Among the three mentioned algorithms, KB-qEGO seems to outperform the two others after 180 seconds despite its lower number of simulations due to the time-consuming AP, as indicated in Figure 3.4.

The **multi-infill approaches** show much better behavior as time increases. Indeed, MIC-qEGO shows considerable improvement compared to KB-qEGO when the time budget is higher. However, the best observed average outcome is for $q = 8$, which still indicates a bad batch effectiveness, and/or a bad scalability. Regarding MACE, the algorithm is outperformed for very low budgets but appears to have a much better batch effectiveness than MIC-qEGO since the outcome improves with both time and q. However, MIC-qEGO is better or equivalent on average with batch sizes 4 or 8 than any batch size and time of the MACE algorithm.

Table 3.2: Scaled outcome averaged over the 5 benchmark 6d-functions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	Timestamps / q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.5659	0.5052	0.4655	0.4498	0.4149	0.4034	0.3898	0.3839
	4	1	0.5556	0.4919	0.4562	0.4451	0.4302	0.4164	0.4074	0.4018
	8	1	0.5207	0.4742	0.4483	0.4384	0.4304	0.4185	0.4121	0.4114
	16	1	0.5486	0.4967	0.4668	0.4614	0.4472	0.4371	0.4320	0.4285
	32	1	0.5748	0.5420	0.5109	0.5026	0.4914	0.4736	0.4601	0.4501
TS-qEGO	2	1	0.7561	0.6801	0.6114	0.5717	0.5390	0.4967	0.4721	0.4517
	4	1	0.6788	0.6077	0.5445	0.5271	0.4937	0.4669	0.4588	0.4437
	8	1	0.6486	0.5617	0.5233	0.5039	0.4791	0.4462	0.4332	0.4289
	16	1	0.6008	0.5411	0.5029	0.4786	0.4622	0.4424	0.4294	0.4247
	32	1	0.5559	0.5094	0.4838	0.4665	0.4566	0.4446	0.4254	0.4191
KB-qEGO	2	1	0.5403	0.4629	0.4004	0.3765	0.3542	0.3244	0.2950	0.2926
	4	1	0.5350	0.4545	0.3912	0.3638	0.3407	0.3057	0.2979	0.2892
	8	1	0.5890	0.5183	0.4376	0.4044	0.3797	0.3422	0.3250	0.3185
	16	1	0.6600	0.5427	0.4789	0.4410	0.3981	0.3655	0.3556	0.3504
	32	1	0.7177	0.6098	0.5122	0.4806	0.4571	0.4244	0.4088	0.4040
MIC-qEGO	2	1	0.5386	0.4407	0.3827	0.3647	0.3400	0.3102	0.2929	0.2854
	4	1	0.4457	0.3646	0.3118	0.2864	0.2486	0.2253	0.2187	0.2085
	8	1	0.4762	0.3715	0.3110	0.2812	0.2611	0.2360	0.2266	0.2184
	16	1	0.5785	0.4370	0.3457	0.3276	0.3079	0.2698	0.2599	0.2521
	32	1	0.6467	0.5055	0.3788	0.3397	0.3032	0.2818	0.2675	0.2611
MACE	2	1	0.7560	0.6565	0.5670	0.4999	0.4089	0.3229	0.2887	0.2768
	4	1	0.6678	0.5723	0.4787	0.4175	0.3353	0.2870	0.2677	0.2518
	8	1	0.6146	0.5070	0.3805	0.3279	0.3033	0.2644	0.2449	0.2334
	16	1	0.5741	0.4577	0.3401	0.3025	0.2793	0.2460	0.2383	0.2328
	32	1	0.4537	0.3268	0.2803	0.2737	0.2603	0.2465	0.2340	0.2193
BSP-EGO	2	1	0.4825	0.3838	0.3171	0.2817	0.2618	0.2389	0.2317	0.2293
	4	1	0.4491	0.3654	0.3184	0.2944	0.2768	0.2500	0.2399	0.2349
	8	1	0.4133	0.3304	0.2832	0.2717	0.2509	0.2336	0.2245	0.2199
	16	1	0.4315	0.3592	0.3075	0.2871	0.2633	0.2466	0.2370	0.2318
	32	1	0.5083	0.4426	0.3661	0.3330	0.3065	0.2584	0.2444	0.2304
ℓBSP-EGO	2	1	0.3888	0.2964	0.2801	0.2794	0.2783	0.2776	0.2775	0.2771
	4	1	0.3350	0.2589	0.2420	0.2417	0.2413	0.2411	0.2410	0.2410
	8	1	0.3272	0.2408	0.2102	0.2027	0.1968	0.1945	0.1930	0.1930
	16	1	0.3415	0.2366	0.1952	0.1872	0.1765	0.1475	0.1453	0.1450
	32	1	0.4156	0.2974	0.2091	0.1831	0.1523	0.1350	0.1296	0.1232
TuRBO	2	1	0.3853	0.3439	0.3088	0.2904	0.2612	0.2547	0.2467	0.2416
	4	1	0.3586	0.3202	0.3018	0.2775	0.2546	0.2240	0.2111	0.2045
	8	1	0.3480	0.3207	0.3035	0.2997	0.2818	0.2510	0.2441	0.2388
	16	1	0.3260	0.3000	0.2847	0.2726	0.2616	0.2526	0.2349	0.2317
	32	1	0.3353	0.2772	0.2574	0.2371	0.2303	0.2062	0.1952	0.1879

Table 3.3: Scaled outcome averaged over the 5 benchmark 12d-functions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.6765	0.6587	0.6355	0.6275	0.6205	0.6121	0.6065	0.6021
	4	1	0.6724	0.6560	0.6413	0.6320	0.6204	0.6120	0.6075	0.6033
	8	1	0.6740	0.6584	0.6468	0.6311	0.6283	0.6221	0.6127	0.6073
	16	1	0.6763	0.6688	0.6563	0.6514	0.6416	0.6287	0.6178	0.6163
	32	1	0.6639	0.6596	0.6502	0.6419	0.6366	0.6280	0.6279	0.6264
TS-qEGO	2	1	0.7616	0.7379	0.7080	0.6899	0.6771	0.6584	0.6472	0.6439
	4	1	0.7493	0.7198	0.6924	0.6843	0.6720	0.6567	0.6531	0.6476
	8	1	0.7258	0.7112	0.6901	0.6801	0.6702	0.6581	0.6530	0.6495
	16	1	0.7259	0.7090	0.6916	0.6848	0.6758	0.6642	0.6549	0.6519
	32	1	0.7081	0.6948	0.6739	0.6662	0.6601	0.6517	0.6468	0.6439
KB-qEGO	2	1	0.6781	0.6535	0.6254	0.6156	0.5970	0.5890	0.5841	0.5769
	4	1	0.6868	0.6666	0.6368	0.6235	0.5997	0.5835	0.5819	0.5800
	8	1	0.6995	0.6814	0.6610	0.6517	0.6360	0.6233	0.6154	0.6112
	16	1	0.7085	0.6855	0.6735	0.6604	0.6509	0.6307	0.6266	0.6238
	32	1	0.7103	0.7062	0.6971	0.6887	0.6738	0.6637	0.6566	0.6482
MIC-qEGO	2	1	0.6776	0.6401	0.6180	0.6045	0.5910	0.5815	0.5788	0.5763
	4	1	0.6117	0.5587	0.5216	0.5007	0.4845	0.4629	0.4512	0.4465
	8	1	0.6138	0.5608	0.5193	0.5048	0.4859	0.4588	0.4442	0.4386
	16	1	0.6594	0.5924	0.5576	0.5383	0.5164	0.4923	0.4776	0.4712
	32	1	0.7037	0.6389	0.5936	0.5726	0.5552	0.5352	0.5264	0.5161
MACE	2	1	0.7217	0.6876	0.6235	0.5804	0.5475	0.5097	0.4792	0.4656
	4	1	0.6847	0.6467	0.5904	0.5629	0.5289	0.4935	0.4778	0.4707
	8	1	0.6539	0.5868	0.5496	0.5311	0.5060	0.4767	0.4567	0.4511
	16	1	0.6222	0.5624	0.5206	0.5024	0.4734	0.4337	0.4159	0.3968
	32	1	0.5969	0.5418	0.5057	0.4756	0.4475	0.4001	0.3768	0.3604
BSP-EGO	2	1	0.6357	0.5781	0.5464	0.5266	0.5063	0.4952	0.4860	0.4846
	4	1	0.6030	0.5761	0.5324	0.5105	0.4940	0.4729	0.4631	0.4564
	8	1	0.5838	0.5354	0.5012	0.4871	0.4642	0.4445	0.4342	0.4312
	16	1	0.5941	0.5559	0.5204	0.4987	0.4880	0.4575	0.4463	0.4393
	32	1	0.6384	0.6004	0.5723	0.5552	0.5419	0.5078	0.4945	0.4898
ℓBSP-EGO	2	1	0.5347	0.4407	0.3666	0.3298	0.3129	0.3105	0.3095	0.3092
	4	1	0.4896	0.3848	0.3137	0.2880	0.2705	0.2671	0.2660	0.2651
	8	1	0.4676	0.3857	0.3144	0.2821	0.2536	0.2378	0.2347	0.2337
	16	1	0.4857	0.4023	0.3405	0.3170	0.2857	0.2527	0.2398	0.2276
	32	1	0.5494	0.4674	0.4045	0.3733	0.3432	0.3096	0.2910	0.2823
TuRBO	2	1	0.5316	0.4421	0.3694	0.3460	0.3272	0.3154	0.3107	0.3050
	4	1	0.4911	0.4142	0.3586	0.3390	0.3196	0.3099	0.3064	0.3044
	8	1	0.4715	0.3983	0.3518	0.3395	0.3310	0.3228	0.3215	0.3203
	16	1	0.4426	0.3781	0.3458	0.3380	0.3322	0.3259	0.3247	0.3230
	32	1	0.5297	0.4289	0.3608	0.3365	0.3238	0.3131	0.3117	0.3112

Partitioning the search space into sub-regions also appears to improve the overall performances of qEGO-like algorithms. Indeed, BSP-EGO shows an average performance equivalent to that of MIC-qEGO while being slightly more effective regarding the batch acquisition. However, the outcome for each time stamp is very similar across the batch size, which also indicates a limited batch effectiveness. ℓBSP-EGO presents better performances than all the baseline algorithms. However, for low batch sizes ($q \leq 4$), BSP-EGO, MIC-qEGO, and TuRBO show equivalent or better outcomes. ℓBSP-EGO appears to be very effective in its ability to improve the outcome with the batch size. Indeed, except before 120 seconds, increasing the batch sizes always improves the outcome for any timestamp. The best average performance of the whole benchmark is realized for ℓBSP-EGO with $q = 32$. TuRBO shows a very good sample efficiency, with a low time budget (<2 minutes) the average outcomes are equivalent to the final outcomes of KB-qEGO. TuRBO also shows a good batch effectiveness, since for a given time, increasing the batch size improves the outcome. The best outcome achieved by TuRBO is found for $q = 32$, nevertheless, for $q = 8$ and $q = 16$ the performance is lower than for $q = 4$.

Increasing the dimension to 12 shows radically different outcomes as reported in Table 3.3. The investigated algorithms still perform better than the baseline, but a clear advantage can be noticed for the methods that intensify a lot: TuRBO and ℓBSP-EGO. Even though apparently the best option, ℓBSP-EGO performs best with $q = 8$ or $q = 16$ instead of $q = 32$ as for the 6d case. Regarding TuRBO, after a fast improvement at the beginning, all the indices beyond 180 seconds of execution remain similar, for all batch sizes. Similarly to the 6d-benchmark, MIC-qEGO and BSP-EGO have close average performances. However, a difference appears with MACE when the budget increases. In the same way as the 6d-benchmark MACE shows an improved outcome when increasing q for a given timestamp. This allows MACE to outperform MIC-qEGO and BSP-EGO with large batch sizes ($q \geq 16$) even for restricted time budgets (starting from 120 seconds).

As we will see in the following, the benchmark functions present very different landscapes, and sometimes a large batch size that increases the exploration of the design space is to be preferred. At the same time, other functions could require more intensification and small batches would be preferable. Then, increasing the batch size could have an opposite effect on the functions, and averaging the results into a single score does not account for this. To report on the latter, the functions are analyzed separately in the incoming section. But beforehand, let us look closer at the scalability and batch effectiveness of the investigated approaches.

Observations on the scalability

Figure 3.4 and Figure 3.5 depict the average number of simulations in the 20-minute budget according to the batch size for respectively the 6d and 12d-Alpine02 functions. We only show the Alpine 02 function since the same observations can be done for any other one.

Looking at the number of performed simulations displayed in Figure 3.4 and Figure 3.5, the main difference between the results in dimensions 6 and 12 lies in the number of simulations. The model being more complex in higher dimensions, it results in a loss in terms of number of simulations. However, the following observations are valid in both 6 and 12 dimensions.

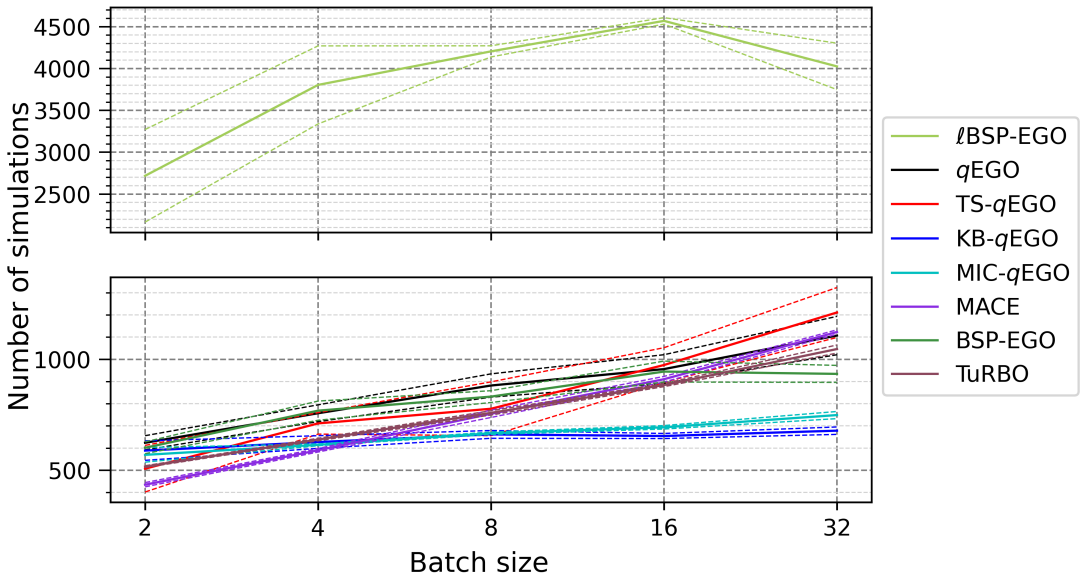


Figure 3.4: Average number of simulations according to the batch size for the 6d-Alpine02 function. Dashed-lines indicates the standard deviation over the 20 repetitions.

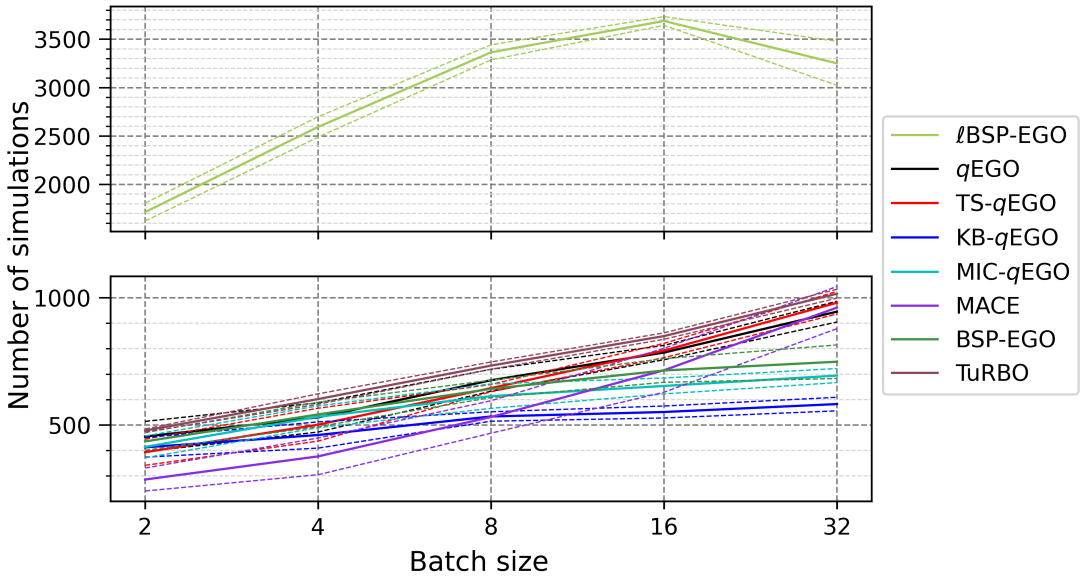


Figure 3.5: Average number of simulations according to the batch size for the 12d-Alpine02 function. Dashed-lines indicates the standard deviation over the 20 repetitions.

We observe a clear difference between ℓBSP-EGO, acting locally, and other global methods. We distinguish three groups that are mainly determined by their AP. Firstly, ℓBSP-EGO uses local models and parallel AP, which explains its time efficiency even for large budgets. ℓBSP-EGO achieves around 4 times more simulations than other methods. However, we can see that its scalability decreases with the batch size. The learning time and acquisition time are controlled by the learning size n_{learn} and by the number of exploited sub-regions, but the per-cycle time still increases. The most probable reasons for this are the computational costs associated with the tree management and the setting of the subset of points for the local models. Indeed, in ℓBSP-EGO the tree only splits the design space into smaller sub-regions, without merging non-promising ones as in BSP-EGO. The selection of the activated leaves and the memory access to the data can become time-consuming for very large trees. In addition, the local models are built with subsets assembled according to a distance-based scheme which can become computationally intensive for large data sets. A more precise analysis of the time of each phase of the algorithm should be conducted to draw a more accurate conclusion. Be that as it may, the good performances of the algorithm with respect to the time budget are directly linked to its good scalability. We can see at the beginning that ℓBSP-EGO is not significantly better than other algorithms, but when others stop improving due to their time-consuming execution, ℓBSP-EGO still manages to improve the outcome.

Secondly, the second set of algorithms is composed of qEGO, TS-qEGO, and MACE. They use a global model and a time-efficient AP. A quite important difference is noted with MACE when increasing the batch size, its scalability is less impacted than others thanks to its AP that has the same cost whatever the batch size is. Indeed, the multi-objective optimization returns a Pareto front on which the algorithm chooses q samples. The main limiting factor is the global surrogate model that takes the most time in the cycles of the algorithm.

Lastly, the heuristic-based APs (KB-qEGO and MIC-qEGO) are the least efficient in terms of number of simulations. Similarly to other qEGO-based algorithms, they use a global model, but they also need a partial update for each new candidate in a batch. Even though the MIC-qEGO algorithm spares some updates by resorting to several criteria, the gain is not significant, and when the batch size exceeds $q = 8$ the number of simulations stagnates. BSP-EGO is between the two groups since it operates in parallel and thus limits the acquisition time, but a larger number of local APs is performed. This makes BSP-EGO less efficient than the second group, but still faster than the last one.

On the other hand, the higher number of simulations is not always synonymous with better outcomes. The most evident example is TS-qEGO which is quite efficient in terms of number of simulations, and nonetheless performs poorly in comparison to the other approaches regarding the outcome. In the following, we try to refine the analysis by looking individually at the benchmark function. We identify the strengths and weaknesses of each algorithm by associating their performances with the known difficulties of the different benchmark functions.

Observations on the batch effectiveness

The batch effectiveness is not easily interpreted with this kind of presentation of the results. In this situation, we should rather talk about time efficiency since we are interested in the outcome for a given time instead of a given number of simulations. Nevertheless, we can see that for a

given time stamp, increasing the batch size does not mean improving the outcome. Actually, for algorithms such as MIC-qEGO or BSP-EGO it is preferable to perform more cycles with small batch sizes than to increase the batch size (even if it allows more simulations). The only algorithm that always profits from larger batches is MACE, but it is generally equivalent to or worse than TuRBO or ℓBSP-EGO in terms of objective value.

TuRBO achieves quite similar outcomes, whatever the batch size, except for the 6d-functions, where $q = 32$ gives the best results at almost any time. ℓBSP-EGO manages to better profit from a larger batch size only if the time budget is sufficient, *i.e.* ≥ 180 seconds at least for the 6d-benchmark, and 300 seconds for the 12d-one. However, in 12d the best outcome is achieved for $q = 16$ and not $q = 32$. The strength of ℓBSP-EGO lies in its scalability rather than in the batch effectiveness.

3.3.4 Discussion on Exploration and Exploitation

The performance of the algorithms varies a lot depending on the function (and its associated landscape, see Appendix C.1) and the average values of Table 3.2 and Table 3.3 fail to account for this. To better represent that aspect, equivalent tables are given in Appendix C.2 for every benchmark function.

In addition, regarding the batch effectiveness that is difficult to assess when considering only the time stamps, we also plot the mean best output (*i.e.*, the objective value) as a function of the number of simulations, for different batch sizes. For obvious reasons, only the necessary graphs will be displayed in due time.

Discussion on space-partitioning

Space-partitioning approaches allow a better intensification, and especially TuRBO and ℓBSP-EGO that allow a deep intensification into a small sub-region. This is indicated by their very good performance on the Ackley test function (displayed in Figure 3.6 for $q = 8$), which presents a single global minimum that can be found only with a strong intensification. On the contrary, the Alpine 02 function is highly multi-modal and a strong intensification could result in being trapped in local minima. It seems to be the case for TuRBO and ℓBSP-EGO when looking at Figure 3.7.

In this case, more exploration is needed. Actually, larger batches favor exploration as seem to indicate the results of TuRBO and ℓBSP-EGO on the 6d-Alpine02 function, reported in Table C.7 of Appendix C.2. Figure 3.8, displaying the results for the 6d-Alpine function with a batch size of 32, supports this hypothesis by showing that TuRBO and ℓBSP-EGO considerably improve their performance compared to MACE with a batch size of $q = 32$. However, MACE with $q = 8$ is still a better choice. Furthermore, their standard deviation is rather large indicating high uncertainty on the final result. It is also quite clear that the number of simulations ℓBSP-EGO is able to perform in a limited time is a strong advantage. It is often less efficient in terms of outcome for a given number of simulations, but considering time instead reverses the trend.

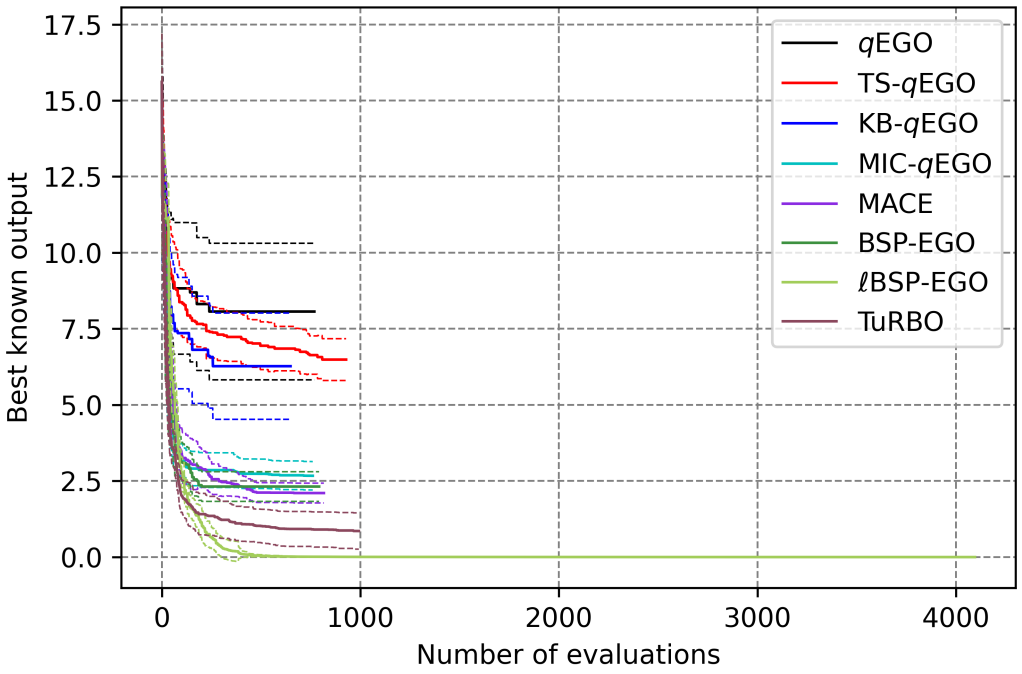


Figure 3.6: Evolution of the outcome for the **6D-Ackley** function with $q = 8$

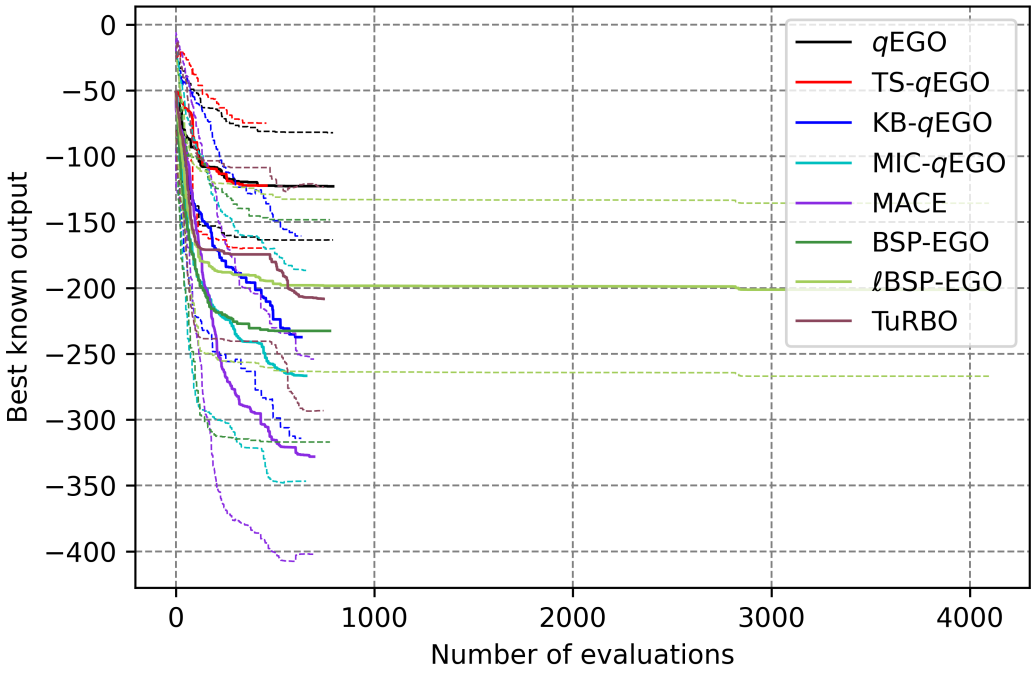


Figure 3.7: Evolution of the outcome for the **6D-Alpine** function with $q = 8$

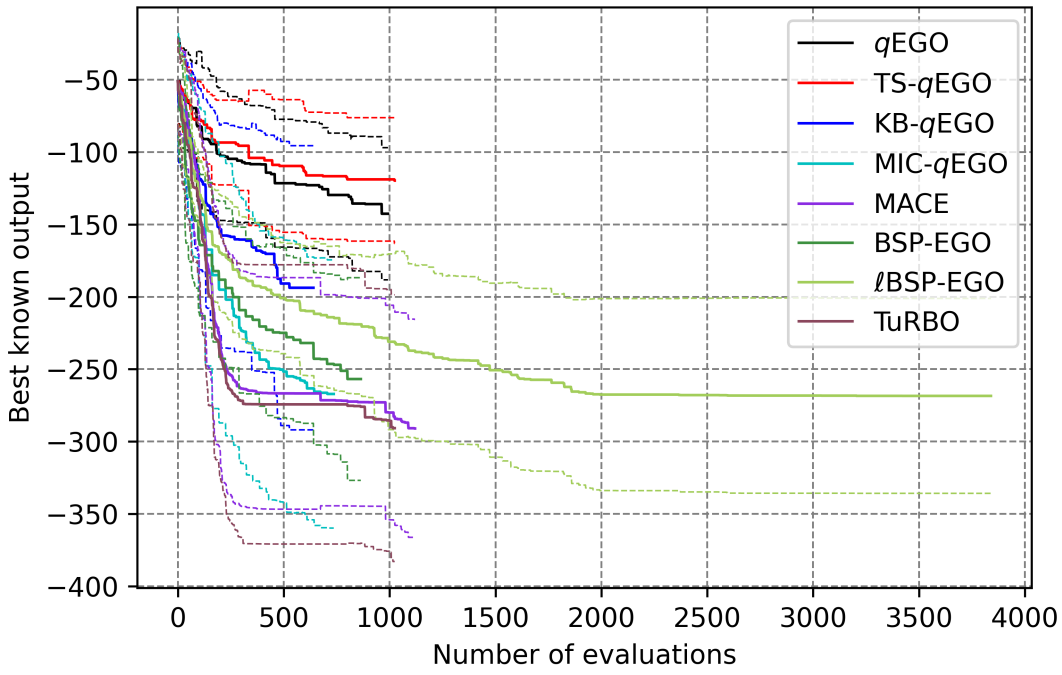


Figure 3.8: Evolution of the outcome for the **6D-Alpine** function with $q = 32$

Discussion on Multi-Infill strategies

The multi-infill approach clearly increases the batch effectiveness and is particularly efficient on the multi-modal Alpine 02 function as indicated in Figure 3.7. Both MIC-qEGO and MACE perform well, indicating a good aptitude for exploration. Actually, it results from the fact that those approaches rely only on AFs, whereas TuRBO and ℓBSP-EGO mostly intensify by restricting the search space based on the best known objective value to enhance the intensification.

The cooperative multi-infill approach seems to generally outperform the competitive one. In addition, MACE has better scalability than MIC-qEGO, which usually turns into better outcomes. However, the latter is denied by the results of the Rastrigin function where MACE performs poorly with $q = 8$. Indeed, as shown in Figure 3.9, MIC-qEGO clearly outperforms MACE. The Rastrigin function presents a single global minimum, but many local ones as it is extremely noisy. This could result in conflicting interests between the AFs, therefore, compromising between them leads to a less valuable batch of points. In that situation the competitive approach might be preferable. However, it is impossible to verify this hypothesis without further investigation.

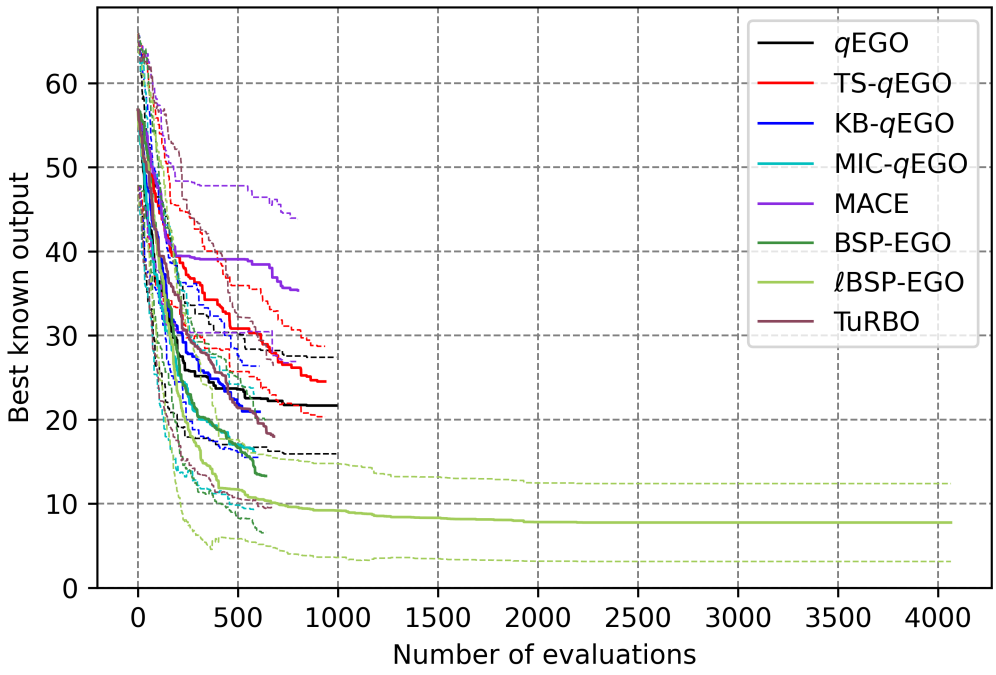


Figure 3.9: Evolution of the outcome for the **6D-Rastrigin** function with $q = 8$

3.3.5 Conclusions and Recommendations

The first objective of this study was to challenge the newly developed algorithms, namely BSP-EGO and ℓBSP-EGO. In addition, we assess the validity of multi-criteria and space-decomposition approaches on a large benchmark experimental setup. Experiments have been conducted on 5 functions in dimensions 6 and 12. The investigated methods are compared to the baselines of state-of-the-art algorithms that have proven their effectiveness. The results clearly indicate that all of BSP-EGO, ℓBSP-EGO, MIC-qEGO, MACE, and TuRBO possess features that are valuable to improve the optimization process as they all consistently outperform the baseline algorithms.

We first observed that the effectiveness of the batch is considerably improved by the multi-criteria selection, be it cooperative or competitive. Multi-infill strategies are effective for enhancing the exploration of the search space and escaping local optima. Cooperative strategies show a better batch effectiveness, and often better outcomes. However, the competitive approach can be more appropriate in some contexts (*e.g.* if cooperating criteria are conflicting).

Regarding the decomposition-based approaches, they allow more intensification and closely approach the global optimum if the promising region is spotted. They are also a good choice in very restricted budgets since they focus on refining the best-known solution and are not drawn towards exploration since the AP does not rely only on the AF. In addition, design space decomposition is clearly the best choice in higher dimensional spaces, even if multi-criteria approaches punctually perform best.

From a general point of view, it is preferable to keep the batch size under 16 to prevent the fast-increasing time cost of the global model. Only ℓBSP-EGO is exempt from this, acting on local models learned over a subset of data. Its better scalability enables the algorithm to outperform any other one of this benchmark analysis when the time budget and batch size increase. It has to be noted that MACE also shows better performances when increasing either the batch size, the time budget, or both. However, it is generally not sufficient to consistently outperform ℓBSP-EGO as the global model learning time is a major brake.

The ℓBSP-EGO algorithm presents the most interesting features as it scales well with the batch size, and continues to improve the objective value with time. It is the best choice for optimizing many of the benchmark functions of this analysis, in both dimensions 6 and 12.

More insights

TuRBO has been tested with and without the restart parameter that resets the length of the trust region. The results are clearly better with the restart option for the 6d-benchmark, so only this version is used here. However, we did not observe any difference in dimension 12. The hypothesis behind this is that TuRBO is designed to tackle higher dimensional spaces and the trust region shrinking is intensive. In dimension 6, TuRBO tends to converge towards very small trust regions quite fast, causing an early stagnation in the optimization process and numerical instabilities in the Gram matrix. Those instabilities do not occur for the considered budget in a higher dimension.

A similar observation can be made for the other algorithms: stagnation is often observed at some point. This could be improved by restarting the algorithm with a new set of design points, or at least a subset of the already observed ones. It has two advantages: limiting the early stagnation and speeding up the optimization process by limiting the size of the data set. Actually, a similar feature could be implemented for ℓBSP-EGO by resetting the tree when the volume of a leaf is too small. In addition, since the batch effectiveness remains limited, we could wonder if running several algorithms with lower batch sizes would be a better strategy.

All algorithms could be tuned to achieve better results. For instance, we could investigate the impact of the set of AFs on MIC-qEGO and MACE. Regarding TuRBO, a different reset option could be tried out, and an equivalent option could be added to ℓBSP-EGO. However, the previous recommendations still stand, and based on the conclusion, coupling space decomposition with multi-infill strategies appears as a suitable future research direction.

As we saw in Chapter 2, the real-world applications are quite different from the benchmark functions in several ways. In particular, we may face operational constraints such as a time-restricted budget. In the following, we tackle a real-world problem dealing with the resources management of an Underground Pumped Hydro-Energy Storage (UPHES) operator participating to the energy market.

3.4 Real-world Test Case: Optimal Scheduling of UPHES

Integrating renewable energy resources is a key challenge to ensure the transition towards a low-carbon energy system. Electricity storage systems provide a valuable solution to compensate for uncertain production, thus offering sustainable means to increase the flexibility of the system [140]. An appropriate option regarding storage technologies is offered by Underground Pumped Hydro-Energy Storage (UPHES). However, in modern competitive energy networks, individual actors rely on efficient operational strategies, enabling them to hedge the uncertainty of renewable energy resources. It is thus essential to dispose of efficient tools to make pertinent decisions at the different time steps of the energy markets (e.g., from long-term towards real-time) [26] (see Section 2.2).

3.4.1 Context and Motivation

From the operator's point of view, the quality of a decision is measured as a profit, so let us assume that for a decision $\boldsymbol{x} \in \mathbb{R}^d$, the expected profit of a UPHES operator is given by $f : \mathbb{R}^d \to \mathbb{R};\ \boldsymbol{x} \mapsto y = f(\boldsymbol{x})$. The simulator, f, is then considered as a time-consuming black-box function, which is further described in Section 3.4.2.

The complexity of physical phenomena in UPHES devices usually calls for model-based approximations [141]. Using these approximations, the evaluation of the expected profit is fast, and classical optimization methods (e.g., dynamic programming, genetic algorithms) can be applied to optimize the UPHES scheduling within a reasonable time budget [142, 143]. However, approximations may lead to unreliable simulations, motivating the recent proposal of a more robust model-based UPHES simulator in Toubeau *et al.* [141]. This comes at a much higher computational cost compared to conventional model-based approximations.

With this increased computational cost of the simulation, existing optimization approaches become impractical as operational constraints require scheduling optimization to be completed within approximately 30 minutes. This motivates us to investigate the use of surrogate models to (partially) replace the time-consuming simulator, in conjunction with parallel computing. To the best of our knowledge, SBO has never been applied to the UPHES scheduling problem. In our previous work [26] (see also Section 2.2), we demonstrate that PBO can be efficient in management problems in electrical engineering. Consequently, the overall optimization time can be considerably reduced. This is essential due to the time constraint arising from both the organization of energy markets and the complex modeling of physical and economic constraints of pumped-hydro systems.

3.4.2 Underground Pumped Hydro-Energy Storage

General description

Due to their ability to quickly and cost-effectively mitigate energy imbalances, Pumped Hydro-Energy Storage (PHES) stations offer an appropriate storage solution. PHES plants are composed of (at least) a lower and an upper reservoir from which water is exchanged to either produce or store energy. In off-peak periods, production might exceed consumption such that energy is saved by pumping water from the lower reservoir into the upper one. Then, it provides a substantial reserve of energy that can be later released when needed, *e.g.*, to maintain the transmission grid stability. Figure 3.10 shows the basic pumped hydro-energy storage unit with one lower and one upper reservoir.

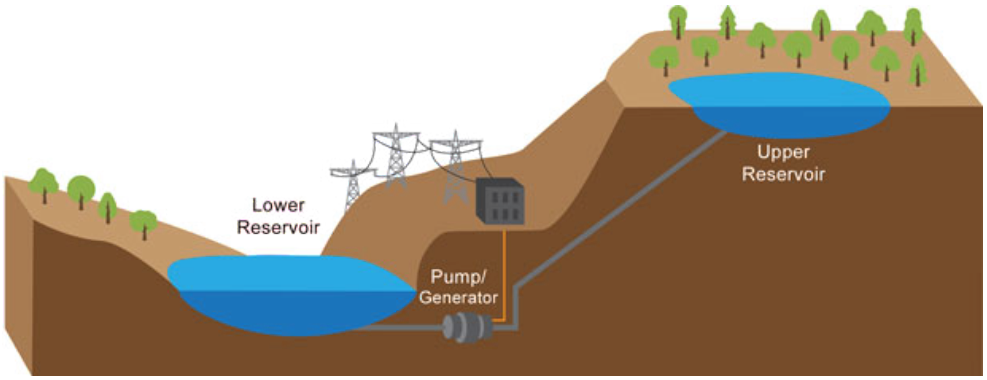


Figure 3.10: Illustration of the basic hydro-energy storage unit. Source: Dominion Energy, Powering Southwest Virginia

Recent progress in power electronics has enabled PHES units to operate with a reliable variable-speed feature in both pump and turbine modes. The flexibility offered by these facilities is highly valuable. Indeed, it improves the economic efficiency of existing resources such as wind farms or thermal power plants [140, 144], and provides ancillary services to ensure grid stability (such as frequency control or congestion management).

The inherent potential of PHES units leads to the development of new technological solutions such as Underground PHES (UPHES) for which the lower basin is located underground. A significant advantage of UPHES is the limitation of expenses from civil engineering works thanks to the recycling of end-of-life mines or quarries. These stations have a very limited impact on the landscape, vegetation, and wildlife, and are not limited by topography so that more sites can be exploited [145]. In the current competitive framework governing the electricity sector, UPHES units are exploited with the objective of return on investment. Consequently, the profit of such stations must be maximized. This task is challenging since the UPHES operation is governed by two main nonlinear effects that cannot be easily modeled with traditional analytical models [141].

Firstly, groundwater exchanges between the reservoirs and their hydro-geological porous surroundings may occur. This situation typically arises for UPHES when the waterproofing work is not feasible or uneconomical [146]. Secondly, UPHES units are generally subject to

important variations of the net hydraulic head (i.e., the height difference between water levels in the reservoirs). These variations are referred to as the head effects [147], and are typically quantified through laboratory measurements on a scaled model of the hydraulic machines [148]. This characterization of head effects is important since the head value defines both the safe UPHES operating range as well as the efficiency of both pump and turbine processes. In this way, the safe operating limits in pump and turbine modes continuously vary over time with regard to head variations. In general, the performance curves of UPHES stations are difficult to model since they present a non-convex and non-concave behavior.

Directly integrating these effects into model-based optimization (which maximizes the UPHES profit in the different market floors) implies a high computational burden or strong assumptions that may jeopardize the feasibility of the obtained solution. To address these issues, a simulation-based BO strategy has been developed in this work. The simulator, which is a black box from the user's perspective (developed independently without access to the source code), returns the daily UPHES profit accounting for all techno-economic constraints. The full description of physical and economical constraints can be found in Toubeau *et al.* [102].

Problem Instance

The experimental evaluation uses a real-world site located in Maizeret (Belgium) as a test case. Its configuration is shown in Figure 3.11. The lower basin is a former underground open pit mine subject to groundwater exchanges. Furthermore, the surface of both reservoirs is relatively limited, which results in significant head effects. The specific features of the UPHES unit are taken into account in the simulator implemented in the Resource-Action-Operation language [149] and Matlab. The UPHES nominal output ranges (for the nominal value of the hydraulic head) are respectively [6, 8] MW and [4, 8] MW in pump and turbine modes and the energy capacity is 80 MWh. The optimization problem involves 12 decision variables.

This simulator is denoted as f in the following and, according to a decision vector $\boldsymbol{x} \in \mathbb{R}^{12}$, it returns the expected profit $y = f(\boldsymbol{x}) \in \mathbb{R}$. The 12-dimensional decision vector includes 8 decision variables to participate in the different time slots of the energy market, and 4 to the reserve market (*i.e.* provision of ancillary services). The design variables from x_1 to x_8 range from -8 to 8 and represent the amount of energy that is delivered or stored (depending on the sign). The remaining ones, from x_9 to x_{12}, range from 0 to 8 as we only provide energy in the reserve market. The number of decision variables is subject to modification in order to gain more flexibility in future studies. The objective is then to find the decisions that maximize the daily expected profit. The objective function f also involves the constraints and deals with them by adding a penalty term inside the simulator.

These UPHES decisions must comply with hydraulic and electro-mechanical constraints over the whole daily horizon. This results in a challenging optimization problem (embedded in the simulator), which is discontinuous (from the cavitation effects of the pump-turbine machine that incur unsafe operating zones), nonlinear (from the complex performance curves of the unit), mixed-integer (to differentiate the pump-turbine-idle operation modes), which is subject to uncertainties (*e.g.*, on water inflows and market conditions). The formulation used in this work can be found in [102].

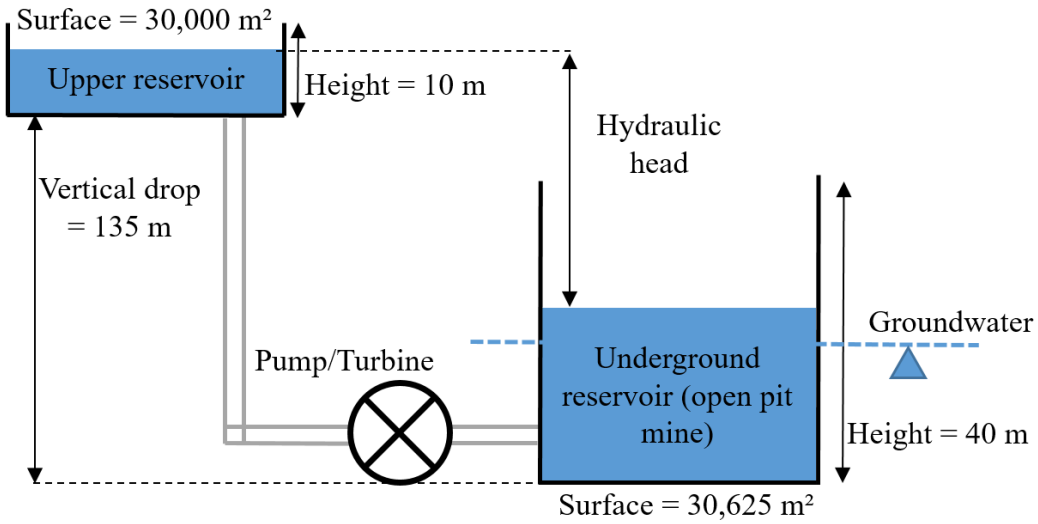


Figure 3.11: Topology of the UPHES unit on Maizeret site [102].

Quick literature review

With full model-based optimization, errors caused by the inherent modeling approximations (typically, linearizations), required to ensure computational tractability of complex effects (*i.e.* nonlinear water levels within reservoirs, penstock head loss, and head-dependent pump/turbine performance curves) may lead to infeasible solutions [141]. Relying instead on surrogate models may provide an efficient and robust solution to this problem.

According to the two surveys of Taktak and D'Ambrosio [142], and Steeger, Barroso, and Rebennack [143], this kind of management problem in electrical engineering is typically solved using Mixed-Integer (Linear) Programming [150], dynamic programming [151], or nonlinear programming [152]. Current techniques also involve metaheuristics such as GA [153] or PSO [154]. However, we are not aware of any BOA or SBO approaches for the UPHES management problem.

3.4.3 Experimental Setup

We investigate several parallel batch-based algorithms using different APs that have proven their efficiency in the previous section and we compare the optimized average profit. Experiments are conducted with different batch sizes to evaluate and compare the scalability of the approaches in regard to the simulation time. The same protocol as already presented in Subsection 3.3.2 is adopted for this experiment to account for the time constraint of the application. Indeed, the optimization must be conducted in a restricted time and, contrary to common assumption, the acquisition time is not negligible compared to the simulation time. Consequently, the acquisition time needs to be taken into account in order to perform an efficient optimization. However, based on the previous results, the number of experiments is reduced to the best performing approaches of Section 3.3.

Objectives

Compared to the precedent benchmark analysis where we focus on the scalability and the batch effectiveness, for the UPHES problem we are mainly interested in the performance of a method in terms of the final outcome. Then, the main objective for the UPHES application is to identify the best strategy to optimize the expected profit regarding the given optimization context. Unlike the benchmark functions, the UPHES simulator has a non-negligible cost. Hence, the scalability of the method may vary. Given a 10-second simulation and a 20-minute budget, the maximum number of cycles is 120. Assuming there is no cost for obtaining a batch of candidates, the total number of simulations should be $120 \times n_{cores}$.

Experimental Protocol

The most promising algorithms from the 12d-benchmark analysis of Section 3.3 are TuRBO, ℓBSP-EGO, MACE, and MIC-qEGO. Consequently, only these four algorithms are investigated in the following, in their exact same settings. The UPHES simulator itself is implemented in Matlab and the domain-specific RAO language. As the black-box UPHES simulator executable requires a software license, experiments are performed on a single node of a university cluster, preventing us from conducting the analysis on large batch sizes. The dedicated node includes 2 Intel(R) Xeon(R) CPU E5-2630 v3 2.40 GHz, with 8 cores each, limiting the batch size to $q = 16$.

Due to the responsive energy market operational constraints, the optimization must be completed within tens of minutes. Hence, to remain consistent with the time constraints, the global budget of each optimization run is chosen as a time budget of 20 minutes, without initial sampling. According to usual recommendations in BO, a fraction of the budget is allocated to the initial sampling. The initial sampling budget is set to 128 in accordance with the previous section's protocol. Experiments are performed for $n_{batch} = 4, 8, 16$, with the same initial sample for any algorithm and batch size. Each run is performed 10 times to assess for the robustness of the results.

In order to avoid too much case-specific conclusions, the choice is made to consider a fixed time of 10 seconds for a simulation. Indeed, one execution of a simulation lasts for 9 to 10 seconds, and a non-negligible overhead results from parallel calls to the black-box simulator. This overhead is only case-specific since the simulator resorts to RAO language, under the form of an executable program which necessitates interfaces between programs not suited for parallel computing. Therefore, the parallel overhead is independent of the parallel framework and is only caused by the software-dependent simulator.

3.4.4 Results and Discussion

Presentation of the results

Results displayed in Figure 3.12, Figure 3.13, and Figure 3.14 present the average profit value of the UPHES management problem according to the number of simulations performed by the

algorithms. Dashed lines are added to indicate the standard deviation around the average objective value. Optimization runs are repeated 10 times with 10 different initial sets, and each algorithm is run once with each initial set. Therefore, within each figure, each curve has the same starting point.

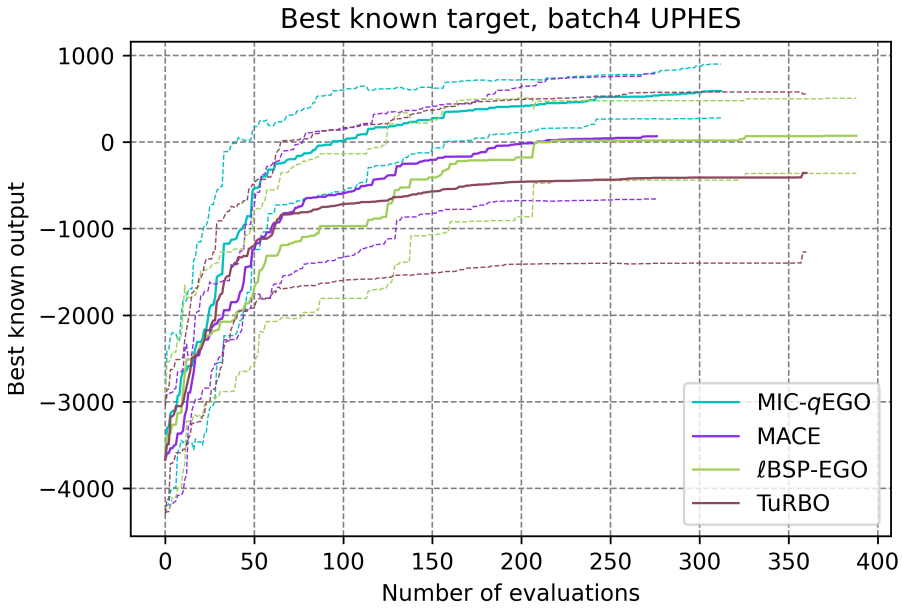


Figure 3.12: Evolution of the best known profit according to the number of simulations. Plain lines display the average profit while dashed-lines represent their standard deviation over the 10 repetitions.

Since the limiting factor is time, each execution of the optimization algorithms does not perform the same number of cycles (and simulations). Consequently, the curves only display the results for which all data are available, and the rest is truncated. Thereby, the final expected profit is not always visible on the graphs. Nevertheless, the final results at the end of the time budget are shown in Table 3.4. The table also presents the minimum, maximum, and standard deviation of each set of optimizations, for all batch sizes and all approaches.

Analysis of the results

MIC-qEGO seems to be the best choice among the considered algorithms, for all budgets and batch sizes. Indeed, Figure 3.12, Figure 3.13, and Figure 3.14 show a clear preference for MIC-qEGO. The performances of TuRBO, MACE, and ℓBSP-EGO are quite similar since neither of them is constantly better than the two others. However, MACE and TuRBO present a larger standard deviation (indicated by the dashed lines on the graphs) which denotes very different results obtained with the same algorithm. Table 3.4 shows the minimum and maximum values obtained by MACE and TuRBO, where we can see a large amplitude between the two extremes. ℓBSP-EGO seems more robust, except for $q = 16$. However, for the largest batch size of this study, none of the methods achieves better performances compared to lower batch

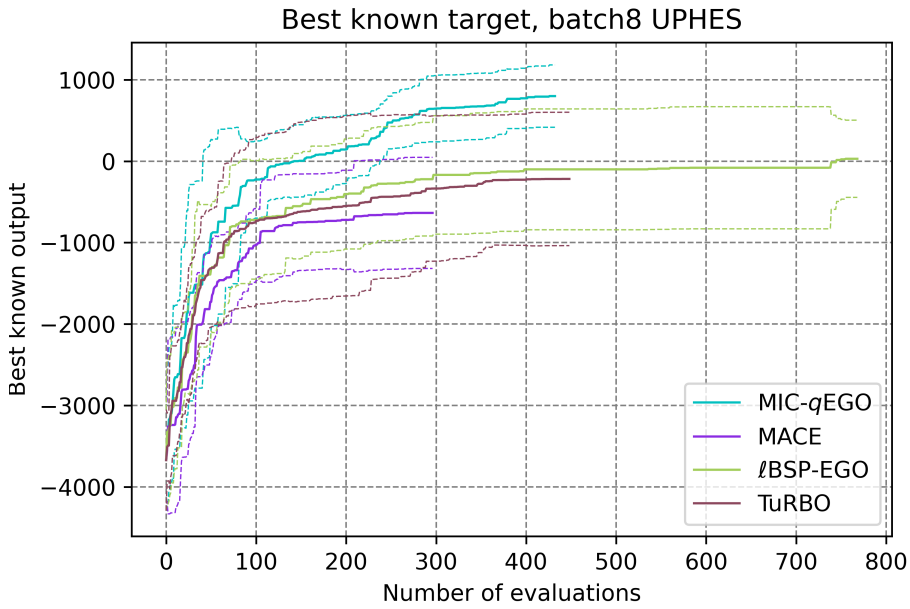


Figure 3.13: Evolution of the best known profit according to the number of simulations. Plain lines display the average profit while dashed-lines represent their standard deviation over the 10 repetitions.

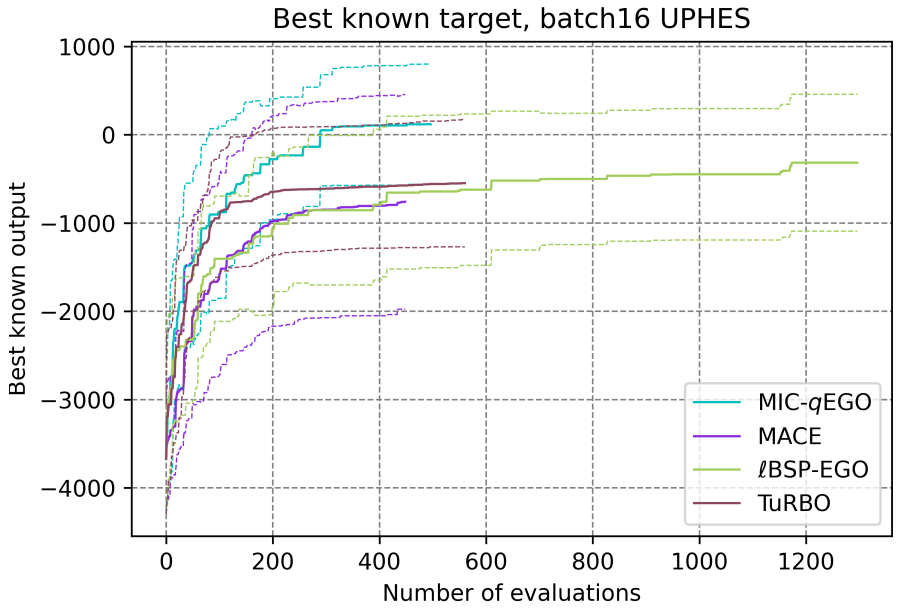


Figure 3.14: Evolution of the best known profit according to the number of simulations. Plain lines display the average profit while dashed-lines represent their standard deviation over the 10 repetitions.

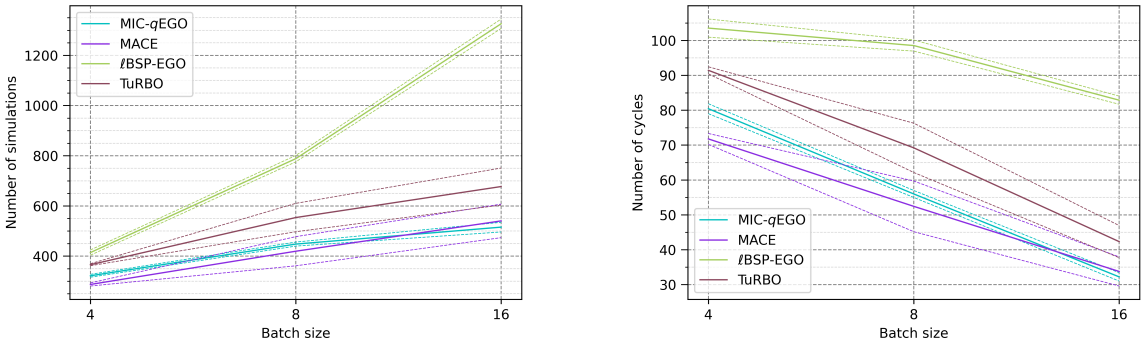
Table 3.4: Minimum, maximum, average profit values (EUR) as well as standard deviation of the UPHES management problem obtained with 10 runs of each method according to batch size.

$n_{batch} = 4$	min	mean	max	sd
MIC-qEGO	**196.74**	**592.24**	1153.59	**331.98**
MACE	-1528.30	68.72	1052.43	767.31
ℓBSP-EGO	-442.72	72.28	940.33	459.83
TuRBO	-1569.05	-327.57	**1310.19**	932.40
$n_{batch} = 8$				
MIC-qEGO	**113.33**	**802.80**	1233.92	409.59
MACE	-1579.85	-420.42	1139.53	764.48
ℓBSP-EGO	-663.41	82.68	489.91	**396.15**
TuRBO	-1225.09	-214.51	**1358.43**	872.61
$n_{batch} = 16$				
MIC-qEGO	**-1252.78**	**119.35**	**876.98**	**724.00**
MACE	-2871.26	-708.70	573.27	1321.22
ℓBSP-EGO	-2179.73	-310.76	475.36	825.06
TuRBO	-1735.59	-542.88	336.86	765.86

sizes. Increasing the batch size from $q = 4$ to $q = 8$ improves the average expected profit for all methods, except for MACE.

The number of simulations is higher for any algorithm when the batch size increases, as indicated in Figure 3.15a. Nevertheless, according to Figure 3.15b, it always results in a loss in terms of cycles. As the batch size increases, the data set size increases even faster, which results in time-consuming surrogate modeling. The only exception is ℓBSP-EGO which operates on local models and thus the learning time is not impacted. Nevertheless, the size of the data still causes an overhead in the algorithm because the local models are fit using a distance-based subset of points. The gain in terms of simulations is tempered by the loss in terms of cycles. This implies that for an equivalent performance, the batch selection must be effective. However, the results displayed in Table 3.4 indicate that going beyond $q = 8$ deteriorates the results for all approaches.

The main difference between the UPHES problem and the benchmarks, despite their evident different nature, is the presence of the time cost of the simulation. Obviously, this impacts the number of simulations the algorithm can perform in 20 minutes and we can see that *slow* APs are better in this context. Our publication [22] related to this section also used KB-qEGO and BSP-EGO and reported the same conclusions: MIC-qEGO outperforms other methods.



(a) (b)

Figure 3.15: Solid lines indicate the average over 10 runs, and dashed lines of same colors indicate their standard deviation. (**a**) Number of simulations as a function of the batch size. (**b**) Number of cycles as a function of the batch size.

Discussion

For the UPHES management problem, the final expected profit is considerably improved compared to the initial sampling thanks to PBO. Even considering a large random sample of almost 12,000 objective function evaluations, the best-observed profit is around -1200 €. All investigated BOAs achieve much better profits with significantly fewer simulations. This demonstrates the need for efficient optimization algorithms for this application.

Using PBO in a time-restricted setting necessitates to take into account the increasing time needed to fit a surrogate model and to optimize the AF. Our experiments have indeed shown that there is a "breaking point" beyond which an increase in the batch size deteriorates performance instead of adding value to the optimization process. The latter is also observed in the benchmark analysis of Section 3.3. However, the time cost of the UPHES simulators induces different behaviors. Concerning the UPHES management optimization, the "breaking point" appears around $q = 8$ as the best performances for all investigated algorithms are obtained for this batch size, and degrade afterward.

Conclusions and Future Research Directions

Four batch-acquisition BOAs are investigated in this study with the objective of identifying the most suited approach for maximizing the profit of a UPHES operator. The profit is given by a simulator computing the expected profit according to a set of decisions. The evaluation of a decision (*i.e., the simulation*) lasts 10 seconds, and the optimization must be completed in 20 minutes (initial sampling excluded). These constraints place us in the context of black-box optimization with a time-consuming simulator, except that the simulator is time-consuming in regards to the time constraint. Therefore, unlike the classical BO assumption, learning and acquisition times are not negligible.

The study reveals that in this context, on benchmark problems as well as on the UPHES management problem, resorting to large batch sizes and large parallelization quickly becomes excessively time-consuming, worsening the performance of any BOA considered in this work. The best trade-off lies between batch sizes of 4 and 8 and the best performances are achieved by MIC-qEGO, our proposed variant of qEGO, which manages to find consistently good solutions to the UPHES management problem. Accordingly, it appears as a viable way to approach time-constrained applications despite the high overhead incurred by surrogate management. However, making the most of parallel computing in this context remains challenging as the best batch size is rather low.

Finally, regarding the robustness of the algorithms, the present study considers only one scenario of UPHES management. For a better robustness of the conclusion, it would be interesting to evaluate the performances of the algorithms on different scenarios.

3.5 Chapter's Conclusions

PBO faces difficulties in scaling to larger batch sizes, and then to profit from the increased capacity of recent processors. On the one hand, the scalability in terms of number of simulations in a dedicated time is hampered by the complexity of the APs. On the other hand, the batch effectiveness is low: the batch acquisition is much less effective than the sequential one. Those two factors contribute to the ineffectiveness of large-scale batched BO. In this chapter, we developed a new strategy that improves the scalability and batch effectiveness of PBO algorithms. The approach leverages spatial decomposition of the design space and executes distinct sub-APs in non-overlapping sub-regions. The different tasks induced by the space decomposition allow the use of parallel computing inside the AP and offer another way to deal with the exploration/exploitation trade-off. The distinct sub-regions ensure the diversity of the candidates, while intensification is promoted as the optimization progresses by splitting the most promising sub-regions. Two algorithms are derived from it, namely BSP-EGO and ℓBSP-EGO. They respectively use global and local surrogate modeling.

We first investigated benchmark functions in a 6-dimension space, where we observed considerable improvement of the multi-infill or space partitioning (including trust regions) approaches compared to multi-point or approximation-based alternatives. More precisely, methods dealing with the design space (using decomposition or trust regions) seem to outperform the others for most budgets. However, they favor intensification at the risk of being trapped in local optima. Multi-infill approaches are less subject to being trapped, but also sometimes lack intensification. This observation becomes even clearer when dealing with 12d-benchmark functions, where intensification is usually preferred, due to the size of the design space. Thus, locally-acting methods such as TuRBO and ℓBSP-EGO have consistently better performances. Nevertheless, it is worth noting that the cooperative multi-infill algorithm, namely MACE, scales well with the batch size and can compete with them for some problems, and MIC-qEGO also punctually performs well. For instance, it is found to be the best algorithm for the UPHES management problem, where the simulation time is not negligible, unlike for the benchmark functions.

More precisely regarding tree-based algorithms, the parallelization of the AP and the intensification induced by the tree evolution manage to improve the optimization process compared to the reference algorithms. However, as for most algorithms of this study, the number of simulations is strongly impacted by the remaining sequential part of the algorithms, which principally means the model fitting. The local modeling approach (*i.e.* ℓBSP-EGO) facilitates scaling the parallelization to higher batch sizes since local searches only depend on a sub-model which is much faster to train. The gain in terms of number of simulations is evident and grants better outcomes in many situations. Consequently, the BSP paradigm offers a scalable framework for PBO, which can easily be coupled with other AP strategies such as cooperative and competitive sampling.

Indeed, the hybridization of the investigated strategies might be a suitable research direction since the scalability and batch effectiveness remain limited. The observed results indicate that for most algorithms, exceeding 8 candidates per batch does not yield better outcomes, except for MACE and ℓBSP-EGO. Regarding exclusively the final outcome, a batch size of 8 or 16 is adequate for all algorithms. Consequently, it might be preferable to run many fast complementary BO algorithms instead of a single one consuming the whole budget.

Considering only moderately time-consuming simulators, a fast operating AP allows the algorithm to perform a lot more evaluations in the given time budget. As illustrated by the good performances of ℓBSP-EGO on the benchmark functions, the fast AP is a strong advantage. Nevertheless, this study does not take into account other SBO algorithms that could be suited in this context, such as BNN-GA in Section 2.3. Furthermore, this analysis does not account for the simulation time as the benchmark functions evaluation have a very low computational cost. Actually, the best-suited strategy for a problem mostly depends on the computational cost of the simulation. This should be connected to the computational budget and the computing resources to be able to select the best algorithm.

Chapter 4

Bayesian *versus*/with Evolutionary Optimization

In the previous chapters, the objective functions are qualified as time-consuming to evaluate. Informally, this means that the number of sequential evaluations in the dedicated time does not exceed a few dozen. If the function evaluation requires human intervention or spending money, one would certainly choose to minimize the number of calls to the objective function and spend more time in the Acquisition Process (AP)[1]. However, this type of optimization problem is quite well addressed with sequential methods like Efficient Global Optimization (EGO)

[1] construct the best surrogate model and perform extensive search to optimize the Acquisition Function (AF)

[1] and assimilated ones [9, 10, 56, 58, 59]. Our approach is more focused on objective functions for which we have no limited number of queries, but rather a fixed time budget. The objective is then to find an AP that can find an effective batch of candidates of arbitrary size within a reasonable time. The study of Section 3.3 revealed very different per-cycle times among the investigated algorithms. But still, for most of them, the time per cycle grows fast, up to becoming detrimental. Only local-model methods manage to keep a reasonable per-cycle time cost. We have already come up to this breaking point in Chapter 2, that recent methods such as multi-infill or partitioning strategies have only managed to push back. However, the previous study does not take into account the time cost of a function evaluation. In addition, we observed in Section 2.3, where BO was compared to SAO, that the relevance of certain classes of methods strongly depends on the budget in terms of a number of simulations, or a total wall time. Consequently, in the following chapter, we investigate further the concept of *time-consuming* by attributing a fictitious cost to the simulations and compare the BO approaches to recent SAEAs.

In this chapter, we intend to better characterize what is a time-consuming simulator and to relate the context of the optimization with the appropriate choice of algorithm. We define the context as the combination of the budget in terms of wall time, the expensiveness of the objective function, and the computational resources available for the optimization. With the latter definition, we further investigate the notion of time-efficiency introduced in Section 3.3 that relates the time complexity of an algorithm with its performance for given time-stamps. We conduct an extensive investigation involving the already studied BOAs which we challenge with SAEAs and EAs to identify their domain of performance regarding the different contexts. The proposed work reveals a threshold below which BOAs are to be favored. The threshold is characterized by the theoretical upper bound of the number of simulations in a given time. The definition of the theoretical upper bound allows us to assess the efficiency of each AP, with respect to the expensiveness of the simulator. For a budget higher than this threshold range, the BOAs are generally hampered by their execution time compared to SAEAs. The time cost associated with the acquisition of candidates through a GP becomes too important and SAEAs are preferred in this situation. (Surrogate-free) EAs and SAEAs show much better scalability regarding the number of simulations performed performed within a given time frame, and SAEAs notably reduce the number of simulations compared to EAs for an equivalent outcome quality. The observed threshold helps practitioners to adequately choose or design time-efficient algorithms. A hybrid algorithm is built upon the threshold range by simply switching from the best-performing BOA of this study to the best SAEA. The hybrid algorithm shows to perform well in a much wider range of contexts.

Using this hybrid algorithm, we tackle another electrical engineering problem related to energy storage. The objective remains the optimal management of a pumped-hydro station, except that the lower basin is not underground, contrary to Section 3.4.2. We will refer to this problem as the PHES management problem. This optimization problem addresses the multi-period day-ahead scheduling faced by PHES plant owners who participate in both the energy and reserve markets (see Section 2.2). The main difference with the UPHES analogous is that 30 design variables are considered, and the simulation is much faster. However, the time cost is still too high to efficiently solve the problem with classical metaheuristics such as EAs or PSO. Consequently, approximations inside the simulator are generally conceded to speed up the simulations and resort to exact methods. For instance, piecewise linear approximations

can be used for a MILP formulation [155]. We first challenge this approach by applying SBO algorithms to the complex (and thus more reliable) simulator. In a second time, we propose a multi-fidelity approach combining the guarantees offered by the MILP solver with the efficiency of SBO algorithms. The simulator is scenario-based, which means that different scenarios of the same problem can be used to assess the robustness of a method. For this study, 11 scenarios are considered.

Higher dimensional spaces usually require a larger sampling and are more complex to tackle with BO. In those spaces, the distance between the design points is rather large making the correlation matrix inefficient in capturing the relation between samples [156, 157]. In addition, AFs that are efficient in lower dimension spaces might not be in higher ones. Actually, it has been observed that the predicted value of the model is a better AF than EI in certain cases [124]. This aspect is partially approached with the PHES management problem which involves 30 design variables, but a lot remains to investigate in this area.

We first describe the investigated algorithms in Section 4.1 before analyzing them in Section 4.2. The study results in the creation of a hybrid algorithm that is further described and validated in Section 4.3. Finally, we challenge the best-performing algorithms through the PHES management problem that involves 30 dimensions. Section 4.4 constitutes an opening to higher dimensional problems which constitutes another big challenge in PBO.

The contributions of this chapter have been published in the following articles:

- *In academic journals*:
  - Gobert, M., Briffoteaux, G., Gmys, J., Melab, N. & Tuyttens, D. Observations in Applying Bayesian versus Evolutionary approaches and their Hybrids in Parallel Time-constrained Optimization, ***Currently under review*** *in Engineering Applications of Artificial Intelligence*
  - Favaro, P., Gobert, M. & Toubeau, J.-F. Multi-fidelity Optimization for Pumped Hydro Energy Storage Participating in Energy and Reserve Markets, ***Currently under review*** *in Applied Energy*

4.1 Towards Time-Efficient Algorithms

The term *time-consuming* has been used many times along this manuscript without being really characterized. In this section, we try to define more precisely what time-consuming means, and what algorithms are the most suited regarding the simulation time and the available computational resources (*i.e.* the number of used processing units).

4.1.1 Context and Motivations

The choice of the category of methods according to the expensiveness of the problem at hand is an important question whose answer will guide practitioners to best solve real-world engineering problems and to design more successful SBO approaches. Classifying a problem as computationally cheap, moderate, or expensive depends on the available time budget and the available computational resources. In fact, a ten-second evaluation of the objective function characterizes a moderately expensive problem when the optimization problem must be solved in a short time budget, such as in the UPHES problem of Section 3.4.2. This situation is particularly faced when dealing with decision-making problems such as in the real-time energy market [22, 26]. This aspect of time-constrained optimization is not often considered although essential in many areas [158].

In the time-constrained context, it is also important to control the time cost of other operations involved in the optimization process, namely the surrogate training and the acquisition of new candidate solutions. This study investigates the existence of a threshold from which BOAs are more efficient than SAEAs. The threshold is identified according to the available computational power and the time cost of the objective function by performing intensive numerical experiments on the CEC2015 test suite designed for expensive black-box optimization problems. The study includes both surrogate-free and surrogate-based approaches to extract more general guidelines.

Even though GPs are said to be non-parametric, the efficient implementation of the BOAs rests on several hyper-parameters. This makes the comparison difficult since the implementations of the model (hyper-parameters and fitting budget), the AF, the inner-optimizer and its budget, the benchmark functions (different versions, domains, dimensions, etc.) are all choices that impact the performances of the method. A recent paper from Le Riche and Picheny [159] reports that BO is not often compared to widely different alternatives. In addition, the benchmarks are restricted, which makes the comparison difficult. Similarly to our previous remark, they raise the point that implementation may vary across the different studies, possibly resulting in different conclusions, making it difficult to build recommendations.

As a partial response to the raised issues, we propose a non-usual benchmarking procedure that takes into account the simulation time and the batch size, and fixes the budget as a wall clock time. We compare the most efficient algorithms of the previous chapter, and supplement the test-bed by the conceptually different SAEAs. For more detailed explanations on BOAs and SAEAs, see Sub-section 1.1.2. The study is conducted on a large set of functions, in different contexts defined by the computational cost of an evaluation and the available computational power. This type of method is already used in Section 2.3 to tackle the tuberculosis transmission

control problem. According to the attributed computational budget, three classes of algorithms are available and investigated in this study: BOA, SAEA, and (Surrogate-free) EAs. A large set of benchmark functions and engineering applications are considered with various computational budgets to come up with guidelines for the choice between the three categories. According to the computational expensiveness of the objective functions and the number of processing cores, we identify a threshold from which SAEAs should be preferred to BOAs. Based on this threshold, we derive a new hybrid Bayesian/Evolutionary algorithm that allows one to tackle a wide range of problems without prior knowledge of their characteristics.

4.1.2 Surrogate-Assisted Evolutionary Optimization

Both SAEAs and BOAs rely on surrogate models to assist the optimization. However, the core element is slightly different. Following the descriptions of Section 1.1.2, SAEAs are SAO algorithms relying on evolutionary optimization and using surrogate models to either filter out or evaluate candidates. In contrast, BOAs use the model to construct an oracle that evaluates the promisingness of a candidate. Maximizing the latter samples the points to evaluate next. However, the surrogate modeling and the selection of candidates can become excessively time-consuming or less efficient. This partially explains why SAEAs are preferred to BOAs for optimizing less expensive objective functions [20].

More precisely, SAEAs differ from BOAs by the fact that the acquisition of new candidates does not rely on the optimization of the AF but rather on evolutionary operators such as crossover and mutation in a Genetic Algorithm (GA)[160] or the velocity and position updates in a Particle Swarm Optimization (PSO) [161]. The surrogate model intervenes in the evolution to screen out and/or evaluate new solutions through the Evolution Control (EC). The EC is similar to the AF as it points out the most promising new candidates [162–164].

Evolutionary Algorithms

The standard EA is described in Algorithm 3, on page 48. The population size p_{size} is a critical parameter impacting the trade-off between exploration and exploitation. Large populations tend to favor exploration and smaller populations promote exploitation [165]. Even if EAs are applied successfully to a large variety of real-world problems [104–106], they require a large number of objective function evaluations to converge. To bypass this weakness preventing the applicability to simulation-based real-world problems, parallel computing, and surrogate modeling are leveraged *via* the definition of an EC [19].

Evolution Control

Inserted between the production of offspring and their evaluation, the EC aims at distinguishing the most promising ones [166, 167]. The criterion defining the promisingness is a comparison operator based on the surrogate prediction and/or predictive uncertainty [19]. On the one hand, the surrogate can be used as a filter to discard unpromising candidates while the remaining offspring are evaluated by the objective function f. On the other hand, the surrogate can also be

used as an evaluator in place of f. In doing so, the population may embed predicted individuals. Let's denote as n_{off} the number of offspring issued per generation. The population of n_{off} offspring is split into three sub-populations of n_{sim}, n_{pred}, and n_{disc} individuals respectively. The first sub-population containing the most promising candidates is evaluated with f, the second sub-population composed of moderately promising candidates is evaluated by the surrogate model, and the less promising candidates are discarded. An EC is defined by the promisingness comparison operator and the values for n_{sim}, n_{pred} and n_{disc} under the following condition: $n_{sim} + n_{pred} + n_{disc} = n_{off}$. The promisingness comparison operator can be defined using only the surrogate model's prediction or using a trade-off between the prediction and the predictive uncertainty. In the first case, a lower prediction indicates a better promise, while in the second case, the offspring with high predictive uncertainty are also considered promising. The second operator category acts similarly to the AF in BOAs.

4.1.3 The Investigated Algorithms

The competing algorithms are categorized into three groups following the previous descriptions: BOAs, SAEAs, and EAs. For each newly introduced algorithm, a succinct description is given along with its associated reference publication.

Bayesian Optimization Algorithms

The BOAs investigated in this comparative study are TuRBO [18], MIC-qEGO [22], BSP-EGO [27], ℓBSP-EGO [123], MACE [75], and ϵ-Shotgun [77]. Except for ϵ-Shotgun, all the BOAs are described in Chapter 3.

The selected BOAs all possess distinct features that have been introduced to tackle the limitations of standard sequential BOAs. They are chosen among recent methods outperforming classical parallel BOAs (such as q-EI-based methods) that present interesting features for our study such as the local APs, parallel acquisition of the candidates, or multi-criteria APs. Consequently, we expect to observe different behaviors of BOAs depending on the contexts considered in the experimental section (see Section 4.2).

First, it is often difficult to identify the best AF, both MACE and MIC-qEGO propose to use simultaneously several AFs. Second, the acquisition strategy might be time-consuming, therefore, MIC-qEGO, BSP-EGO and ℓBSP-EGO offer the possibility to make the acquisition parallel. The GP variance is known to be sometimes unreliable which often results in over-exploring the search space [124], especially when the dimension is relatively high. TuRBO and BSP-based methods make use of spatial decomposition to increase the intensification as the optimization advances. Finally, ϵ-Shotgun is the only new BOA of this section. It uses another strategy to avoid the expensive optimization, it finds the best candidates by optimizing one AF and samples around it.

ϵ-Shotgun:
The ϵ-Shotgun algorithm [77] uses an ϵ-greedy strategy: with probability $1 - \epsilon$ the AF is used to identify the first candidate of the batch $\boldsymbol{x}_{b1}$; else, $\boldsymbol{x}_{b1}$ is randomly sampled. The novelty

of ϵ-shotgun lies in the selection of the remaining candidates to complete the batch. They are sampled according to a normal distribution centered at $\boldsymbol{x}_{b1}$ with the standard deviation computed as follows: $r = \frac{|pred(\boldsymbol{x}_{b1}) - \min \boldsymbol{y}|}{L} - \frac{\sigma(\boldsymbol{x}_{b1})}{L}$, with L the local estimated Lipschitz constant. L is estimated within a hypercube centered in $\boldsymbol{x}_{b1}$ with side lengths of twice the length scale of the GP kernel. The algorithm is investigated on both real-world and synthetic problems from 1 to 10 dimension(s) and batch sizes from 2 to 20.

Surrogate-free Evolutionary Algorithms

The selected EAs for this study are the GA and the PSO algorithms.

Genetic Algorithms:
We use a GA where the selection of parents is based on the fitness of the individuals. In order to produce the offspring solutions, the crossover and mutation reproduction operators are employed. The crossover combines features of the parents while the mutation introduces small random perturbations. The replacement step is elitist, this means only the best-fitted individuals are kept for the next generation. The GA uses a tournament selection, a simulated binary crossover and a polynomial mutation as advised in [32]. Two variants of this GA are considered by setting the size of the population to either 32 or 64, thus providing different trade-offs between exploration and exploitation. In the following, these variants are denoted **GA32** and **GA64** respectively.

Particle Swarm Optimization:
In the PSO algorithm, the population is a swarm of particles evolving in the search space Ω. The evolution of the swarm follows the three standard steps of the EAs. The selection step consists in determining the best neighbor of each particle. The production of new candidates relies on the current position of the particle, on the best neighbor, on the best solutions of the swarm, and on the personal best position of the particle. The generation of offspring also implies to update velocities and positions through formulas that can be compared with reproduction operators in GAs. The replacement is mandatory in a PSO algorithm as each new position of the particle replaces the previous one in the swarm. However, a memory is maintained to keep each best position of the particle occupied so far. The considered PSO uses the constriction method to update the velocities [168, 169] and the absorbent walls strategy to ensure that the new particles always stay within the box-constrained domain Ω [170]. The neighborhood is global so that all particles are connected to each other which favors exploitation as the best neighbor corresponds to the best individual of the swarm. Consistently with the GAs, the PSO algorithm is also proposed with the two population sizes and named **PSO32** and **PSO64** in the following.

Surrogate-Assisted Evolutionary Algorithms

The investigated SAEAs use the GA and PSO algorithms, with two distinct strategies to integrate the surrogate model: namely Surrogate as a Filter (SaaF), also known as indirect fitness replacement [163], and Surrogate as an Evaluator and a Filter (SaaEF) mixing direct and indirect fitness replacement.

Surrogate-Assisted Genetic Algorithms:
In the Surrogate-Assisted Genetic Algorithm (SAGA) [171], the number of offspring produced at each generation exceeds the population size so that the EC only retains the most promising ones. To enhance diversification and give the reproduction operators a better opportunity to propose interesting candidates, multiple crossover and mutation operators are employed. In **SAGA-SaaF**, the surrogate model is only used as a filter to discard solutions considered unpromising by the EC. The offspring are produced by employing two different crossover and mutation operators, namely simulated binary and 2-point crossover, and polynomial and Gaussian mutation. Using the second version named **SAGA-SaaEF**, by using the surrogate to evaluate offspring solutions, predicted individuals may appear in the population. The offspring produced per generation are split into three sets: n_{disc} are discarded, n_{sim} are simulated and n_{pred} are only predicted before undergoing replacement. SAGA-SaaEF uses simulated binary and 2-point crossover as well as the polynomial mutation.

Surrogate-Assisted Particle Swarm Optimization:
In Surrogate-Assisted Particle Swarm Optimization (SAPSO) [171], multiple new positions are proposed for each particle at each iteration but only one position is retained per particle according to the EC. Multiple neighborhoods and formulas for velocity and position updates are leveraged to increase the chance of producing good new candidates. As for SAGA, the SAPSO is proposed with the SaaF and SaaEF variants. In **SAPSO-SaaF**, the new particles are obtained by invoking two formulas for velocity updates: the constriction and the inertia. Two types of neighborhoods are also considered to determine the local best particle: in the first one, all the particles are connected while the second one consists of 8 clusters inter-connected *via* one tie only. In **SAPSO-SaaEF**, similarly to SAGA-SaaEF, the offspring are split into three categories: simulated, predicted, and discarded. The constriction and inertia formulas are applied using the fully connected neighborhood.

In all the SAEAs, the promisingness comparison operator varies during the search. During the first half of the search, exploration is promoted as the more distant an offspring is from the set of all previously simulated candidates, the more promising it is. During the second part of the search, offspring with low predicted objective values are promoted to improve exploitation. This strategy has been shown to perform well in SAGA-SaaF and SAGA-SaaEF [172] and it is extended to SAPSO-SaaEF and SAPSO-SaaF in this study.

GA and PSO are considered in this work as they have been applied to a large range of real-world problems [32]. They offer different operators to evolve the population, resulting in different paths across the search space [173, 174]. The social interactions in the population are primordial in PSO while the GA focuses more on global elitism. Integrating the surrogate model as a filter in the evolution allows more possibility to the reproduction to come up with interesting candidates [171]. An inaccurate surrogate would erroneously discard promising candidates but the reproduction operators should be able to recover the right search path because the population is only composed of simulated solutions. Integrating the surrogate as an evaluator to replace the objective function is more error-prone as the population can embed predicted individuals. However, the computational budget is preserved [163].

A Hybrid Algorithm Combining SAEA and BOA

In this work, we introduce a way to build a hybrid algorithm based on the expected threshold from which the SAEAs perform better than the BOAs. The basic idea of this hybrid algorithm is to use a BOA to initiate the optimization process and continue with a SAEA when the efficiency of the BOA starts to drop. The initial population of the SAEA is formed with the last queries from the BOA. The hybrid algorithm is then defined by a BOA, the threshold, and a SAEA. They will be chosen in the following based on the performances of the investigated algorithms.

4.2 BOAs *versus* SAEAs

A fair comparison of the different classes of algorithms is difficult since they are not usually employed to tackle the same problems. BOAs are often used to tackle the optimization of time-consuming simulators and their budget is limited to a few hundred evaluations at most. However, we have seen that BO can be used in other contexts such as time-constrained optimization. In such context, the simulation is not the only time-consuming part of the algorithm and it is important to handle it. SAEAs are less sample-efficient but operate a lot more rapidly with a much better scalability. The proposed protocol is designed to establish the frontier of suitability between BOAs and SAEAs in regard to the computational cost of an evaluation, and the available computing resources.

4.2.1 Experimental Protocol

Implementation and Calibration

For all the SBO algorithms described in the previous section, a GP is selected as the surrogate model. The trend is assumed constant but unknown, the covariance kernel follows the Matérn $\frac{5}{2}$ model and is fitted through MLE. Those parameters are identified from previous works and literature as good default choices [7, 97]. The surrogate models are built using `GPyTorch` [96].

In the BOAs, the training set is composed of all the available data, except for ℓBSP-EGO where n_{learn} needs to be calibrated, while it is restricted to the last 96 simulations in the case of the SAEAs. Indeed, in SAEAs, the EA drives the search towards promising regions. By restricting the training set to the last evaluations, we expect the surrogate to adapt to the last identified regions while saving computational budget by limiting the costs of surrogate training. All BOAs use the `BOTorch` [97] library in addition to `GPyTorch`.

TuRBO[2], MACE[3], and ϵ-Shotgun[4] implementation information are available in their respective GitHub repositories. MIC-qEGO, gBSP-EGO, and ℓBSP-EGO are original implementations[5] based on the same libraries. In ℓBSP-EGO, the number of leaves of the binary tree and the

[2] https://botorch.org/tutorials/turbo_1
[3] https://github.com/Alaya-in-Matrix/py\acrshort{mace}
[4] https://github.com/georgedeath/eshotgun
[5] Available in https://github.com/MaGbrt/pySBO.git

training set size for the local models are set from prior experiments to $n_{learn} = \min(128; |\mathcal{D}|)$ and $n_{leaves} = 2 \times q$. The calibration of ℓBSP-EGO allows a fast training of the local models while ensuring local accuracy. Regarding the number of leaves, having n_{leaves} proportional to q is important to balance the parallel workload and to select a subset of candidates avoiding not worthy ones. The ϵ parameter in ϵ-Shotgun is set to 0.1 in accordance with the original work [77].

Both the EAs and SAEAs are implemented using the pySBO[6] Python platform [134]. The GA is tuned according to the recommendations provided in [32], namely a tournament selection of size 2, a simulated binary crossover with probability 0.9 and distribution index 2, and a polynomial mutation with probability 0.1 and distribution index 20. The PSO is calibrated according to both preliminary experiments and guidance from the literature. The constriction method is used with parameters $c_1 = c_2 = 2.05$ and $\kappa = 0.5$.

In the SaaF version of SAGA and Surrogate-Assisted Particle Swarm Optimization (SAPSO), the population size is set to $p_{size} = 32$. The number of offspring produced per generation is set to $n_{off} = 128$ among which $n_{sim} = 32$ are simulated and $n_{disc} = 96$ are discarded. In the second version, referred to with the SaaEF suffix, the population size is set to $p_{size} = 64$. The algorithms still produce $n_{off} = 128$ offspring per generation among which $n_{disc} = 64$ ones are discarded, $n_{sim} = 32$ ones are simulated, and $n_{pred} = 32$ ones are predicted.

The way to fit the GP model can be seen as a hyper-parameter as well since it is an implementation choice. An identical routine is used for all methods, except that SAEAs use an early stopping criterion which results in a very fast learning of the GP model necessary to be integrated into the SAEA framework. It is observed in [19] that it can be counterproductive to accurately train GP model as it strongly impacts the number of performed simulations when the time frame is limited. On the contrary, more time is dedicated to the model fitting in BOAs since it is not used simply as a filter but as an oracle to select the candidates, which justifies the training of more accurate models.

The parallel implementation is handled with MPI for Python, MPI4Py [175].

Protocol and Objectives

This protocol is designed to estimate the budget range in terms of number of simulations where BOAs and SAEAs are the most efficient. We expect to observe a threshold that will guide the choice of the appropriate category of algorithm and to design hybrid algorithms. Ideally, this threshold is determined for each problem. However, in this study, we aim at providing a generic threshold that can be refined afterward if needed for more precise purposes.

To account for the acquisition time of the different methods, the global budget is defined as a limited wall-clock time. We set up several experimental settings defined by the simulation time t_{sim} and the number of used computing cores n_{cores}. Each (t_{sim}, n_{cores}) pair is representative of a scenario the user can face. This set of objective functions is chosen large enough to be representative of various difficulties relative to the landscape of the function, such as multi-modality.

[6] https://github.com/GuillaumeBriffoteaux/pySBO

The numerical comparison includes 18 benchmark functions of 10 decision variables each, optimized with the 15 algorithms presented in the previous sections. The benchmark consists of 15 functions defined in CEC2015 [176] and 3 additional problems including Alpine 02 and two custom functions. This set of functions is a subset of CEC2014 [177], whose functions are implemented in `Pygmo2` [120]. For the sake of clarity and consistency with the names of functions in `Pygmo2`, each of them is referred to with its CEC2014 identifier CEC2014_i, where $i \in \{2, 3, 6, 11, 12, 13, 14, 15, 16, 17, 19, 22, 23, 25, 27\}$. We supplement the benchmark with the highly multi-modal Alpine 02 problem, defined in [178] and already used in Chapter 3. In addition, we build two additional artificial problems that we define according to the hybrid and composition construction schemes of CEC2014 [177]. The first custom problem is called Hybrid_AW and is obtained by hybridization of the Alpine 02 and the Weierstrass functions. The second custom function is denoted by Composition_AS and is generated *via* the composition of the Alpine 02 and the Schwefel functions. The Alpine 02, Hybrid_AW and Composition_AS functions are also available in the `pySBO` GitHub repository.

The computational budget for one search is set to 20 minutes on different numbers of available CPU cores $n_{cores} \in \{2, 4, 8, 16, 32\}$. Since the benchmark functions have negligible time cost, we add an artificial cost of t_{sim} seconds each time we evaluate a solution on a single core. We assume that the evaluation time is constant so that the synchronous parallel evaluation of n_{cores} candidates also takes t_{sim} seconds. This allows one to study the impact of the evaluation time on the number of iterations or generations by artificially increasing t_{sim} *a posteriori*. The considered values for t_{sim} are $\{5, 15, 30, 60\}$ seconds.

The optimization runs are repeated 10 times on each benchmark function. For each repetition, the initial sampling of 96 points is the same across the algorithms. With the 18 functions, each algorithm is evaluated based on 180 executions. All the experiments are realized on the same machine featuring Intel Xeon Gold 5220 CPU and stemming from the Grid5000 infrastructure for distributed computing [5]. The global experimental plan represents a total of $13,500$ distinct optimization exercises.

Significance of the results

The 15 algorithms are compared for each (t_{sim}, n_{cores}) pair by computing the Friedman's test rank. For each of the 18 benchmark functions, the methods are ranked from best to worst based on the average outcome of the 10 repetitions. Finally, the ranks are averaged to give the Friedman's rank displayed in Table 4.1. A gradient from blue to white is applied to the table to be more visually readable. The lower ranks indicate a better performance and are highlighted by a blue background while the higher ranks indicate a lesser efficiency. For each (t_{sim}, n_{cores}) pair and a fixed budget of 20 minutes, we associate the theoretical upper bound of function evaluations, denoted ρ, which is reached if the query for new candidates has no cost (*e.g.*, pure random search). For instance, our 20-minute budget allows a maximum of $\rho = 40$ simulations if $t_{sim} = 60$ and $n_{cores} = 2$.

Based on the rank of each algorithm, we can assess the significance of the difference between the outcomes of the algorithms by performing the Friedman's Two-way Analysis of Variance, and the *post-hoc* pairwise Friedman's test [179]. Figure 4.1 displays the results of the pairwise tests. If the presented p-value is significant (< 0.1), the cell background is colored in blue

shades. Darker colors indicate higher significance. In addition, in case of a significant difference between the two algorithms, an arrow indicates which one is to be preferred.

Both Table 4.1 and Figure 4.1 contain the results of the hybrid algorithm (namely TuRBO-SAGA) so that it is easily comparable with the other algorithms. However, the it will be analyzed after the determination of both the threshold and the best performing algorithms.

4.2.2 Determination of the Threshold

We define the threshold as the budget range from which BOAs face a drop of efficiency while SAEAs begin to compete with BOAs in terms of optimization outcomes. The characterization of the latter helps practitioners to efficiently choose the suitable family of algorithms. In Table 4.1, for each (t_{sim}, n_{cores}) pair, we observe different ranks. The significance of the differences is confirmed by the Friedman's Test (Two Way Analysis of Variance by Ranks) for which the p-value is far smaller than 0.01. The p-values of the pairwise comparisons shown in Figure 4.1 clarify the differences.

More precisely, in Table 4.1, we observe a loss in the performance of the BOAs when the budget increases. Indeed, most of the blue shaded cells can be observed for BOAs for $n_{cores} = 2$ while clearer and white cells appear progressively until representing the majority of cells for $n_{cores} = 32$. Conversely, an improvement of the rank is to be noted for EAs and SAEAs when the budget increases as the cells' color tends towards darker blue as n_{cores} increases. These observations are similar for any fixed value of n_{cores}, with decreasing t_{sim}. In both cases, increasing n_{cores} or reducing t_{sim} means a budget increase which translates into a loss of performance from the BOAs compared to EAs and SAEAs.

In Table 4.1, the best-performing BOA is TuRBO and the best SAEA is SAGA-SaaF. Regarding the p-values of the pairwise comparisons displayed in Figure 4.1, we observe a significant advantage for BOAs, and particularly TuRBO for budgets smaller than $(t_{sim}, n_{cores}) = (15, 2)$, $(15, 4)$, $(15, 8)$, $(60, 16)$ and $(60, 32)$, which indicates a shift when the budget approaches 640 maximum simulations (*i.e.* $\rho = 640$). A clear advantage is visible for any BOA when the budget is smaller than 160 and TuRBO extends the statistical dominance of BO up to 320 simulations. Despite a better average performance of TuRBO on higher budgets (*e.g.*, $(t_{sim}, n_{cores}) = (5, 2)$, $(15, 4)$, $(15, 8)$, $(30, 16)$ and $(60, 32)$) the confidence provided by the rank test results shown in Figure 4.1 is not always sufficient to assert the dominance ($p_{val} > 5\%$). The lower confidence is often due to the SAGA-SaaF algorithm that achieves good performances, often similar or better than most BOAs except TuRBO as soon as the budget exceeds 640 simulations (see Table 4.1). For higher ρ, ℓBSP-EGO shows advantages compared to TuRBO, however, it coincides with the better performances of evolutionary approaches. A transition to SAGA-SaaF seems to occur when the budget exceeds $(t_{sim}, n_{cores}) = (5, 4), (5, 8), (15, 16)$, and $(30, 32)$ representing an upper bound of 960, 1920, 1280, and 1280 simulations respectively. Nevertheless, the advantage of SAEAs and more precisely SAGA-SaaF is significant over the BOAs and TuRBO only for $(t_{sim} = 5, n_{cores} \geq 4)$ and $(t_{sim} = 15, n_{cores} \geq 16)$ which means a minimum of 960 simulations. One can also note that the SAEAs and mainly SAGA-SaaF remain very efficient for budgets of a few thousands of simulations. In addition, we do not observe any context where EAs are consistently outperforming SAEAs.

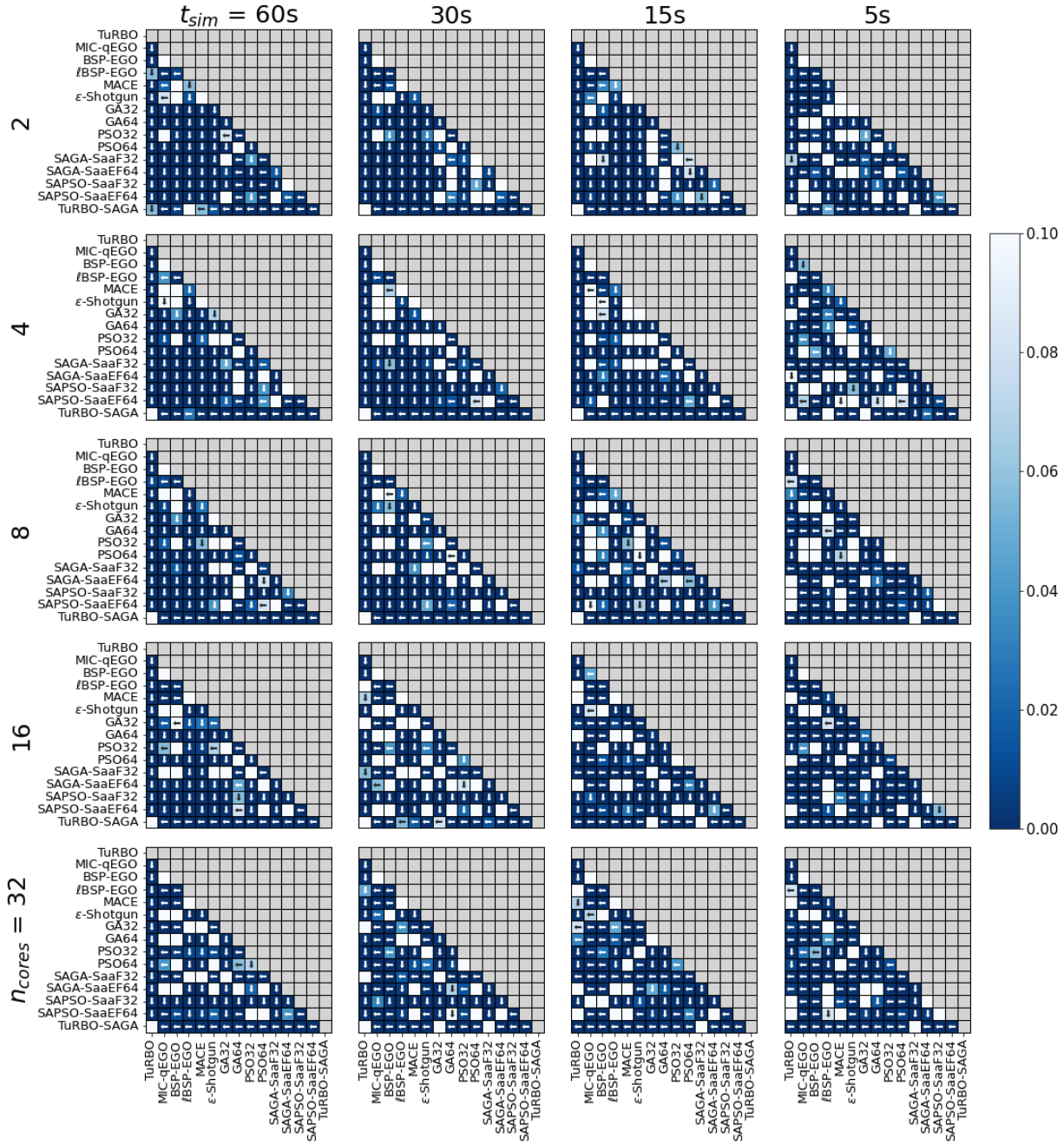


Figure 4.1: p-values of the pairwise comparison *post-hoc* Friedman's rank test
Low values indicate statistically significant differences, and are highlighted by the color scale. In case of p-value < 0.1, an arrow indicates the direction of the algorithm outperforming the other.

Table 4.1: Friedman's rank of the algorithms in relation to n_{cores} and t_{sim}.
Lower values indicate a better performance of the algorithm and are highlighted by a darker cell background. ρ indicates the maximum number of simulations possible within a fixed budget of 20 minutes using n_{cores} processing cores for a simulation lasting t_{sim} seconds.

	Class		BOAs						EAs				SAEAs				Hybrid
	Method	ρ	TuRBO	MIC-qEGO	gBSP-EGO	ℓBSP-EGO	MACE	ϵ-Shotgun	GA32	GA64	PSO32	PSO64	SAGA-SaaF32	SAGA-SaaEF64	SAPSO-SaaF32	SAPSO-SaaEF64	TuRBO-SAGA
n_{cores}	t_{sim}																
2	60	40	2.67	6.78	5.78	3.94	5.22	5.61	8.94	14.00	7.78	15.00	9.17	10.78	11.22	9.17	3.94
	30	80	2.06	7.50	6.94	4.50	5.28	6.89	9.22	11.61	8.33	10.94	10.00	11.78	12.28	10.17	2.50
	15	160	1.78	8.17	7.61	4.83	6.17	6.72	9.22	11.94	8.67	9.94	8.78	11.11	12.67	10.06	2.33
	5	480	3.78	10.83	9.78	5.67	7.67	7.67	6.61	10.39	7.94	9.78	5.00	8.17	11.89	10.50	4.33
4	60	80	2.61	6.00	6.94	4.61	6.17	7.11	8.33	11.78	7.72	11.28	9.67	12.11	12.67	9.89	3.11
	30	160	1.83	7.11	7.61	5.33	6.39	7.11	8.06	11.50	7.28	11.00	8.89	12.06	13.61	9.83	2.39
	15	320	2.56	7.94	8.72	5.33	6.83	7.61	7.56	11.67	6.94	10.89	7.00	10.17	13.89	9.56	3.33
	5	960	5.94	10.33	11.61	6.56	8.00	9.61	5.11	7.94	8.94	10.28	3.11	7.06	10.89	9.11	5.50
8	60	160	2.72	6.28	7.17	4.50	6.61	8.11	8.56	12.33	7.89	10.67	8.78	11.83	13.28	9.50	1.78
	30	320	2.78	7.39	7.61	4.89	6.50	8.89	7.00	11.78	7.50	10.67	7.89	11.06	13.61	10.22	2.22
	15	640	4.00	9.22	8.78	5.83	7.17	9.11	5.44	10.17	8.44	10.22	5.67	8.94	13.11	10.33	3.56
	5	1920	7.67	10.94	11.39	6.50	9.11	11.28	5.78	5.33	11.11	10.33	3.50	6.83	8.44	8.67	3.11
16	60	320	3.11	8.50	8.00	5.22	5.39	8.44	6.89	11.78	7.22	11.06	7.72	10.44	13.06	10.61	2.56
	30	640	4.61	10.39	9.33	5.50	5.83	9.39	5.39	9.94	7.94	9.33	5.89	9.11	13.61	9.50	4.22
	15	1280	6.67	11.72	10.39	6.56	7.11	10.56	4.72	6.67	9.67	8.72	3.72	7.28	13.33	8.61	4.28
	5	3840	9.06	12.44	11.89	6.78	8.61	11.94	5.61	4.11	11.06	9.28	3.72	6.39	7.17	8.44	3.50
32	60	640	3.44	10.61	9.89	6.22	6.28	9.72	5.72	10.50	8.00	9.22	5.89	9.56	13.22	8.17	3.56
	30	1280	4.83	11.83	11.22	6.17	7.11	10.28	4.78	6.94	9.83	8.78	4.56	8.17	13.28	8.06	4.17
	15	2560	6.72	12.28	11.78	6.89	7.94	11.06	5.56	5.28	10.28	8.89	3.06	6.89	12.72	7.44	3.22
	5	7680	7.83	12.89	12.67	6.67	8.61	11.94	6.00	5.28	11.39	9.67	3.33	5.89	6.89	7.83	3.11

Among the EAs, it is clear through Table 4.1 that a small population, enhancing intensification, is preferred when dealing with very limited budgets as those with $n_{cores} = 2$. Actually, for such budgets, exploitation is to be favored. The suitability of larger populations arises when the budget increases as shows the rank of GA64 for $t_{sim} \leqslant 15$ and $n_{cores} \geqslant 16$.

Within the BOAs, some differences are also observed. Clearly, TuRBO outperforms the five others in most cases, except for larger budgets where ℓBSP-EGO shows its interest. The good average performance of TuRBO can be explained by the local AP performed in a specific sub-region. This also explains the good performance of ℓBSP-EGO in comparison with MACE, MIC-qEGO, gBSP-EGO, and ϵ-Shotgun, whose performances are generally close. Similarly to TuRBO, ℓBSP-EGO intends to focus on promising sub-regions as the budget fades, which seems to be a profitable strategy. The MACE acquisition strategy that consists in finding a trade-off between several AFs also seems to perform well for small to moderate budgets ($\rho \leq 640$). The MACE algorithm seems to efficiently use an increased batch size since its performances do not drop when n_{cores} increases, contrary to other BOAs, except TuRBO. Resorting to complementary AFs

seems beneficial, especially for large batches. In comparison with the benchmark analysis of Section 3.3, we clearly see the impact of the number of simulations here.

Table 4.2 provides a concise summary of Table 4.1, showing for each (t_{sim}, n_{cores}) pair the best method to use among BO, EA, and SAEA. Bold font indicates a statistical dominance of at least one algorithm of the dominant class compared to all the algorithms of other classes. The threshold from which switching from BOAs to SAEAs consequently lies between 640 and 960 expected simulations.

Table 4.2: Recommendation of the method according to t_{sim} and n_{cores}, and their equivalent in terms of maximum expected simulations (ρ, in parenthesis). Bold font indicates a stronger confidence (low p-value) in the results.

t_{sim} / n_{cores}	2	4	8	16	32
60	**BO** (40)	**BO** (80)	**BO** (160)	**BO** (320)	BO (640)
30	**BO** (80)	**BO** (160)	**BO** (320)	BO (640)	SAEA (1280)
15	**BO** (160)	**BO** (320)	**BO** (640)	**SAEA** (1280)	**SAEA** (2560)
5	BO (480)	**SAEA** (960)	**SAEA** (1920)	**SAEA** (3840)	**SAEA** (7680)

4.2.3 Efficiency of the Acquisition Processes

In order to assess the parallel efficiency of the algorithms, we observe the number of performed simulations for each algorithm. We report in Table 4.3 the ratio between the average number of simulations (over the 10 repetitions of all the benchmark functions), and the theoretical upper bound of number of simulations (ρ). It allows the estimation of the proportion of the time budget spent in the acquisition of the candidates compared to the time spent in simulations. Hence, it represents the scalability. However, it is not entirely reliable since the landscape of the benchmark functions may influence the number of simulations, mainly because of the different learning times of the surrogate model. In Table 4.3, we can see that the scalability of BOAs is strongly impacted by the increase of ρ. The displayed ratio radically falls down when approaching $\rho = 1000$, indicating a time-consuming AP and surrogate fitting for all BOAs. On the contrary, EAs and SAEAs achieve excellent scalability, except for very low batch sizes. Indeed, when the remaining budget is not sufficient to execute a new generation, the algorithm is stopped prematurely. This explains why GA64 and PSO64 cannot perform a single generation when $\rho = 40$.

However, the scalability must also take into account the effectiveness of the APs when increasing the batch size. Then, we relate the quality of the final outcome with the number of simulations. In other terms, does the algorithm benefit from the parallel execution of large batches of simulations, or is it preferable to perform more iterations with smaller batches? This was previously referred to as the batch effectiveness.

Table 4.3: Efficiency of the algorithms in terms of number of simulations.
The ratio between the averaged number of simulations over all the test problems and its theoretical maximum ρ is close to 1 (dark background) if most of the time budget of the algorithm is spent in simulations.

	Class		BOAs						EAs				SAEAs				Hybrid
	Method	ρ	TuRBO	MIC-qEGO	gBSP-EGO	ℓBSP-EGO	MACE	ϵ-Shotgun	GA32	GA64	PSO32	PSO64	SAGA-SaaF32	SAGA-SaaEF64	SAPSO-SaaF32	SAPSO-SaaEF64	TuRBO-SAGA
n_{cores}	t_{sim}																
2	60	40	0.95	0.95	0.95	0.95	0.85	0.95	0.80	0.00	0.80	0.00	0.80	0.80	0.80	0.80	0.95
	30	80	0.93	0.93	0.93	0.96	0.77	0.91	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.93
	15	160	0.88	0.87	0.87	0.93	0.63	0.84	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.87
	5	480	0.62	0.60	0.60	0.82	0.35	0.56	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.59
4	60	80	0.94	0.93	0.94	0.95	0.87	0.95	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.95
	30	160	0.91	0.89	0.90	0.95	0.79	0.89	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.92
	15	320	0.82	0.77	0.78	0.92	0.64	0.78	0.90	0.80	0.90	0.80	0.90	0.90	0.90	0.90	0.82
	5	960	0.46	0.42	0.43	0.79	0.33	0.43	0.97	0.93	0.97	0.93	0.97	0.96	0.97	0.97	0.71
8	60	160	0.92	0.88	0.92	0.95	0.88	0.92	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.94
	30	320	0.86	0.78	0.85	0.94	0.78	0.85	0.90	0.80	0.90	0.80	0.90	0.89	0.90	0.90	0.87
	15	640	0.68	0.58	0.66	0.89	0.58	0.66	0.95	0.90	0.95	0.90	0.95	0.94	0.95	0.95	0.80
	5	1920	0.31	0.26	0.30	0.71	0.26	0.30	0.98	0.97	0.98	0.97	0.98	0.96	0.95	0.93	0.81
16	60	320	0.87	0.75	0.87	0.95	0.83	0.88	0.90	0.80	0.90	0.80	0.90	0.89	0.90	0.90	0.89
	30	640	0.73	0.57	0.70	0.91	0.66	0.74	0.95	0.90	0.95	0.90	0.95	0.93	0.95	0.95	0.86
	15	1280	0.49	0.36	0.45	0.83	0.43	0.49	0.97	0.95	0.97	0.95	0.97	0.95	0.97	0.95	0.87
	5	3840	0.19	0.14	0.18	0.57	0.17	0.19	0.99	0.98	0.99	0.98	0.97	0.95	0.87	0.79	0.85
32	60	640	0.75	0.55	0.77	0.92	0.74	0.82	0.95	0.90	0.95	0.90	0.95	0.92	0.95	0.95	0.90
	30	1280	0.54	0.36	0.52	0.88	0.51	0.57	0.97	0.95	0.97	0.95	0.97	0.94	0.97	0.95	0.90
	15	2560	0.32	0.20	0.30	0.74	0.30	0.34	0.99	0.97	0.99	0.97	0.97	0.94	0.93	0.87	0.88
	5	7680	0.12	0.07	0.11	0.42	0.11	0.13	1.00	0.99	1.00	0.99	0.94	0.91	0.71	0.58	0.80

It is generally accepted that BO is particularly efficient in addressing very expensive objective functions because the time cost associated with the surrogate model fitting and the acquisition of candidates remains negligible compared to the simulation time. According to Table 4.3, it is clear that BOAs are sample-efficient for tight budgets. The ratio is close to 1, and the performances displayed in Table 4.1 are also very good. For (t_{sim}, n_{cores}) corresponding to small budgets (less than 320 simulations in Table 4.1), BOAs spend the major part of the 20-minute budget in simulations and BOAs are strongly recommended. However, when increasing n_{cores}, the BOAs seem to be less efficient.

Indeed, with increased parallel computing capabilities and a shorter simulation time, the data size increases rapidly, leading most of the BOAs to scale very poorly, essentially (but not only) because of the global GP model. Only ℓBSP-EGO manages to notably increase the number of simulations by relying on local surrogate models learned over the subset of data, and local parallel APs. This allows ℓBSP-EGO to generally outperform MIC-qEGO, MACE, gBSP-EGO

and ϵ-Shotgun. Nevertheless, the higher number of simulations does not always provide better outcomes than TuRBO. Furthermore, its parallel efficiency also drops after a few thousand simulations. The APs are also responsible for the differences in terms of number of simulations between BOAs. MIC-qEGO and MACE have time-consuming acquisition strategies, relying on multiple updates of the surrogate model and multiple optimizations of the AFs for the first one, while the second one resorts to multi-objective optimization. TuRBO is slightly faster by means of the local optimization inside a trust region, gBSP-EGO parallelizes the process by operating multiple local optimizations, and ϵ-Shotgun only optimizes the AF once.

On the other hand, both EAs and SAEAs are not limited by the data set size since they operate on populations of fixed size and rely on fast-to-compute evolutionary operators. Furthermore, SAEAs only consider the last 96 simulations for fitting the GP model and use an early stopping criterion to reduce the training time. It has to be noted that for the EAs and SAEAs, a generation is performed only if the remaining time budget is sufficient. This observation apart, EAs show an efficient scaling by achieving almost the maximum number of possible simulations. However, a perceptible difference between the SaaF and SaaEF strategies is observed in Table 4.3 to the advantage of the SaaF one. Indeed, the SaaEF strategy requires a bit more computation than SaaF for each generation. Consequently, the advantage of SaaF becomes perceptible for larger budgets. A larger difference is observed between SAGAs and SAPSO algorithms. Actually, the evolution of the population in GAs requires less data and computation than the evolution of the swarm in the PSO algorithm. The latter becomes substantial after a few thousand evaluations and SAGAs appear to be a better choice in this situation. Furthermore, the population-based algorithms show improved outcomes compared to BOAs when increasing ρ, which indicates a better batch effectiveness in addition to the very good scalability.

In BOAs, the whole batch of candidates supplements the data set at each iteration which becomes too large for GP to be fitted fast. Even though ℓBSP-EGO appears to outperform all BOAs when the budget exceeds $\rho = 1920$, SAEAs are a better choice on average. The conjunction of the two aspects, namely the costly training and difficulty to point out numerous promising new candidates, leads to bad parallel efficiency and is referred to as the breaking point of BOAs in [22] and was also observed in Section 3.3. These two aspects explain why SAEAs are clearly preferred when the budget approaches 1000 simulations. The best performance of BOAs seems to be achieved with simulation time greater than 15 seconds and a number of available cores smaller than 8. This allows a maximum of 640 simulations in this protocol.

4.3 Hybrid Methods Combining SAEA and BOA

4.3.1 Threshold-based Hybrid Algorithm

Based on the previous observations and identified limitations of BOAs, a hybrid algorithm is built upon the TuRBO and SAGA-SaaF algorithms. The basic idea is to benefit from the efficiency of the BOA in the early stages of the optimization process to create a population. The latter is then used after a certain number of simulations to initialize a SAEA to benefit from its fast execution and good scalability. The early stages of the search are performed by TuRBO and

Algorithm 10 Pseudo code of the hybrid algorithm TuRBO-SAGA

Input

f: objective function
$\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$: Initial DoE
$\mathcal{T}$: Trust region
$\mathcal{P}$, p_{size}: population and its size
t: threshold

1: **while** $|\mathcal{D}| < t$ **and** budget available **do**
2: $\quad \mathcal{M} = \mathcal{GP}(\mathcal{D})$
3: $\quad X_{next} =$ TuRBO $acquisition\_process(\mathcal{M}, \mathcal{T})$
4: $\quad \boldsymbol{y}_{next} = f(X_{next})$
5: $\quad \mathcal{D} = \mathcal{D} \cup \{X_{next}, \boldsymbol{y}_{next}\}$
6: $\quad \mathcal{T} = update\_trust\_region\,(\mathcal{T})$
7: **end while**
8: $\mathcal{P}$ = initialise_population$(p_{size}, \mathcal{D})$
9: **while** budget available **do**
10: $\quad \mathcal{M} = \mathcal{GP}(\mathcal{D})$
11: $\quad \mathcal{P}_{par} = select\_parents(\mathcal{P})$
12: $\quad \mathcal{P}_{off} = generate\_offspring(\mathcal{P}_{par})$
13: $\quad X_{next} = evolution\_control(\mathcal{M}, \mathcal{P}_{off})$
14: $\quad \boldsymbol{y}_{next} = f(X_{next})$
15: $\quad \mathcal{P} = update\_population(\mathcal{P}, (X_{next}, \boldsymbol{y}_{next}))$
16: **end while**
17: **return** best$(\boldsymbol{x}, y)$

the switch to SAGA-SaaF occurs when the threshold is reached. The threshold lies for ρ ranging from 640 and 960. In BO, this would correspond to 450 effective simulations as training the surrogate and acquiring new solutions are not computation-free operations. Table 4.2 suggests that after 320 possible simulations, BOAs start to drop in terms of efficiency. Adding the 96 initial samples, we propose to set the threshold to 420: as soon as the data set holds 420 points, the algorithm switches from TuRBO to SAGA-SaaF. The population of the GA is initialized based on the last simulations from TuRBO.

Algorithm 10 illustrates the serial application of the two algorithms, where lines 1 to 7 depict the TuRBO algorithm with each iteration involving a GP fitting, an AP, the simulations, and the updates of the data set and trust region. Once the threshold is reached, the SAGA-SaaF algorithm takes over and operates from line 9 to 15 by successively fitting a GP model, selecting parents to generate offspring that pass through the EC to filter them out before resorting to the simulator, and updating the population.

The last column of Table 4.1 and Table 4.3 displays the performance indicators of the described hybrid algorithm, namely TuRBO-SAGA. Looking at Table 4.1, except for $(t_{sim}, n_{cores}) = (5, 4)$, the new algorithm provides similar performances than TuRBO for low computational budgets. Moreover, TuRBO-SAGA improves over SAGA-SaaF for larger budgets

taking advantage of the initial population composed of solutions acquired by TuRBO (line 8 in Algorithm 10). Regarding Table 4.3, we can see that the scalability of the approach does not suffer from the BO's early-optimization strategy and offers an average efficiency between 0.8 and 0.9. The switch occurs before TuRBO becomes too slow. Its good overall performance is significant in most contexts according to Figure 4.1. Despite the very simple rule guiding the algorithm, it offers a general suitable option, and mostly many perspectives to design hybrid algorithms. In order to confirm the previous results, we propose to apply the same strategy to new problems that have not been used to determine the threshold.

4.3.2 Validation Through Unseen Problems

Two additional problems from engineering are used to validate TuRBO-SAGA on problems that have not been used to find the threshold. They are the multi-product batch plant problem [180] and the rolling element bearing design problem [181], both implemented in [182]. They both possess constraints in their formulation which are dealt with by penalizing the objective function as suggested in [182]. Since the hybrid algorithm is built on top of TuRBO and SAGA-SaaF, which seems to outperform in most contexts their contestants in their respective classes, only TuRBO, SAGA-SaaF and TuRBO-SAGA are compared based on this set of problems.

The experimental protocol remains the same as in the previous section. Figure 4.2 and Figure 4.3 display the evolution of the best-known output according to the number of evaluations in semi-logarithmic scale. On each figure, we observe in solid lines the average best output value, and its associated standard deviation is represented using dashed lines in the same color. Vertical dotted lines illustrate the average number of simulations that would be expected if the evaluation lasted $t_{sim} \in \{60, 30, 15, 5\}$ seconds. In particular, we observe the evolution of the best objective value around the threshold (324 evaluations on the graphs) and after, by zooming in on the graphs. It has to be noted that in order to plot the average outcome, the curves are truncated to the minimum number of evaluations over the 10 repetitions. Therefore, the average number of simulations given by the vertical doted-lines accounting for $t_{sim} = 5$ might be slightly shifted to the right compared to where the curves end.

The first overall observation is that the results are consistent with the previous section: the hybrid algorithm performs well and achieves fast improvement at the beginning using the TuRBO part before the threshold, and continues improving the objective value after it while TuRBO does not, through the switch to SAGA. It is particularly observable for the multi-product batch plant problem displayed in Figure 4.3. The threshold seems to generalize since for a number of evaluations smaller than 500, TuRBO is always to be preferred. According to Table 4.3, the parallel efficiency of TuRBO is approximately 0.7 when $\rho = 640$, which corresponds to about 450 real simulations. The two observations are then consistent as well. Figure 4.2 and Figure 4.3 also illustrate the better scalability in terms of number of evaluations and obtained outcomes. Firstly, in terms of number of evaluations, the SAGA-based algorithms achieve many more evaluations than TuRBO in the given time budget. This is illustrated by the vertical doted-lines on which we can see that the smaller t_{sim} is, the bigger the difference between TuRBO and SAGA-based algorithms. Secondly, regarding the outcomes, we can see especially for the multi-product batch plant problem that increasing the number of computing units comes with an improvement in the final outcomes for the hybrid algorithm and the SAEA. On the contrary,

TuRBO does not benefit as much as its contestants from the higher computing capabilities regarding the number of simulations. However, we can see especially in Figure 4.3 that increasing n_{cores} also improves the final outcome.

4.3.3 Conclusions and Discussion

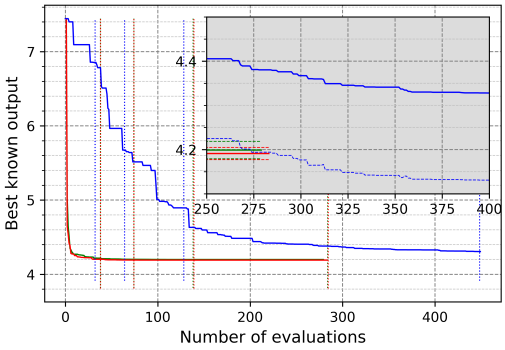
In this study, we compared 14 algorithms belonging to 3 classes of GO algorithms: BOAs, SAEAs, and surrogate-free EAs. In addition, we derive a hybrid algorithm combining two of the most efficient investigated algorithms. The implemented protocol accounts for the time cost of the objective function and the number of available computing units, which offers a different point of view to better choose a suitable approach for solving a black-box problem. We examined principally two aspects which are their performance regarding the final outcome with respect to the computational budget and their parallel efficiency in regard to the data set size and the available computing power.

The primary objective was to determine the threshold in terms of simulation budget from which on SAEAs outperform BOAs using an experimental protocol accounting for the simulation time. The proposed experimental study establishes the frontier of suitability of BOAs and SAEAs between 640 and 960 simulations (ρ). This threshold can be used by practitioners to choose the suitable class of algorithms accounting for their operational constraints (time budget, number of processing units, and simulation time).

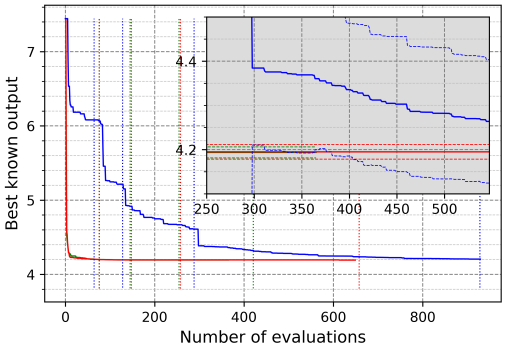
The present study also highlighted two algorithms outperforming their contestants in their respective classes: TuRBO among the BOAs, and SAGA-SaaF among SAEAs. This emphasizes the importance that is rightfully attributed to the acquisition strategy since the performance strongly depends on this choice. The SAGA-SaaF algorithm almost always achieves better or equivalent results compared to any investigated surrogate-free EAs for any considered budget, and can even compete with TuRBO on moderate budgets (more than 640 simulations). SAGA-SaaF also gets the best scalability among the considered SBO algorithms. Actually, unlike BOAs it scales very well with the number of available cores, and the outcomes are improved by the larger sampling offered by the parallel machines. Indeed, all investigated BOAs are very sample-efficient for low batch sizes (≤ 8), the time-consuming acquisition strategy pays off when the number of simulations is very limited.

Nevertheless, their scalability is limited by the low effectiveness of large batches (≥ 16) of candidates and strongly hampered by the fast-increasing data set size causing time-consuming model fittings. Only TuRBO, ℓBSP-EGO, and MACE manage to get a good batch effectiveness regarding the final outcome and still perform well with $n_{cores} \geq 16$. As for the scalability, only ℓBSP-EGO manages to significantly increase the number of simulations with the batch size. Given sufficient time budget, ℓBSP-EGO outperforms all the other BOAs, but is also outperformed by the best-performing SAEAs. This confirms the conclusion of Chapter 3 that local acquisition strategies such as in TuRBO or ℓBSP-EGO, as well as multi-criteria APs such as in MACE improve the batch effectiveness.

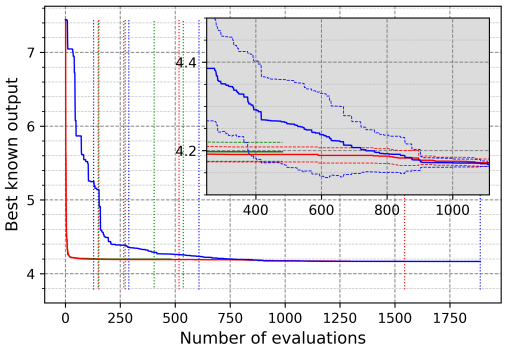
The identified threshold offers a new perspective to design hybrid algorithms that fit any context, regardless of the simulation time or available computational resources. A hybrid TuRBO-SAGA algorithm is designed to switch from TuRBO to SAGA-SaaF when a threshold is



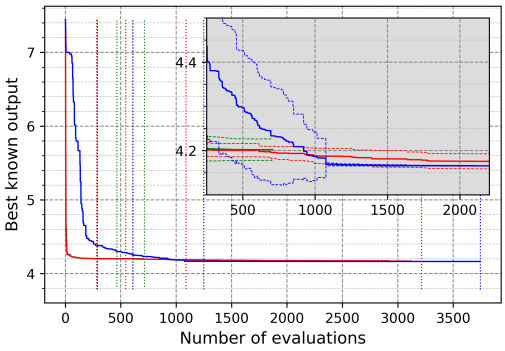
(a) $n_{cores} = 2$



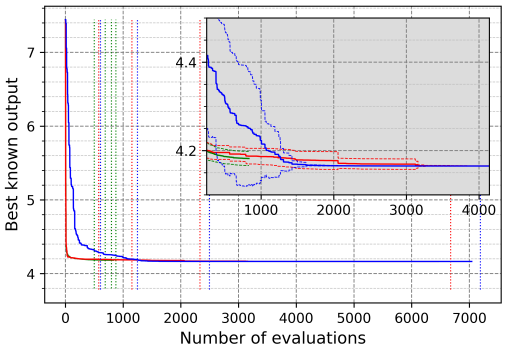
(b) $n_{cores} = 4$



(c) $n_{cores} = 8$



(d) $n_{cores} = 16$



(e) $n_{cores} = 32$

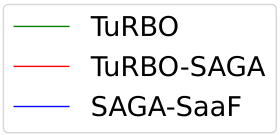
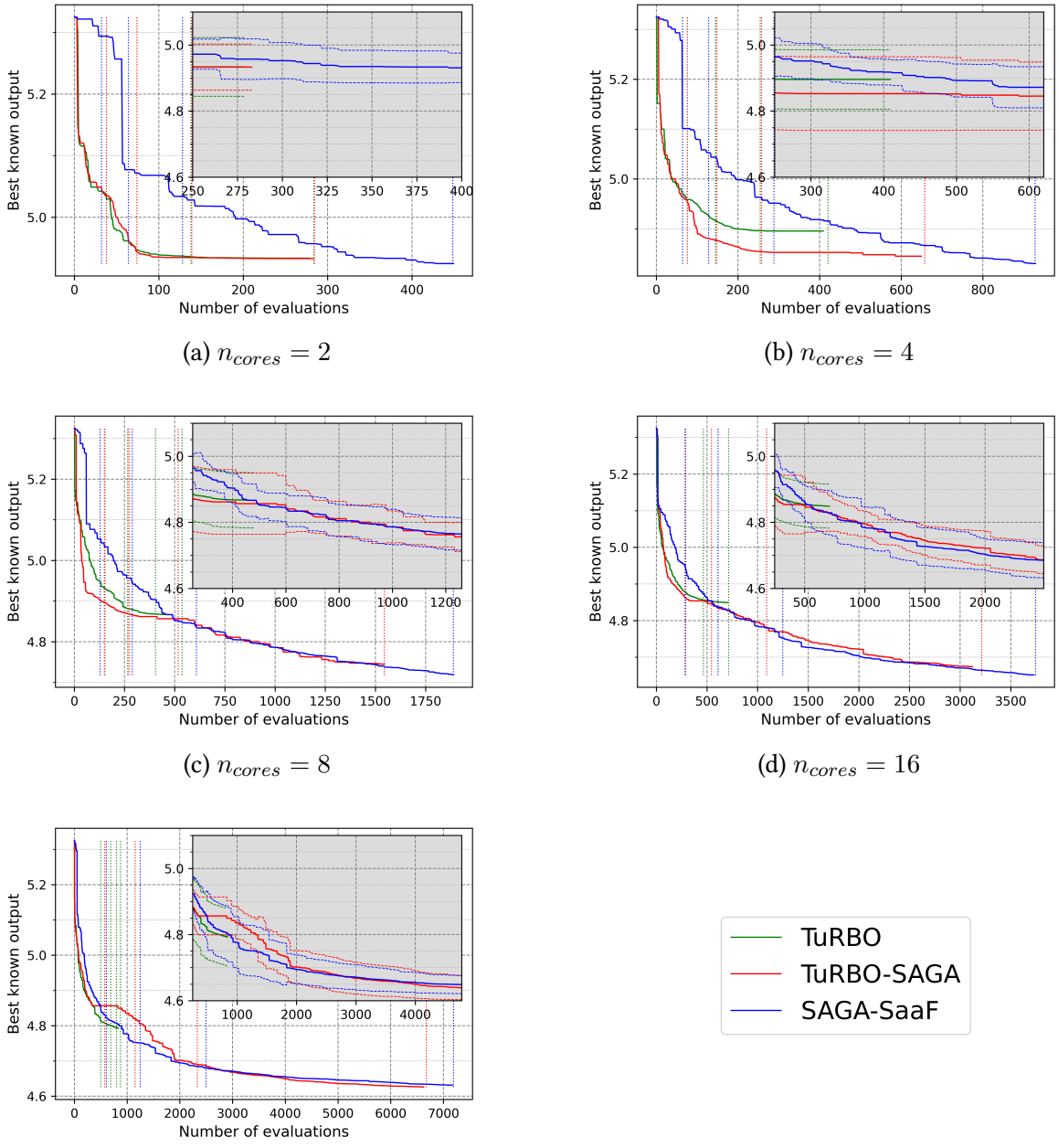


Figure 4.2: Evolution of the best average outcome in a function of the number of evaluations for the **rolling element bearing design problem**. Dashed-lines indicates the standard-deviation and vertical dotted-lines the mean number of evaluations for the different t_{sim}.



(a) $n_{cores} = 2$

(b) $n_{cores} = 4$

(c) $n_{cores} = 8$

(d) $n_{cores} = 16$

(e) $n_{cores} = 32$

Figure 4.3: Evolution of the best average outcome in a function of the number of evaluations for the **multi-product batch plant problem**. Dashed-lines indicates the standard-deviation and vertical dotted-lines the mean number of evaluations for the different t_{sim}.

reached. The proposed TuRBO-SAGA algorithm is first investigated on the initial benchmark set and compared to the other approaches. It presents the advantages of BOAs in the beginning of the optimization and the advantages of SAEAs when the data set gets bigger, if the budget is large enough. In the following, the observed results are validated on two engineering problems for which we observe consistent performance of the hybrid algorithm. Its good observed performance is validated afterward The threshold obtained on the benchmark problems generalizes well to the considered real-world applications.

BOAs are known to be more efficient on low-dimensional problems. As a matter of fact, the dimension of the problem also impacts the surrogate fitting time, the AP time, and thus the global efficiency of the algorithms. This is particularly important when the budget is defined as a limited time. The impact of the problem dimension will be investigated in the future as a complementary criterion to the computational budget to select the appropriate algorithm. The poor scalability of BOAs in terms of optimization quality and number of simulations suggests that multiple BOAs operating on small batches are preferable to a single algorithm with a large batch size. Another question that is not addressed is the time dedicated to the model learning. We clearly see that the results are dependent on the quality of the model. Nevertheless, a cheaper model would enable more optimization cycles and possibly improve the final outcome. This is difficult to quantify without a proper analysis. The combination of BOAs and SAEAs appears as a suitable direction for improving the current SBO algorithms and can be supplemented by different BOAs operating in parallel to maximize the parallel efficiency and batch effectiveness.

4.4 Opening to Higher Dimensional Problems

BO is known to not scale well with the problem dimension [18, 124, 156]. GPs are much less reliable since the distance between the design points does not allow an efficient correlation between them. Consequently, it would require a lot more observations and we observed in previous sections and chapters that BOAs do not scale well with the data set size either. In addition, a large dimension implies more hyper-parameters, which also contributes to the inefficiency and computational burden of BO.

The limit in terms of dimension from where BO loses efficiency is not well-established. In the previous chapter we already observed significant differences regarding the performances of the algorithms between the problems involving different numbers of design variables. However, several works mention 20 as the limit for *traditional* BOAs [3, 7, 183]. Beyond this limit, three main approaches to improve the performances of BO are reported in a survey from Binois and Wycoff [3]. The first one is to reduce the dimension by removing the design variables with the least influence on the output. This is also mentioned as feature selection in opposition to feature extraction which is another alternative consisting of creating new features carrying the maximum information from the initial features in less variables (*e.g.* principal component analysis). The last mentioned direction is to assume additivity of effects of the variables, such as in additive kernels GP [44]. Nevertheless, those approaches require additional assumptions over the objective function, which we are not able to assert.

We have seen that partitioning the search space and relying on several criteria improved the performance of BOAs. Particularly, TuRBO is initially designed to tackle higher dimensional problems by drastically reducing the search space and scaling according to the length scales. In addition, as suggested in Subsection 1.2.3, imposing a minimal value for the length scales or using automatic relevance determination might help in higher dimensional spaces. Consequently, we propose to tackle with the previously investigated algorithms another optimal management problem similar to the one of Section 3.4.2, but involving 30 dimensions.

4.4.1 PHES Optimal Management Problem

The integration of larger shares of renewable energy sources into global production is crucial to limit global warming. However, the intermittency of solar and wind power sources poses significant challenges to grid operations and can disrupt the grid stability. Therefore, disposing of storage facilities is crucial to maintain the balance between power generation and consumption. As already introduced in Section 3.4.2, PHES plants are valuable assets for this purpose.

Context and Objectives

Efficient tools for maximizing the profitability of PHES plants are needed to facilitate the integration of renewable energy sources into the global consumption. Consequently, a simulator is developed in Favaro *et al.* [155] to compute the profit y of the operator according to a decision $\boldsymbol{x}$. The simulator is referred to as $f : \mathbb{R}^{30} \to \mathbb{R}; f(\boldsymbol{x}) \mapsto y$ in the following. In this work, the simulator is assumed already known and black-box, but the details about its construction are available in [155].

Most methods tackling this problem involve its reformulation, such as with a MILP formulation [184], so that the problem can be solved with linear programming solvers. This type of approach is able to guarantee the optimality - or at least an estimate of the error - of the obtained solution and operates fast, but concedes some approximations that possibly result in unfeasible decisions. Another alternative is to directly optimize the simulator, but without any guarantee of optimality. Usual approaches in this case include various metaheuristics such as EA or PSO [185]. In Favaro *et al.* [155], the authors proposed a neural network-assisted MILP formulation of the problem which offers the desirable optimality guarantees and avoids divergences. The method shows to outperform the state of the art on several scenarios of the considered PHES plant. Despite these advances, the solution of the MILP algorithm may not always be feasible due to the inherent approximations of the model.

In the present work, we challenge the MILP formulation with the previously studied SBO algorithms. Similarly to the mentioned EAs and PSO algorithms, they are directly applied to the simulator, which operates without approximation, and thus captures the full complexity of the process and ensures the feasibility of the final decision. Nevertheless, the main drawback of this approach is that we lose the optimality guarantees provided by the Mixed-Integer Linear Programming (MILP) formulation.

The first step of this work aims to assess if the SBO algorithms applied to the simulator are able to find a better solution in a limited time than the MILP approach. The cost of a simulation is less than 0.2 second, therefore we do not expect BOAs to compete with SAEAs or with the MILP formulation. However, the dimension is not prohibitively high to apply BO and thus it is still interesting to study the differences between the BOAs with the additional prism of the dimension. In a second time, a multi-fidelity approach is proposed to combine the guarantees, tractability, and robustness offered by the MILP solver with the versatility of surrogate-based methods. In this context, the BOAs might be successful in refining the solution thanks to their sampling efficiency. The initial sample of the SBO includes decisions found by the MILP optimization so that SBO algorithms are used only to refine the decision vector.

4.4.2 Experimental Protocol

The simulator is based on scenarios representing different states of the energy market, mainly characterized by the prices for each time interval. Those data are extracted from real data sets and account for different days and conditions the PHES operator may face. The ability to consistently find a good decision for any scenario is essential to ensure the profitability of the storage units. Therefore, we investigate 11 different operation days.

The methods taking part in this study are the same as in the benchmark analysis of Section 4.1. Considering the short execution time of the simulator, we investigate another version of TuRBO (which is expected to perform best among BOAs), where the surrogate model fitting is faster. Up to now, learning the hyper-parameters of the GP model was done identically for each BOA. The only difference was the size of the data set for ℓBSP-EGO. However, as mentioned earlier the routine to fit the model can vary, resulting in different accuracies. The `GPytorch` library offers the possibility to tune the learning phase by granting a budget to the likelihood maximization step. The impact of the accuracy of the model should be investigated in a separate study. However, in this context, reducing the learning time seems relevant.

The experiments are led on an Intel Xeon Gold 5220 CPU from the Grid5000 infrastructure for distributed computing [5]. The batch size is fixed to 8 so that the protocol is transferable to most recent single-processor machines. The initial design of experiment includes 296 ($\approx 10d$) points and 4 initial conditions are considered. The first one is generated through Latin hypercube sampling only, while the three others also include solutions from the MILP resolution. This strategy is referred to as the warm-start optimization and the computational budget is divided between the MILP resolution and the Bayesian Optimization (BO). The total optimization time is fixed to 20 minutes, including the initial sampling. The time dedicated to the warm-start is $t_{warm} \in \{1, 2, 5\}$ minutes. Each of the 11 scenarios is executed for the 4 warm-start conditions and repeated 10 times. The best profit is monitored after $t \in \{2, 3, 5, 10, 15, 20\}$ minutes.

4.4.3 Experimental Results

As a preliminary task, 8 neural networks are created to be used in the MILP formulation. They approximate the turbine operation curves that intervene in the simulator. It is an external task that is not included in the 20-minute budget as it can be done only once for the installation.

Consequently, the differences in the approximation cause differences in the outcome of the MILP. The maximum resolution time is fixed to 20 minutes, and those 8 solutions constitute our baseline for the following comparison.

Optimization without warm-start

First, we compare the methods between them depending on the time budget, without warm-start. Table 4.4 shows the Friedman's rank computed with the 10 repetitions of the 11 scenarios. A lower rank indicates a better performance compared to the other approaches. For short times, BOAs achieve better outcomes than EAs. Among them, the Fast-TuRBO algorithm presents the best results but all of them are outperformed by SAGA as soon as the time reaches 300 seconds.

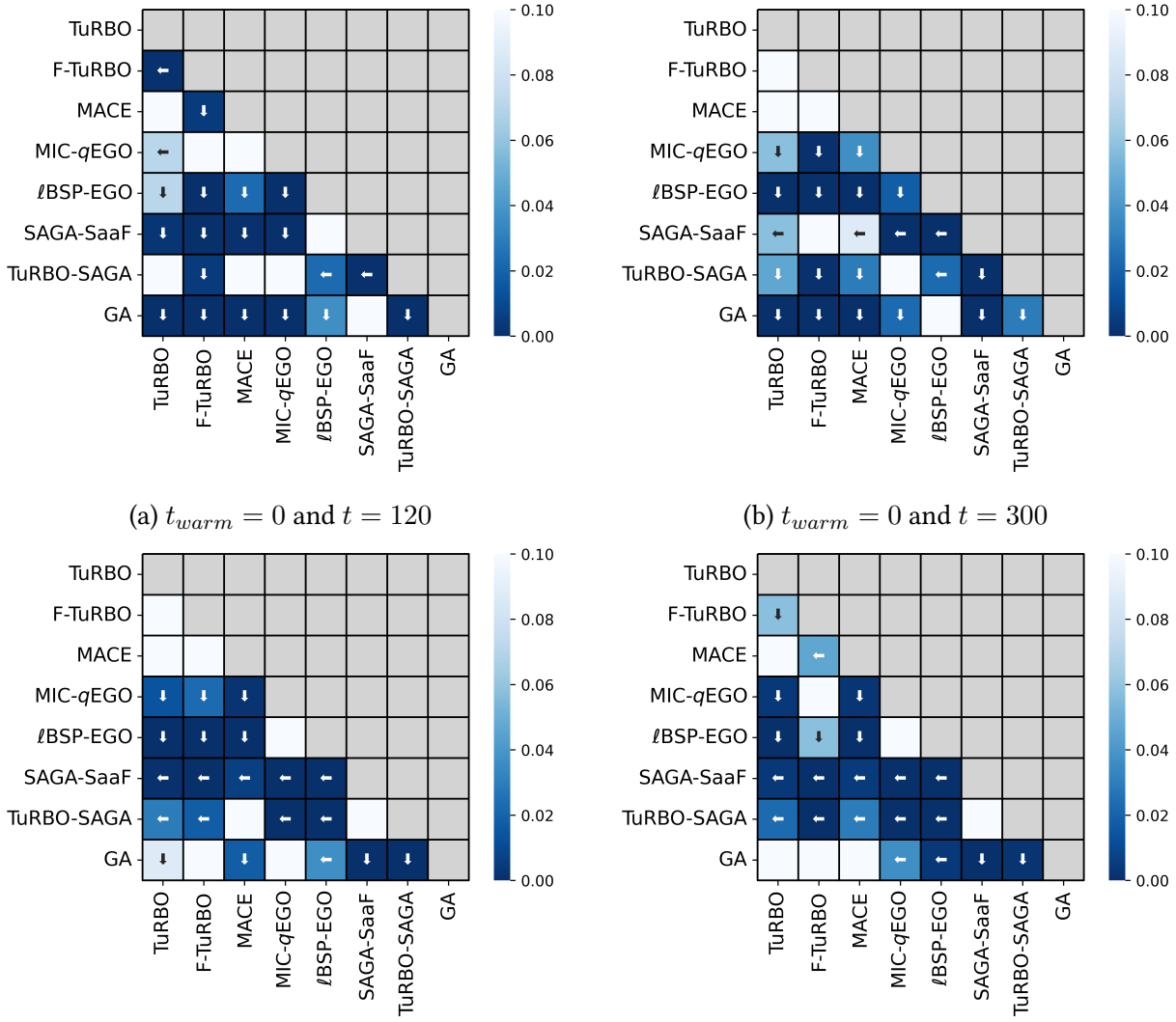
t_{warm}	t	TuRBO	Fast-TuRBO	MACE	MIC-qEGO	ℓBSP-EGO	SAGA-SaaF	TuRBO-SAGA	GA
0	120	4.18	1.00	3.73	2.45	5.91	7.09	3.73	7.91
	300	3.45	2.00	3.27	5.27	7.55	1.64	5.36	7.45
	600	4.27	4.45	3.64	6.64	7.91	1.00	2.18	5.91
	900	4.18	5.55	3.91	6.91	7.64	1.00	2.00	4.82
	1200	4.00	5.82	3.91	6.82	7.64	1.18	1.82	4.82

Table 4.4: Friedman ranks for the PHES management problem, without warm-start

Table 4.5 presents the number of simulations performed for each time stamp, excluding the initial sample. According to Table 4.5, BOAs remain very sample-efficient as they have the best outcomes with much fewer simulations than EAs. The faster version of TuRBO appears to outperform the others, most likely because of the higher number of simulations that Fast-TuRBO is able to perform compared to TuRBO. However, given enough time ($t \geq 600$s), TuRBO outperforms its faster version despite a much lower number of simulations. The performances of TuRBO and MACE are close in terms of Friedman's rank despite a slightly lower number of simulations achieved by MACE. In this context, the cooperative acquisition strategy of MACE seems to be a better choice than the competitive counterpart represented by MIC-qEGO. Unsurprisingly, ℓBSP-EGO does not perform well in this context. The binary decomposition is irrelevant for such dimension, and the low batch size does now allow the algorithm to fully exploit what makes it efficient.

Without warm-start, according to Table 4.4, Fast-TuRBO should be preferred if the time budget is lower than 300 seconds, and SAGA-SaaF otherwise. The latter observations are confirmed by the *post-hoc* pairwise Friedman's test [179] displayed in Figure 4.4 for each time-stamp.

Figure 4.5 presents the best-expected profit according to the number of simulations for day 6. The results observed for day 6 are rather representative of the observations done for any other



(a) $t_{warm} = 0$ and $t = 120$

(b) $t_{warm} = 0$ and $t = 300$

(c) $t_{warm} = 0$ and $t = 600$

(d) $t_{warm} = 0$ and $t = 1200$

Figure 4.4: p-values of the pairwise comparison *post-hoc* Friedman's rank test for the PHES problem. Low values indicates statistically significant differences, and are highlighted by the color scale. In case of p-value < 0.1, an arrow indicates the direction of the algorithm outperforming the other.

t_{warm}	Method	120	300	600	1200
0	TuRBO	46.76	160.22	277.89	434.98
	GA	256.29	1272.15	2938.18	6249.89
	TuRBO-SAGA	47.05	210.84	1996.22	5018.47
	SAGA-SaaF	235.93	1138.04	2643.78	5592.73
	MACE	31.56	126.62	233.31	369.75
	Fast-TuRBO	125.09	403.13	669.09	979.13
	MIC-qEGO	31.56	129.24	231.20	365.09
	ℓBSP-EGO	76.87	338.18	708.80	1318.04

Table 4.5: Average number of simulations on the PHES management problem, excluding the initial sample.

day. The black dotted line shows the best outcome from the 8 initial MILP results. Only the SAGA and TuRBO-SAGA algorithms consistently (10 times out of 11 days) achieve better outcomes than the best MILP outcome. No significant difference compared to day 6 is observed on the other investigated scenarios and similar observations can be made for any day, except day 1, where no algorithm gives better results than the MILP.

Finally, we compare the expected profit of the best-performing methods to the best result of the MILP formulation. For each day, we report the minimum, average, and maximum profit and scale them according to the best MILP result. Last, to summarize the data, the average is computed over the 11 days and the results are presented in Table 4.6. We can see that SAGA approaches are the only ones consistently outperforming the MILP formulation, but only with at least 600 seconds of execution. Considering the 20-minute budget, SAGA-SaaF offers an average gain of 18%. Only the day-1 scenario shows a loss between 16% (worst case) and 12%.

Optimization with warm-start

We have demonstrated the interest of SBO algorithms and especially of the SAEAs in this context. However, we have no indication on the optimality of the solution, and the obtained results might be, on rare cases, less profitable than the MILP ones. Another strategy that might be interesting to bypass these issues is to initialize the data set with the preliminary results from the MILP formulation. Doing so offers a stronger guarantee of profitability, and possibly much better outcome if the time-budget is sufficient. The MILP solver is stopped after t_{warm} seconds and the outcome is used in the initial data set.

Table 4.7 presents the Friedman's rank of the methods according to the time dedicated to the warm start, for each time-stamp. With the warm-start, both TuRBO versions perform well at any time, with a small advantage for the faster alternative. Considering small additional times after the warm-start ($t_{warm} + t_{add}$ where $t_{add} \leq 300$), Fast-TuRBO offers the best outcomes. If the additional time is 600 seconds at least, SAGA-SaaF is often the best option.

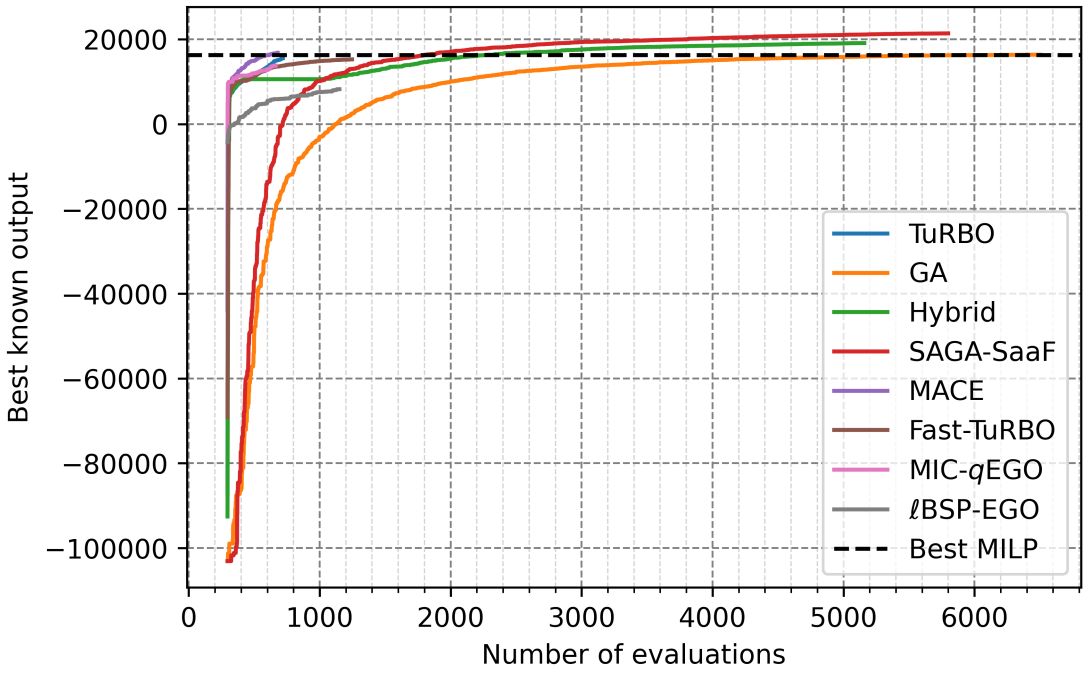


Figure 4.5: Best expected profit according to the number of simulations for day 6 of the PHES management problem.

Method	Value	120	300	600	900	1200
TuRBO	Min	-0.14	0.55	0.68	0.72	0.77
	Avg	0.23	0.67	0.78	0.86	0.91
	Max	0.49	0.81	0.90	0.98	1.05
TuRBO-SAGA	Min	-0.08	0.47	0.87	1.03	1.08
	Avg	0.22	0.60	0.98	1.10	1.15
	Max	0.46	0.71	1.09	1.19	1.24
SAGA-SaaF	Min	-17.88	0.57	0.98	1.06	1.09
	Avg	-7.80	0.78	1.08	1.14	1.18
	Max	-1.01	0.93	1.17	1.23	1.27
MACE	Min	-0.16	0.50	0.65	0.73	0.77
	Avg	0.17	0.70	0.84	0.89	0.91
	Max	0.41	0.85	0.95	0.98	1.00
Fast-TuRBO	Min	0.47	0.58	0.59	0.59	0.59
	Avg	0.59	0.75	0.78	0.79	0.79
	Max	0.70	0.89	0.94	0.95	0.95

Table 4.6: Gain of the SBO algorithms compared to the MILP formulation for the PHES management problem, **without warm-start**

Figure 4.6a shows that a large warm-start time is not mandatory to get good preliminary outcomes. Very few changes are observed in the MILP outcomes after 120 seconds. Spending more time in the MILP resolution only increases the guarantee in the optimality of the final outcome. However, limiting t_{warm} to 120 seconds offers more time to the SBO algorithms which generally manage to improve the expected profit, given a sufficient time budget. Consequently, fixing t_{warm} to 120 seconds seems a fair compromise between optimality guarantee and efficiency. For stronger guaranties, one can decide to run the MILP a bit longer to achieve to desired guaranty before resorting to SBO. Table 4.8 shows the average gain of the best SBO methods compared to the best MILP outcome, given a 120-second warm-start. We can see that Fast-TuRBO shows improvement after a small additional time, but if the total budget exceeds 600 seconds, SAGA-based algorithms should be preferred.

Using the warm-start procedure obviously induces a bias in the search and incentivizes the sampling close to the initial MILP results. Regarding only SAGA-SaaF in Table 4.8 and Table 4.5 we can see that the warm-start strategy slightly delays the higher gains. However, using the full budget, the two approaches (with and without a warm-start) offer very similar results, but with a higher guarantee for the warm-start one.

t_{warm}	t	TuRBO	Fast-TuRBO	MACE	MIC-qEGO	$\emptyset$BSP-EGO	SAGA-SaaF32	TuRBO-SAGA	GA
60	300	2.73	1.91	3.18	7.18	6.91	5.45	4.82	3.82
	600	4.00	2.09	4.73	7.36	6.91	1.82	3.64	5.45
	900	3.73	2.91	5.55	7.36	7.09	1.82	2.64	4.91
	1200	3.45	3.27	5.55	7.36	7.18	1.91	2.55	4.73
120	300	2.36	3.27	3.09	7.18	7.00	5.36	4.36	3.36
	600	3.45	1.91	4.82	7.36	6.73	2.36	4.45	4.91
	900	3.27	2.55	5.55	7.36	7.09	2.09	3.18	4.91
	1200	3.27	3.00	5.64	7.45	7.00	2.18	2.73	4.73
300	600	3.27	1.45	3.36	7.09	7.55	3.73	5.27	4.27
	900	4.09	2.18	4.91	7.27	7.18	1.91	3.27	5.18
	1200	3.55	2.73	5.27	7.27	7.27	1.91	2.91	5.09

Table 4.7: Friedman ranks for the PHES management problem, **with warm-start**

Finally, we compare the MILP only strategy with the multi-fidelity approach defined by a 120 seconds warm-start followed by the SAGA-SaaF algorithm. The average profit according to the time spent is displayed on Figure 4.6. The average value is represented in solid lines while the dotted lines shows the best and worst outcomes. Figure 4.6b displays the results of the best observed strategy, that is SAGA-SaaF after a 120-second warm-start. This approach allows to increase the profit of the operator by 18% in average.

Method	Value	120	300	600	900	1200
TuRBO	Min	1.00	1.00	1.00	1.02	1.06
	Avg	1.00	1.00	1.02	1.06	1.10
	Max	1.00	1.00	1.05	1.11	1.14
TuRBO-SAGA	Min	1.00	1.00	1.00	1.05	1.09
	Avg	1.00	1.00	1.01	1.11	1.15
	Max	1.00	1.00	1.05	1.20	1.24
SAGA-SaaF	Min	1.00	1.00	1.01	1.06	1.08
	Avg	1.00	1.00	1.06	1.13	1.17
	Max	1.00	1.00	1.14	1.22	1.26
Fast-TuRBO	Min	1.00	1.01	1.04	1.05	1.05
	Avg	1.00	1.02	1.08	1.10	1.10
	Max	1.00	1.04	1.13	1.14	1.15

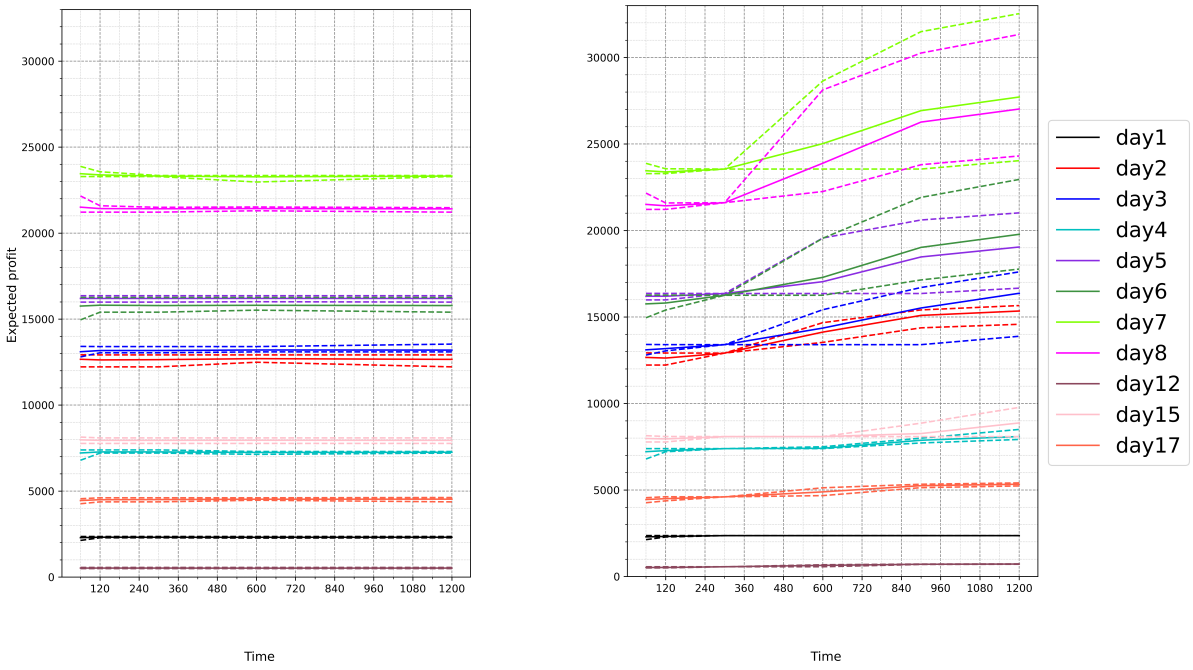
Table 4.8: Gain of the multi-fidelity strategy (SBO after a **120s MILP warm-start**) for the PHES management problem

4.4.4 Conclusion and Discussion

We applied different SBO algorithms to the PHES optimal management problem and proposed a suitable strategy for increasing the expected profit of the operator. The best-identified strategy consists in solving (in parallel) several MILP approximations with a maximum operation time of 120 seconds, and then to apply a SAEA to the simulator, initialized with the results of the warm-start. This strategy results in an average gain of 18% of the expected profit on 11 scenarios employed in this study.

SAEAs are the only methods that present consistent improvements over the MILP formulation. The dimension of the problem makes BOAs less efficient, but mostly the small time cost of the simulator strongly favors fast operating algorithms such as SAEAs. This is also observed with the Fast-TuRBO algorithm, where a fast model learning allows many more simulations in the dedicated time. However, even if it results in a fast improvement at the beginning, at some point it is preferable to spend more time learning the hyper-parameters to make better choices in the candidate selection.

Finally, accounting only for the number of performed simulations, BOAs manage to improve the outcome with only few evaluations. Both multi-infill and trust region methods significantly improve the initial sampling and often achieve results equivalent to or better than the GA, with a much smaller number of simulations. This indicates that for more time-consuming problems, BOAs could be a good choice even in dimension 30.



(a) Best outcome of the MILP formulation according to the optimization time

(b) Best outcome of the proposed multi-fidelity strategy according to time: $t_{warm} = 120$ followed by SAGA-SaaF

Figure 4.6: Comparison between the baseline approach (left) and the proposed one (right) for the PHES problem. Solid lines indicate the average outcomes while dotted-lines represent their respective minimum and maximum.

Conclusions and Perspectives

Conclusions

Simulation, machine learning, optimization and parallel computing are the ingredients for solving computationally expensive black-box engineering problems. These latter refer to optimization problems in which the objective function is CPU time-intensive, is derivative free and has an unknown analytical expression. In this thesis, we investigate Parallel Bayesian Optimization (PBO) combining optimization with surrogate-based machine learning and parallel computing, resulting in a powerful approach to deal with these computationally expensive problems. BO Algorithms (BOAs) are Surrogate-Driven Optimization (SDO) methods, where the next sampling point is queried by the optimization of an Acquisition Function (AF). Sequential BOAs such as Efficient Global Optimization (EGO) are particularly efficient and achieve excellent performances with tight computational budgets. Their ability to identify good solutions with only few simulations makes BO one of the most promising approach for tackling expensive problems. Considering a fixed time budget as part of the operational constraints, driving BOAs parallel would allow the algorithms to perform more simulations and possibly achieve better outcomes. However, as further developed in the following, the parallelisation of BOAs raises significant challenges.

As a first contribution, we analysed the performances and limitations of state-of-the-art algorithms, including EGO and one parallel version named qEGO, applied to challenging real-world problems. The sequential EGO algorithm is extremely sample-efficient and particularly adapted for very expensive simulations where the parallel resources can be exploited inside the simulation. When parallel calls to the simulator are possible, qEGO can be used to perform more simulations per cycle. However, we observed a much lower performance of qEGO compared to its sequential counterpart for an equivalent number of simulations. Actually, many BOAs suffer from a low batch effectiveness (*i.e.* batch of poor quality). In addition, we observed a poor scalability regarding the number of simulations mainly caused by the computational burden of the surrogate model fitting and the Acquisition Process (AP). The two aspects result in a waste of the computational budget as increasing the resources and consequently the number of simulations does not always improve the final outcome. In order to emphasize the computational burden of BOA, in particular regarding the AP time, it is decided to define the budget as a fixed time instead of a number of simulations or cycles (*i.e.*, q simultaneous simulations). When compared with evolutionary alternatives, the sample-efficiency of BOAs such as qEGO is confirmed. Indeed, Evolutionary Algorithm and their surrogate-assisted counterparts usually require much more objective function evaluations to compete with BOAs. However, the evolutionary operators responsible for the generation of candidates are more time-efficient than the AP in BOAs. Consequently, with a moderately time-consuming simulator, the AP time might not be negligible anymore compared to the simulation time and Surrogate-Assisted Evolutionary Algorithms (SAEAs) might be preferred.

An efficient PBO algorithm requires a well-designed Acquisition Process (AP). The computational cost of the AP must remain reasonable compared to the simulation time, and the batch

effectiveness must be improved to preserve the effectiveness of the sequential candidate selection in a parallel framework. We proposed a new AP using spatial decomposition to introduce parallelism into the candidate selection and to better control the diversification/intensification trade-off, improving the batch effectiveness. We derived two algorithms from this approach, namely Binary Space Partitioning Efficient Global Optimization (BSP-EGO) and Local BSP-EGO (ℓBSP-EGO), and studied the impact of spatial decomposition in BOAs. BSP-EGO preserves the global search by fitting the Gaussian Process (GP) surrogate model over the whole available data, while ℓBSP-EGO selects a subset of the data to fit a local surrogate model inside a sub-region. The two algorithms were challenged with state-of-the-art BOAs using multiple single-point Acquisition Functions (AFs), multi-point AFs, or APs inside a trust region. We demonstrated the efficiency of the decomposition scheme to reduce the acquisition time and improve the batch effectiveness as BSP-EGO performs better than qEGO and KB-qEGO on average. However, we also observed that the global model fitting time becomes prohibitively large when the data set size increases. This problem occurs for any BOA, except for ℓBSP-EGO thanks to the local surrogate modeling. ℓBSP-EGO shows a good scalability as it is able to perform a lot more objective function evaluations than its contestants in the fixed time budget. In addition, the gain in terms of number of simulations results in an improved final outcome which indicates a good batch effectiveness. The two aspects made ℓBSP-EGO the best-performing algorithm of the benchmark study. Another perspective regarding those results is offered when adding a fictitious time cost to the benchmark function evaluation. Indeed, the excellent average performance of ℓBSP-EGO is dependent on the number of additional simulations compared to other algorithms. Among BOAs, it achieves better performances than most alternatives except from TuRBO which generally performs best for budgets below one thousand of simulations (for any batch size).

The objectives of this thesis also included providing guidelines for practitioners to adequately select an algorithm for a given problem. For this purpose, we investigated the performance of several BOAs that operate the batch acquisition in different ways, but also completely different alternatives such as SAEAs. The proposed protocols account for the computational cost of the simulator, the available computing units, and the total time budget. From the extensive benchmark analysis and the diverse real-world problems, we were able to derive several recommendations regarding the choice of the most suitable algorithm. On the one hand, focusing only on the BOAs, we observed that relying on multiple AFs is generally a good strategy, either competitively or cooperatively. The competitive way consists in choosing several AFs that act separately, it is quite effective for low computational budgets but does not scale well with the batch size as the AP remains time-consuming. Concerning the cooperative way, it necessitates multi-objective optimization to find the best trade-offs between the AFs. The main advantage of this strategy is that it is easy to sample from the Pareto front to get an arbitrarily large number of candidates of good quality. Consequently, the scalability is better regarding the batch size. Another profitable strategy is to progressively focus on sub-regions to force intensification. The BSP-based algorithms evolve at each cycle to intensify the search in the most promising regions. One can also rely on trust regions such as in the TuRBO algorithm. Exploration is favored at the beginning of the search, and as the budget fades, exploitation is promoted. This approach is particularly efficient when the dimension increases (*i.e.* 10 or more). However, the binary decomposition is not drastic enough to compensate for the curse of dimensionality. In a 30-dimensional space, TuRBO seems to be more effective as it acts on all dimensions at once.

On the other hand, considering Surrogate-Assisted Optimization (SAO), an alternative to the SDO, allowed us to extract more general insights regarding the choice of the algorithm. Indeed, it is observed that SAEAs considerably reduce the number of necessary simulations compared to surrogate-free Evolutionary Algorithms (EAs). Especially, using evolutionary operators to generate candidates and filter them using a GP model (*e.g.* as in SAGA-SaaF) provides excellent results with a computing budget of few hundred evaluations. SAEAs consistently outperform any BOAs as soon as the budget exceeds about one thousand of exact evaluations. The identified threshold between BOAs and SAEAs drove us to propose a hybrid algorithm using a BOA at the early stages of the search to benefit from its sample efficiency and switch to a more time-efficient SAEA as soon as the BOA becomes too time-consuming. This algorithm is experimented using TuRBO and SAGA-SaaF and displays excellent any-time performance, offering a suitable algorithm for many situations.

From the application point of view, we addressed five real-world engineering problems using well-chosen SBO algorithms. The introduction of SBO into the different applications resulted in improved outcomes compared to the approaches used so far. The inverse identification problem in mechanical engineering has been approached using the EGO algorithm. The associated simulator is very expensive and software-dependent, which prevents larger parallelization, consequently sequential BO is particularly suited. The optimized parameters provide a better accuracy than the ones from the reference work. PBO is successfully applied to the tuberculosis transmission control problem, where the simulator evaluates the impact of the budget allocation (*i.e.* the design vector) on the expected prevalence of the disease in the Philippines. The simulation of the optimized decision shows a positive impact of the treatment allocation on the population. In this application, the simulator is only moderately time-consuming as the budget can be large enough to achieve many simulations. In light of the threshold identified in Chapter 4, SAEAs could be a better choice with an increased budget. We have also demonstrated that PBO can be leveraged in time-constrained optimization problems such as the ones involved in the energy market. Three applications related to the energy market are tackled in this thesis. They involve from 3 to 30 design variables and various simulation costs and all concern the management of the resources of a plant operator seeking for maximum profit. We proposed two optimization approaches depending on the expensiveness and the dimension of the simulator. Namely, BOAs are used for simulators with a moderate time-cost and dimensions lesser than 12, while SAEAs yield better profits in the case of cheaper simulators and higher dimensions.

In a nutshell

The main contributions of this thesis are summarized as follows:

- We demonstrated the usefulness of BOAs for solving efficiently real-world problems and in domains where they are not often considered.
- We redefined the concept of optimization budget by taking into account the experimental conditions (*i.e.* simulation time, computational resources, and total time).

- We designed and implemented new algorithms that introduced parallelism into the AP, which is uncommon in BO, speeding up the batch acquisition, and enhancing its promisingness.
- We packaged and made publicly available all the developed algorithms as well as the full experimental framework (all algorithms and benchmark functions) in a dedicated GitHub repository: `https://github.com/MaGbrt/pySBO`.
- We confronted BOAs with other SBO algorithms, namely SAEAs and EAs, to identify their domain of applicability in terms of number of simulations.
- We derived recommendations to select or design relevant algorithms according to the experimental conditions.

Perspectives

The previous contributions also revealed some leads for future developments. In this work, we mainly confronted the different APs of the BOAs to identify their strengths. However, we did not consider the simultaneous use of, for instance, spatial decomposition and complementary AFs. The two features seem to perform differently depending on the objective function. The combination of both could help to efficiently tackle a wider range of problems.

Despite improvements regarding the batch effectiveness, having a large batch size does not always mean better outcomes at the end. The latter suggests that concurrent BOAs acting in parallel with moderate batch sizes could be beneficial compared to a single one with a large batch size. For instance, 4 (different) algorithms running with $q = 8$ might be preferable to one with $q = 32$.

In addition, the decomposition scheme adopted in BSP-based algorithms shows its limitations when increasing the dimension of the objective function. The decomposition of the design space could be more intensive such as in the trust region methods, where all dimensions are reduced at once. For instance, each sub-region could be characterized by its best sample with a trust region built around it.

In the last section, we initiated the questioning of the way surrogate models are fitted. The TuRBO algorithm is experimented with a fast routine to fit the hyper-parameters of the model and noticeable differences are observed. The faster version tends to give better outcomes at early stages, but for late improvements, the more precise version shows better final outcomes. This should be further investigated to evaluate the impact of the model fitting on the optimization.

Hybrid methods combining BOAs and SAEAs showed promising results. The hybridization is based on the identified threshold and the first population is initialized with the last simulations. These two aspects could be further investigated and, for instance, dynamic criterion that triggers either the BOA or the SAEA could be considered for the hybridization.

The length-scale parameters of the GP model are sometimes used to identify the most impacting design variables, such as in TuRBO which uses them to scale the trust region. This strategy appears to be quite efficient and somehow reminds landscape analysis methods. The

latter is rarely considered since it requires many function evaluations to be pertinent. However, with the increased number of evaluations of recent BOAs and considering the performances of algorithms such as TuRBO, information from landscape analysis could be integrated into the optimization process to dynamically and adaptively select the best strategy.

All along the manuscript, we considered synchronous parallelization. However, with parallel APs and restricted time budgets, the development of asynchronous parallel algorithms could help to exploit the full potential of parallel processors. Indeed, in case of irregular workloads (*e.g.*, simulations) in a synchronous framework, most worker processes are idling at some point waiting for the last worker to finish.

Likewise, only single-objective problems, or scalarized multi-objective ones, are tackled. In the multi-objective context, the proposed approaches should be revisited using uncommonly a Pareto-based approach taking into account the correlations between the objectives as in [186].

Bibliography

1. Jones, D. R., Schonlau, M. & Welch, W. J. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **13,** 455–492. ISSN: 1573-2916. https://doi.org/10.1023/A:1008306431147 (1998).
2. Kushner, H. J. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Fluids Engineering* **86,** 97–106. ISSN: 0098-2202. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/86/1/97/5763745/97_1.pdf. https://doi.org/10.1115/1.3653121 (1964).
3. Binois, M. & Wycoff, N. A Survey on High-dimensional Gaussian Process Modeling with Application to Bayesian Optimization. *ACM Trans. Evol. Learn. Optim.* **2.** https://doi.org/10.1145/3545611 (2022).
4. Rasmussen, C. E. & Williams, C. K. I. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). ISBN: 026218253X (The MIT Press, 2005).
5. Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., *et al.* Adding Virtualization Capabilities to the Grid'5000 Testbed. in *Cloud Computing and Services Science* (eds Ivanov, I. I., van Sinderen, M., Leymann, F. & Shan, T.) 3–20 (Springer International Publishing, 2013). ISBN: 978-3-319-04518-4.
6. Wang, X., Jin, Y., Schmitt, S. & Olhofer, M. Recent Advances in Bayesian Optimization. *ACM Comput. Surv.* **55.** ISSN: 0360-0300. https://doi.org/10.1145/3582078 (2023).
7. Frazier, P. A Tutorial on Bayesian Optimization (2018).
8. Močkus, J. On bayesian methods for seeking the extremum. in *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974* (ed Marchuk, G. I.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 1975), 400–404. ISBN: 978-3-540-37497-8.
9. Srinivas, N., Krause, A., Kakade, S. & Seeger, M. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. in (2010), 1015–1022.
10. Hennig, P. & Schuler, C. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research* **13** (2011).
11. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & de Freitas, N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE* **104,** 148–175. ISSN: 1558-2256 (2016).
12. Binois, M., Collier, N. & Ozik, J. A portfolio approach to massively parallel Bayesian optimization. working paper or preprint. 2023. https://hal.inria.fr/hal-03383097.

13. Chen, J., Luo, F., Li, G. & Wang, Z. Batch Bayesian optimization with adaptive batch acquisition functions via multi-objective optimization. *Swarm and Evolutionary Computation* **79,** 101293. ISSN: 2210-6502. https://www.sciencedirect.com/science/article/pii/S2210650223000664 (2023).

14. Palma, A. D., Mendler-Dünner, C., Parnell, T., Anghel, A. & Pozidis, H. Sampling Acquisition Functions for Batch Bayesian Optimization. 2019. arXiv: 1903.09434 [cs.LG].

15. Adachi, M., Hayakawa, S., Hamid, S., Jørgensen, M., Oberhauser, H. & Osborne, M. A. SOBER: Highly Parallel Bayesian Optimization and Bayesian Quadrature over Discrete and Mixed Spaces. 2023. arXiv: 2301.11832 [cs.LG].

16. Ginsbourger, D., Le Riche, R. & Carraro, L. A Multi-points Criterion for Deterministic Parallel Global Optimization based on Kriging (2008).

17. Ginsbourger, D., Le Riche, R. & Carraro, L. Kriging Is Well-Suited to Parallelize Optimization. in (2010).

18. Eriksson, D., Pearce, M., Gardner, J., Turner, R. D. & Poloczek, M. Scalable Global Optimization via Local Bayesian Optimization. in *Advances in Neural Information Processing Systems* **32** (Curran Associates, Inc., 2019). https://proceedings.neurips.cc/paper_files/paper/2019/file/6c990b7aca7bc7058f5e98ea909e924b-Paper.pdf.

19. Briffoteaux, G. Algorithmes parallèles et basés sur méta-modèles pour la résolution de problèmes d'optimisation coûteux. PhD thesis (2022).

20. Briffoteaux, G., Gobert, M., Ragonnet, R., Gmys, J., Mezmaz, M., Melab, N. & Tuyttens, D. Parallel surrogate-assisted optimization: Batched Bayesian Neural Network-assisted GA versus q-EGO. *Swarm and Evolutionary Computation* **57,** 100717. ISSN: 2210-6502. http://www.sciencedirect.com/science/article/pii/S2210650220303709 (2020).

21. Kugalur Palanisamy, N., Rivière Lorphèvre, E., Gobert, M., Briffoteaux, G., Tuyttens, D., Arrazola, P.-J. & Ducobu, F. Identification of the Parameter Values of the Constitutive and Friction Models in Machining Using EGO Algorithm: Application to Ti6Al4V. *Metals* **12.** ISSN: 2075-4701. https://www.mdpi.com/2075-4701/12/6/976 (2022).

22. Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Batch Acquisition for Parallel Bayesian Optimization; Application to Hydro-Energy Storage Systems Scheduling. *Algorithms* **15.** ISSN: 1999-4893. https://www.mdpi.com/1999-4893/15/12/446 (2022).

23. Ducobu, F., Kugalur Palanisamy, N., Briffoteaux, G., Gobert, M., Tuyttens, D., Arrazola Arriola, P.-J. & Rivière-Lorphèvre, E. Identification of the Constitutive and Friction Models Parameters via a Multi-Objective Surrogate-Assisted Algorithm for the Modeling of Machining - Application to ALE orthogonal cutting of Ti6Al4V. *Journal of Manufacturing Science and Engineering,* 1–54. ISSN: 1087-1357. eprint: https://asmedigitalcollection.asme.org/manufacturingscience/article-

pdf/doi/10.1115/1.4065223/7324066/manu-23-1749.pdf. https://doi.org/10.1115/1.4065223 (2024).

24. Gobert, M., Briffoteaux, G., Gmys, J., Melab, N. & Tuyttens, D. Observations in Applying Bayesian versus Evolutionary approaches and their Hybrids in Parallel Time-constrained Optimization, ***Currently under review*** *in Engineering Applications of Artificial Intelligence.*

25. Favaro, P., Gobert, M. & Toubeau, J.-F. Multi-fidelity Optimization for Pumped Hydro Energy Storage Participating in Energy and Reserve Markets, ***Currently under review*** *in Applied Energy.*

26. Gobert, M., Gmys, J., Toubeau, J.-F., Vallée, F., Melab, N. & Tuyttens, D. Surrogate-Assisted Optimization for Multi-stage Optimal Scheduling of Virtual Power Plants. in *2019 International Conference on High Performance Computing Simulation (HPCS)* (2019), 113–120.

27. Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Adaptive Space Partitioning for Parallel Bayesian Optimization. in *HPCS 2020 - The 18th International Conference on High Performance Computing Simulation* (Barcelona / Virtual, Spain, 2021). https://hal.inria.fr/hal-03121209.

28. Gobert, M., Gmys, J., Toubeau, J.-F., Melab, N., Tuyttens, D. & Vallée, F. Parallel Bayesian Optimization for Optimal Scheduling of Underground Pumped Hydro-Energy Storage Systems. in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2022), 790–797.

29. Filipič, B., Depolli, M., Zupančič, J., Gmys, J., Gobert, M., Melab, N. & Tuyttens, D. ECG Simulator Tuning: A Parallel Multiobjective Optimization Approach. in *Proceedings OLA'2018 International Workshop on Optimization and Learning: Challenges and Applications* (2018), 25–28.

30. Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Towards Adaptive Space Partitioning for Large-scale Parallel Bayesian Optimization. in *OLA'2020 - International Conference on Optimization and Learning* (Cadix, Spain, 2020). https://hal.archives-ouvertes.fr/hal-02898960.

31. Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Space Partitioning with multiple models for Parallel Bayesian Optimization. in *OLA 2021 - Optimization and Learning Algorithm* (Sicilia / Virtual, Italy, 2021). https://hal.archives-ouvertes.fr/hal-03324642.

32. Talbi, E.-G. Metaheuristics: from design to implementation, 566. https://hal.inria.fr/hal-00750681 (Wiley, 2009).

33. Diaz-Manriquez, A., Toscano Pulido, G., Barron-Zambrano, J. & Tello, E. A Review of Surrogate Assisted Multiobjective Evolutionary Algorithms. *Computational Intelligence and Neuroscience* **2016,** 1–14 (2016).

34. Jin, Y. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* **1,** 61–70. ISSN: 2210-6502. https://www.sciencedirect.com/science/article/pii/S2210650211000198 (2011).

35. Chauvet, P. Aide-Mémoire de Géostatistique Linéaire (1993-2006).

36. Krige, D. G. A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa* **52,** 119–139 (1951).

37. Matérn, B. Spatial variation: Stochastic models and their application to some problems in forest surveys and other sampling investigations. 1960.

38. Gandin, L. S. Objective analysis of meteorological fields. *Quarterly Journal of the Royal Meteorological Society* **92,** 447–447. eprint: https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.49709239320. https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49709239320 (1966).

39. Matheron, G. Principles of geostatistics. Econ Geol (Lancaster). *Economic Geology* (1963).

40. Santner, T., Williams, B. & Notz, W. The Design and Analysis Computer Experiments. ISBN: 978-1-4419-2992-1 (2003).

41. Durrande, N. Étude de classes de noyaux adaptées à la simplification et à l'interprétation des modèles d'approximation. Une approche fonctionnelle et probabiliste. Thèse de doctorat dirigée par Carraro, Laurent Mathématiques Saint-Etienne, EMSE 2011. PhD thesis (2011). http://www.theses.fr/2011EMSE0631/document.

42. Durrande, N., Ginsbourger, D. & Roustant, O. Additive covariance kernels for high-dimensional Gaussian process modeling. *Annales de la Faculté de Sciences de Toulouse* **Tome 21.** http://afst.cedram.org/afst-bin/fitem?id=AFST_2012_6_21_3_481_0, p. 481–499. https://hal.archives-ouvertes.fr/hal-00644934 (2012).

43. Duvenaud, D. Automatic model construction with Gaussian processes. https://www.repository.cam.ac.uk/handle/1810/247281 (2014).

44. Duvenaud, D. K., Nickisch, H. & Rasmussen, C. Additive Gaussian Processes. in *Advances in Neural Information Processing Systems* (eds Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F. & Weinberger, K.) **24** (Curran Associates, Inc., 2011). https://proceedings.neurips.cc/paper_files/paper/2011/file/4c5bde74a8f110656874902f07378009-Paper.pdf.

45. Liu, D. & Nocedal, J. On the limited memory BFGS method for large scale optimization. English (US). *Mathematical Programming* **45,** 503–528. ISSN: 0025-5610 (1989).

46. Picheny, V., Wagner, T. & Ginsbourger, D. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization* **48** (2013).

47. Ababou, R., Bagtzoglou, A. & Wood, E. On the condition number of covariance matrices in kriging, estimation, and simulation of random fields. *Mathematical Geology* **26,** 99–133 (1994).

48. Roustant, O., Ginsbourger, D. & Deville, Y. DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. *Journal of Statistical Software* **51,** 1–55. https://www.jstatsoft.org/index.php/jss/article/view/v051i01 (2012).

49. Snoek, J., Swersky, K., Zemel, R. & Adams, R. Input warping for Bayesian optimization of non-stationary functions. in *International Conference on Machine Learning* (2014), 1674–1682.

50. Gramacy, R., Lee, H. & Macready, W. Parameter space exploration with Gaussian process trees (2004).

51. Gramacy, R. B. & Lee, H. K. H. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association* **103,** 1119–1130 (2008).

52. Assael, J. M., Wang, Z. & de Freitas, N. Heteroscedastic Treed Bayesian Optimisation. *CoRR* **abs/1410.7172.** arXiv: 1410.7172. http://arxiv.org/abs/1410.7172 (2014).

53. Damianou, A. & Lawrence, N. D. Deep Gaussian Processes. in *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics* (eds Carvalho, C. M. & Ravikumar, P.) **31** (PMLR, Scottsdale, Arizona, USA, 2013), 207–215. https://proceedings.mlr.press/v31/damianou13a.html.

54. Hebbal, A., Brevault, L., Balesdent, M., Talbi, E.-G. & Melab, N. Bayesian optimization using deep Gaussian processes with applications to aerospace system design. *Optimization and Engineering*. https://hal.science/hal-03046272 (2020).

55. Marmin, S. Warping and sampling approaches to non-stationary gaussian process modelling. PhD thesis (2017).

56. Noè, U. & Husmeier, D. On a New Improvement-Based Acquisition Function for Bayesian Optimization. *ArXiv* **abs/1808.06918** (2018).

57. Kandasamy, K., Krishnamurthy, A., Schneider, J. & Poczos, B. Asynchronous Parallel Bayesian Optimisation via Thompson Sampling. 2017. arXiv: 1705.09236 [stat.ML].

58. Hernández-Lobato, J. M., Hoffman, M. W. & Ghahramani, Z. Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. in *NIPS* (2014).

59. Wang, Z. & Jegelka, S. Max-value Entropy Search for Efficient Bayesian Optimization. 2018. arXiv: 1703.01968 [stat.ML].

60. Villemonteix, J., Vazquez, E. & Walter, E. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization* **44,** 509. ISSN: 1573-2916. https://doi.org/10.1007/s10898-008-9354-2 (2008).

61. Schonlau, M. Computer Experiments and Global Optimization. AAINQ22234. PhD thesis (CAN, 1997). ISBN: 0612222349.

62. Marmin, S., Chevalier, C. & Ginsbourger, D. Differentiating the Multipoint Expected Improvement for Optimal Batch Design, 37–48. https://doi.org/10.1007/978-3-319-27926-8_4 (2015).

63. Shah, A. & Ghahramani, Z. Parallel Predictive Entropy Search for Batch Global Optimization of Expensive Objective Functions. 2015. arXiv: 1511.07130 [cs.LG].

64. Chevalier, C. & Ginsbourger, D. Fast Computation of the Multi-points Expected Improvement with Applications in Batch Selection. working paper or preprint. 2012. https://hal.archives-ouvertes.fr/hal-00732512.

65. Wang, J., Clark, S. C., Liu, E. & Frazier, P. Parallel Bayesian Global Optimization of Expensive Functions. *Oper. Res.* **68,** 1850–1865. https://api.semanticscholar.org/CorpusID:12234219 (2016).

66. Jones, D. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *J. of Global Optimization* **21,** 345–383 (2001).

67. Zhan, D., Qian, J. & Cheng, Y. Balancing global and local search in parallel efficient global optimization algorithms. *Journal of Global Optimization* **67,** 873–892. ISSN: 1573-2916. https://doi.org/10.1007/s10898-016-0449-x (2017).

68. Wang, H., Bäck, T. & Emmerich, M. Multi-point Efficient Global Optimization using Niching Evolution Strategy. in (2014).

69. González, J., Dai, Z., Hennig, P. & Lawrence, N. D. Batch Bayesian Optimization via Local Penalization. 2015. arXiv: 1505.08052 [stat.ML].

70. Viana, F. A. C., Haftka, R. T. & Watson, L. T. Efficient global optimization algorithm assisted by multiple surrogate techniques. *Journal of Global Optimization* **56,** 669–689. ISSN: 1573-2916. https://doi.org/10.1007/s10898-012-9892-5 (2013).

71. Wang, Y., Han, Z.-H., Zhang, Y. & Song, W. Efficient Global Optimization using Multiple Infill Sampling Criteria and Surrogate Models. in (2018).

72. Chevalier, C., Bect, J., Ginsbourger, D., Vazquez, E., Picheny, V. & Richet, Y. Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set. *Technometrics* **56,** 455–465. https://hal.archives-ouvertes.fr/hal-00641108 (2014).

73. Liu, J., Song, W., Han, Z.-H. & Zhang, Y. Efficient aerodynamic shape optimization of transonic wings using a parallel infilling strategy and surrogate models. *Structural and Multidisciplinary Optimization* **55** (2017).

74. Hoffman, M., Brochu, E. & de Freitas, N. Portfolio allocation for Bayesian optimization. in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence* (AUAI Press, Barcelona, Spain, 2011), 327–336. ISBN: 9780974903972.

75. Lyu, W., Yang, F., Yan, C., Zhou, D. & Zeng, X. Batch Bayesian Optimization via Multi-objective Acquisition Ensemble for Automated Analog Circuit Design. in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, Stockholmsmässan, Stockholm Sweden, 2018), 3306–3314. http://proceedings.mlr.press/v80/lyu18a.html.

76. Feng, Z., Zhang, Q., Zhang, Q., Tang, Q., Yang, T. & Ma, Y. A multiobjective optimization based framework to balance the global exploration and local exploitation in expensive optimization. *J Glob Optim* **61,** 1–18 (2014).

77. De Ath, G., Everson, R. M., Fieldsend, J. E. & Rahat, A. A. M. ϵ-shotgun: ϵ-greedy batch bayesian optimisation. in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, Cancún, Mexico, 2020), 787–795. ISBN: 9781450371285. https://doi.org/10.1145/3377930.3390154.

78. Wolpert, D. & Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1,** 67–82 (1997).

79. Donachie, M. Titanium: A Technical Guide. *American Society for Testing and Materials,* 29–34 (1988).

80. Bridges, P. & Magnus, B. Manufacture of Titanium Alloy Components for Aerospace and Military Applications, *Cost Effective Application of Titanium Alloys in Military Platforms* (2002).

81. Lütjering, G. & Williams, J. C. Titanium (Engineering Materials and Processes). ISBN: 978-3-540-71398-2. https://www.amazon.com/Titanium-Engineering-Materials-Processes-L (Springer, 2013).

82. Ducobu, F. Contribution à l'étude de la formation du copeau de Ti6Al4V en coupe orthogonale. Approches numérique et expérimentale pour la compréhension des mécanismes de coupe macroscopique et microscopique. fr. PhD thesis (Université de Mons, 2013).

83. Yi, J., Zhou, W. & Deng, Z. Experimental Study and Numerical Simulation of the Intermittent Feed High-Speed Grinding of TC4 Titanium Alloy. *Metals* **9.** ISSN: 2075-4701. https://www.mdpi.com/2075-4701/9/7/802 (2019).

84. Markopoulos, A. Finite Element Method in Machining Processes. ISBN: 978-1-4471-4329-1 (ASM, 2012).

85. Ducobu, F., Rivière-Lorphèvre, E. & Filippi, E. Application of the Coupled Eulerian-Lagrangian (CEL) method to the modeling of orthogonal cutting. *European Journal of Mechanics - A/Solids* **Volume 59,** 58–66 (2016).

86. Kugalur-Palanisamy, N., Rivière-Lorphèvre, E., Arrazola, P. J. & Ducobu, F. Comparison of Johnson-Cook and modified Johnson-Cook material constitutive models and their influence on finite element modelling of Ti6Al4V orthogonal cutting process. *PROCEEDINGS OF THE 22ND INTERNATIONAL ESAFORM CONFERENCE ON MATERIAL FORMING: ESAFORM 2019* (2019).

87. Kugalur Palanisamy, N., Riviere, E., Arrazola, P. & Ducobu, F. Influence of Coulomb's Friction Coefficient in Finite Element Modeling of Orthogonal Cutting of Ti6Al4V. *Key Engineering Materials* **926,** 1619–1628 (2022).

88. Melkote, S., Grzesik, W., Outeiro, J., Rech, J., Schulze, V., Attia, H., Arrazola, P., M'Saoubi, R. & Saldana, C. Advances in material and friction data for modelling of metal machining. *CIRP Annals - Manufacturing Technology* **66** (2017).

89. Ducobu, F., Arrazola, P.-J., Rivière-Lorphèvre, E., de Zarate, G. O., Madariaga, A. & Filippi, E. The CEL Method as an Alternative to the Current Modelling Approaches for Ti6Al4V Orthogonal Cutting Simulation. *Procedia CIRP* **58.** 16th CIRP Conference on Modelling of Machining Operations (16th CIRP CMMO), 245 –250. ISSN: 2212-8271. http://www.sciencedirect.com/science/article/pii/S2212827117303700 (2017).

90. Leseur, D. Experimental investigations of material models for Ti-6A1-4V and 2024-T3. https://www.osti.gov/biblio/11977 (1999).

91. Seo, S., Min, O. & Yang, H. Constitutive equation for Ti–6Al–4V at high temperatures measured using the SHPB technique. *International Journal of Impact Engineering - INT J IMPACT ENG* **31,** 735–754 (2005).

92. Ducobu, F., Riviere, E. & Filippi, E. Experimental contribution to the study of the Ti6Al4V chip formation in orthogonal cutting on a milling machine. in. **8** (2014).

93. Ducobu, F., Arrazola, P.-J., Rivière-Lorphèvre, E., de Zarate, G. O., Madariaga, A. & Filippi, E. The CEL Method as an Alternative to the Current Modelling Approaches for Ti6Al4V Orthogonal Cutting Simulation. *Procedia CIRP* **58.** 16th CIRP Conference on Modelling of Machining Operations (16th CIRP CMMO), 245–250. ISSN: 2212-8271. https://www.sciencedirect.com/science/article/pii/S2212827117303700 (2017).

94. Iman, R. L. Latin Hypercube Sampling. in *Encyclopedia of Quantitative Risk Analysis and Assessment* (John Wiley & Sons, Ltd, 2008). ISBN: 9780470061596. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470061596.risk0299. https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470061596.risk0299.

95. Sammut, C. & Webb, G. I. Leave-One-Out Cross-Validation. in *Encyclopedia of Machine Learning* 600–601 (Springer US, Boston, MA, 2010). ISBN: 978-0-387-30164-8. https://doi.org/10.1007/978-0-387-30164-8_469.

96. Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q. & Wilson, A. G. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. in *Advances in Neural Information Processing Systems* (2018).

97. Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G. & Bakshy, E. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. in *Advances in Neural Information Processing Systems 33* (2020). http://arxiv.org/abs/1910.06403.

98. Arrazola, P., Özel, T., Umbrello, D., Davies, M. & Jawahir, I. Recent advances in modelling of metal machining processes. *CIRP Annals - Manufacturing Technology* **62,** 695–718 (2013).

99. Toubeau, J.-F., Vallée, F., De Grève, Z. & Lobry, J. A new approach based on the experimental design method for the improvement of the operational efficiency in Medium Voltage distribution networks. *International Journal of Electrical Power and Energy Systems* **66,** 116 –124. ISSN: 0142-0615 (2015).

100. Toubeau, J.-F., De Grève, Z. & Vallée, F. Medium-Term Multimarket Optimization for Virtual Power Plants: A Stochastic-Based Decision Environment. *IEEE Transactions on Power Systems* **33,** 1399–1410 (2018).

101. Bruninx, K. Improved modeling of unit commitment decisions under uncertainty. PhD thesis (KU Leuven, 2016).

102. Toubeau, J.-F., Bottieau, J., Vallée, F. & De Grève, Z. Deep Learning-Based Multivariate Probabilistic Forecasting for Short-Term Scheduling in Power Markets. *IEEE Transactions on Power Systems* **34,** 1203–1215. ISSN: 0885-8950 (2019).

103. Pandžić, H., Kuzle, I. & Capuder, T. Virtual power plant mid-term dispatch optimization. *Applied Energy* **101,** 134–141 (2013).

104. Alkan, B. & Kaniappan Chinnathai, M. Performance Comparison of Recent Population-Based Metaheuristic Optimisation Algorithms in Mechanical Design Problems of Machinery Components. *Machines* **9.** ISSN: 2075-1702. https://www.mdpi.com/2075-1702/9/12/341 (2021).

105. Kaveh, M. & Mesgari, M. Application of Meta-Heuristic Algorithms for Training Neural Networks and Deep Learning Architectures: A Comprehensive Review. *Neural Processing Letters.* ISSN: 1573-773X. https://doi.org/10.1007/s11063-022-11055-6 (2022).

106. Torres-Jiménez, J. & Pavón, J. Applications of metaheuristics in real-life problems. *Progress in Artificial Intelligence* **2.** ISSN: 2192-6360. https://doi.org/10.1007/s13748-014-0051-8 (2014).

107. Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T. & Crombecq, K. A Surrogate Modeling and Adaptive Sampling Toolbox for Computer Based Design. *J. Mach. Learn. Res.* **11,** 2051–2055 (2010).

108. MATLAB Optimization Toolbox. The MathWorks, Natick, MA, USA.

109. Caldwell, J. M., Le, X., McIntosh, L., Meehan, M. T., Ogunlade, S., Ragonnet, R., O'Neill, G. K., Trauer, J. M. & McBryde, E. S. Vaccines and variants: Modelling insights into emerging issues in COVID-19 epidemiology. *Paediatric Respiratory Reviews* **39,** 32–39. ISSN: 1526-0542. https://www.sciencedirect.com/science/article/pii/S152605422100066X (2021).

110. Trauer, J. M., Ragonnet, R., Doan, T. N. & McBryde, E. S. Modular programming for tuberculosis control, the "AuTuMN" platform. *BMC Infectious Diseases* **17,** 546. ISSN: 1471-2334. https://doi.org/10.1186/s12879-017-2648-6 (2017).

111. Global tuberculosis report 2018. WHO/CDS/TB/2018.20 (World Health Organization, Geneva, 2018). http://www.who.int/tb/publications/global_report/en/.

112. Briffoteaux, G., Ragonnet, R., Mezmaz, M., Melab, N. & Tuyttens, D. Evolution Control for parallel ANN-assisted simulation-based optimization application to Tuberculosis Transmission Control. *Future Generation Computer Systems* **113,** 454–467. ISSN: 0167-739X. https://www.sciencedirect.com/science/article/pii/S0167739X19308635 (2020).

113. Deb, K. & Nain, P. An Evolutionary Multi-objective Adaptive Meta-modeling Procedure Using Artificial Neural Networks. in *Evolutionary Computation in Dynamic and Uncertain Environments* (eds Yang, S., Ong, Y.-S. & Jin, Y.) doi: http://dx.doi.org/10.1007/978-3-540-49774-5_13, 297–322 (Springer, Berlin, Heidelberg, 2007). ISBN: 978-3-540-49772-1.

114. Poloni, C., Giurgevich, A., Onesti, L. & Pediroda, V. Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynam ics. *Computer Methods in Applied Mechanics and Engineering* **186.** doi: https://doi.org/10.1016/S0045-7825(99)00394-1, 403 –420. ISSN: 0045-7825. http://www.sciencedirect.com/science/article/pii/S0045782599003941 (2000).

115. Syberfeldt, A., Grimm, H., Ng, A. & John, R. I. A parallel surrogate-assisted multi-objective evolutionary algorithm for computationally expensive optimization problems. in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* doi: https://doi.org/10.1109/CEC.2008.4631228 (2008), 3177–3184.

116. Gaspar-Cunha, A. & Vieira, A. A Multi-Objective Evolutionary Algorithm Using Neural Networks to Approximate Fitness Evaluations. *International Journal of Computers, Systems and Signals* **6,** 18–36 (2005).

117. Vicario, G., Craparotta, G. & Pistone, G. Meta-models in Computer Experiments: Kriging versus Artificial Neural Networks. *Quality and Reliability Engineering International* **32,** 2055–2065. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.2026. https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.2026 (2016).

118. Gal, Y. Uncertainty in Deep Learning. PhD thesis (University of Cambridge, 2016).

119. Gal, Y. & Ghahramani, Z. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *arXiv e-prints.* arXiv: 1506.02142 [stat.ML] (2015).

120. Biscani, F. & Izzo, D. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software* **5,** 2338. https://doi.org/10.21105/joss.02338 (2020).

121. Jones, D. The DIRECT global optimization algorithm. *Encyclopedia of Optimization* **1** (2001).

122. Jones, D. R. & Martins, J. The DIRECT algorithm: 25 years Later. *J. Glob. Optim.* **79,** 521–566 (2021).

123. Gobert, M., Gmys, J., Melab, N. & Tuyttens, D. Space Partitioning with multiple models for Parallel Bayesian Optimization. in *OLA 2021 - Optimization and Learning Algorithm* (Sicilia / Virtual, Italy, 2021). https://hal.archives-ouvertes.fr/hal-03324642.

124. Rehbach, F., Zaefferer, M., Naujoks, B. & Bartz-Beielstein, T. Expected Improvement versus Predicted Value in Surrogate-Based Optimization. 2020. arXiv: 2001.02957 [cs.NE].

125. Siivola, E., Vehtari, A., Vanhatalo, J., González, J. & Andersen, M. R. Correcting boundary over-exploration deficiencies in Bayesian optimization with virtual derivative sign observations. 2018. arXiv: 1704.00963 [stat.ML].

126. Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6,** 182–197 (2002).

127. Villanueva, D., Le Riche, R., Picard, G. & Haftka, R. Dynamic Design Space Partitioning for Optimization of an Integrated Thermal Protection System. in (2013). ISBN: 978-1-62410-223-3.

128. Wang, G. & Simpson, T. Fuzzy clustering based hierarchical metamodeling for design space reduction and optimization. English (US). *Engineering Optimization* **36,** 313–335. ISSN: 0305-215X (2004).

129. Li, Z., Ruan, S., Gu, J., Wang, X. & Shen, C. Investigation on parallel algorithms in efficient global optimization based on multiple points infill criterion and domain decomposition. *Structural and Multidisciplinary Optimization* **54,** 747–773 (2016).

130. Wang, S. & Hui Ng, S. Partition-Based Bayesian Optimization for Stochastic Simulations. in *2020 Winter Simulation Conference (WSC)* (2020), 2832–2843.

131. Jones, D., Perttunen, C. & Stuckman, B. Lipschitzian Optimisation Without the Lipschitz Constant. *Journal of Optimization Theory and Applications* **79,** 157–181 (1993).

132. Diouane, Y., Picheny, V., Le Riche, R. & Scotto Di Perrotolo, A. TREGO: a Trust-Region Framework for Efficient Global Optimization. *Journal of Global Optimization* **86,** 1–23 (2022).

133. Li, Q., Fu, A., Wei, W. & Zhang, Y. A Trust Region Based Local Bayesian Optimization without Exhausted Optimization of Acquisition Function. *Evolving Systems,* 1–20 (2022).

134. Briffoteaux, G. pySBO: Python framework for Surrogate-Based Optimization. https://pysbo.readthedocs.io/. 2021.

135. Martinez-Cantin, R. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. *Journal of Machine Learning Research* **15,** 3735–3739 (2014).

136. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. 2022. arXiv: 1312.6114 [stat.ML].

137. Pleiss, G., Jankowiak, M., Eriksson, D., Damle, A. & Gardner, J. R. Fast Matrix Square Roots with Applications to Gaussian Processes and Bayesian Optimization. 2020. arXiv: 2006.11267 [cs.LG].

138. Bossek, J., Doerr, C. & Kerschke, P. Initial Design Strategies and their Effects on Sequential Model-Based Optimization An Exploratory Case Study Based on BBOB. in *Genetic and Evolutionary Computation Conference (GECCO'20)* (Cancun, Mexico, 2020). https://hal.sorbonne-universite.fr/hal-02871959.

139. Neal, R. Bayesian Learning for Neural Networks. *Lecture Notes in Statistics* (1996).

140. Toubeau, J.-F., Bottieau, J., De Grève, Z., Vallée, F. & Bruninx, K. Data-Driven Scheduling of Energy Storage in Day-Ahead Energy and Reserve Markets With Probabilistic Guarantees on Real-Time Delivery. *IEEE Transactions on Power Systems* **36,** 2815–2828 (2021).

141. Toubeau, J.-F., De Grève, Z., Goderniaux, P., Vallée, F. & Bruninx, K. Chance-Constrained Scheduling of Underground Pumped Hydro Energy Storage in Presence of Model Uncertainties. *IEEE Transactions on Sustainable Energy* **11,** 1516–1527 (2020).

142. Taktak, R. & D'Ambrosio, C. An overview on mathematical programming approaches for the deterministic unit commitment problem in hydro valleys. *Energy Systems* **8,** 1–23 (2017).

143. Steeger, G., Barroso, L. & Rebennack, S. Optimal Bidding Strategies for Hydro-Electric Producers: A Literature Survey. *Power Systems, IEEE Transactions on* **29,** 1758–1766 (2014).

144. Abreu, L. V. L., Khodayar, M. E., Shahidehpour, M. & Wu, L. Risk-Constrained Coordination of Cascaded Hydro Units With Variable Wind Power Generation. *IEEE Transactions on Sustainable Energy* **3,** 359–368 (2012).

145. Montero, R., Wortberg, T, Binias, J & Niemann, A. Integrated assessment of underground pumped-storage facilities using existing coal mine infrastructure. in (2016), 953–960. ISBN: 978-1-138-02977-4.

146. Pujades, E., Orban, P., Bodeux, S., Archambeau, P., Erpicum, S. & Dassargues, A. Underground pumped storage hydropower plants using open pit mines: How do groundwater exchanges influence the efficiency? *Applied Energy* **190,** 135–146. ISSN: 0306-2619. https://www.sciencedirect.com/science/article/pii/S0306261916318608 (2017).

147. Ponrajah, R., Witherspoon, J. & Galiana, F. Systems to Optimize Conversion Efficiencies at Ontario Hydro's Hydroelectric Plants. *Power Systems, IEEE Transactions on* **13,** 1044 –1050 (1998).

148. Pannatier, Y. Optimisation des stratégies de réglage d'une installation de pompage-turbinage à vitesse variable (2010).

149. Artiba, A., Emelyanov, V. & Iassinovski, S. Introduction to Intelligent Simulation: The RAO Language. *The Journal of the Operational Research Society* **51** (2000).

150. Cheng, C., Wang, J. & Wu, X. Hydro Unit Commitment With a Head-Sensitive Reservoir and Multiple Vibration Zones Using MILP. *IEEE Transactions on Power Systems* **31,** 4842–4852 (2016).

151. Arce, A., Ohishi, T. & Soares, S. Optimal dispatch of generating units of the Itaipu hydroelectric plant. *IEEE Transactions on Power Systems* **17,** 154–158 (2002).

152. Catalao, J. P. S., Mariano, S. J. P. S., Mendes, V. M. F. & Ferreira, L. A. F. M. Scheduling of Head-Sensitive Cascaded Hydro Systems: A Nonlinear Approach. *IEEE Transactions on Power Systems* **24,** 337–346 (2009).

153. Chen, P.-H. & Chang, H.-C. Genetic aided scheduling of hydraulically coupled plants in hydro-thermal coordination. *IEEE Transactions on Power Systems* **11,** 975–981 (1996).

154. Yu, B., Yuan, X. & Wang, J. Short-term hydro-thermal scheduling using particle swarm optimization method. *Energy Conversion and Management* **48,** 1902–1908. ISSN: 0196-8904. https://www.sciencedirect.com/science/article/pii/S0196890407000489 (2007).

155. Favaro, P., Dolányi, M., Vallée, F. & Toubeau, J.-F. Neural network informed day-ahead scheduling of pumped hydro energy storage. *Energy* **289,** 129999. ISSN: 0360-5442. https://www.sciencedirect.com/science/article/pii/S0360544223033935 (2024).

156. Priem, R. Optimisation bayésienne sous contraintes et en grande dimension appliquée à la conception avion avant projet. Theses (ISAE-SUPAERO, 2020). https://hal.science/tel-03096022.

157. Priem, R., Bartoli, N., Diouane, Y., Dubreuil, S. & Saves, P. High-dimensional efficient global optimization using both random and supervised embeddings. in *AIAA AVIATION 2023 Forum* (). eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2023-4448. https://arc.aiaa.org/doi/abs/10.2514/6.2023-4448.

158. Storti, G. L., Paschero, M., Rizzi, A. & Frattale Mascioli, F. M. Comparison between time-constrained and time-unconstrained optimization for power losses minimization in Smart Grids using genetic algorithms. *Neurocomputing* **170.** Advances on Biological Rhythmic Pattern Generation: Experiments, Algorithms and Applications Selected Papers from the 2013 International Conference on Intelligence Science and Big Data Engineering (IScIDE 2013) Computational Energy Management in Smart Grids, 353–367.

ISSN: 0925-2312. https://www.sciencedirect.com/science/article/pii/S0925231215008772 (2015).

159. Le Riche, R. & Picheny, V. Revisiting Bayesian Optimization in the light of the COCO benchmark. *Structural and Multidisciplinary Optimization*. https://hal.archives-ouvertes.fr/hal-03188590 (2021).

160. Holland, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. ISBN: 9780262275552. https://doi.org/10.7551/mitpress/1090.001.0001 (The MIT Press, 1992).

161. Poli, R., Kennedy, J. & Blackwell, T. M. Particle swarm optimization. *Swarm Intelligence* **1,** 33–57 (1995).

162. Briffoteaux, G., Ragonnet, R., Mezmaz, M., Melab, N. & Tuyttens, D. Evolution Control for parallel ANN-assisted simulation-based optimization application to Tuberculosis Transmission Control. *Future Generation Computer Systems* **113,** 454–467. ISSN: 0167-739X. https://www.sciencedirect.com/science/article/pii/S0167739X19308635 (2020).

163. Díaz-Manríquez, A., Toscano, G., Barron-Zambrano, J. & Tello-Leal, E. A Review of Surrogate Assisted Multiobjective Evolutionary Algorithms. *Computational Intelligence and Neuroscience* **2016.** doi: https://doi.org/10.1155/2016/9420460, 14. ISSN: Article ID 9420460. https://www.hindawi.com/journals/cin/2016/9420460/ (2016).

164. Sun, C., Jin, Y., Cheng, R., Ding, J. & Zeng, J. Surrogate-Assisted Cooperative Swarm Optimization of High-Dimensional Expensive Problems. *IEEE Transactions on Evolutionary Computation* **21,** 644–660 (2017).

165. Chen, T., Tang, K., Chen, G. & Yao, X. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science* **436,** 54–70. ISSN: 0304-3975. https://www.sciencedirect.com/science/article/pii/S0304397511001368 (2012).

166. Jin, Y., Olhofer, M. & Sendhoff, B. On Evolutionary Optimization with Approximate Fitness Functions. in *Proceedings of the 2Nd Annual Conference on Genetic and Evolutionary Computation* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000), 786–793. ISBN: 1-55860-708-0. http://dl.acm.org/citation.cfm?id=2933718.2933864.

167. Jin, Y., Olhofer, M. & Sendhoff, B. Managing approximate models in evolutionary aerodynamic design optimization. in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)* **1** (2001), 592–599 vol. 1.

168. Khokhar, M. A., Boudt, K. & Wan, C. Cardinality-Constrained Higher-Order Moment Portfolios Using Particle Swarm Optimization. in, 169–187 (2021). ISBN: 978-3-030-70280-9.

169. Clerc, M. & Kennedy, J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* **6,** 58–73 (2002).

170. Carroll, E. Multi-Swarm Adaptive Velocity PSO for Constrained Engineering Problems. PhD thesis (2017).

171. Shi, L. & Rasheed, K. A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms. in *Computational Intelligence in Expensive Optimization Problems* 3–28 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2010). ISBN: 978-3-642-10701-6.

172. Briffoteaux, G., Ragonnet, R., Mezmaz, M., Melab, N. & Tuyttens, D. Evolution Control Ensemble Models for Surrogate-Assisted Evolutionary Algorithms. in *High Performance Computing and Simulation 2020* (Barcelona, Spain, 2021). https://hal.inria.fr/hal-03332521.

173. Boeringer, D. & Werner, D. Particle swarm optimization versus genetic algorithms for phased array synthesis. *IEEE Transactions on Antennas and Propagation* **52,** 771–779 (2004).

174. Eberhart, R. C. & Shi, Y. Comparison between Genetic Algorithms and Particle Swarm Optimization. in *Evolutionary Programming* (1998). https://api.semanticscholar.org/CorpusID:14050546.

175. Dalcin, L. & Fang, Y.-L. L. mpi4py: Status Update After 12 Years of Development. *Computing in Science & Engineering* **23,** 47–54 (2021).

176. Chen, Q., Liu, B., Zhang, Q. F., Liang, J. J., Suganthan, P. N. & Qu, B. Problem Definitions and Evaluation Criteria for CEC 2015 Special Session on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization. in (2015).

177. Liang, J., Qu, B. & Suganthan, P. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Tech. rep. (2013).

178. Clerc, M. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. in. **3** (1999), 1957 Vol. 3. ISBN: 0-7803-5536-9.

179. Derrac, J., García, S., Molina, D. & Herrera, F. A practical tutorial on the use of non-parametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation* **1,** 3–18. https://app.dimensions.ai/details/publication/pub.1011052808 (2011).

180. Kumar, A., Wu, G., Ali, M. Z., Mallipeddi, R., Suganthan, P. N. & Das, S. A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. *Swarm and Evolutionary Computation* **56,** 100693. ISSN: 2210-6502. https://www.sciencedirect.com/science/article/pii/S2210650219308946 (2020).

181. Xiao, Y.-N., Guo, Y., Cui, H., Wang, Y., Li, J. & Zhang, Y. IHAOAVOA: An improved hybrid aquila optimizer and African vultures optimization algorithm for global optimization problems. *Mathematical biosciences and engineering: MBE* **19,** 10963–11017 (2022).

182. Thieu, N. V. ENOPPY: A Python Library for Engineering Optimization Problems. 2023. https://github.com/thieu1995/enoppy.

183. Moriconi, R., Deisenroth, M. & K S, S. K. High-dimensional Bayesian optimization using low-dimensional feature spaces. *Machine Learning* **109,** 1925–1943 (2020).

184. Toufani, P., Karakoyun, E. C., Nadar, E., Fosso, O. B. & Kocaman, A. S. Optimization of pumped hydro energy storage systems under uncertainty: A review. *Journal of Energy Storage* **73,** 109306. ISSN: 2352-152X. https://www.sciencedirect.com/science/article/pii/S2352152X23027044 (2023).

185. Vivien Lai, Van Son Lai, Huang, Y. F., Yuk Feng Huang, *et al.* A Review of Reservoir Operation Optimisations: from Traditional Models to Metaheuristic Algorithms. *Archives of Computational Methods in Engineering*. MAG ID: 4214558369 (2022).

186. Hebbal, A., Balesdent, M., Brevault, L., Melab, N. & Talbi, E.-G. Deep Gaussian process for multi-objective Bayesian optimization. *Optimization and Engineering* **24,** 1–40 (2022).

187. Srinivas, N. & Deb, K. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation* **2,** 221–248. https://api.semanticscholar.org/CorpusID:13997318 (1994).

List of Figures

List of Tables

List of Algorithms

Appendices

Appendix A

Mathematics for Bayesian Optimization

A.1 Notions of Probability and Statistics

Definition A.1.1. *Common estimators*

Given a set of observations $(X_i)_{i=1,\dots,n}$ *of a random variable* X*, standard estimators of the expectation, variance are respectively:*

- $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$
- $s^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$ *or* $s_*^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$

Definition A.1.2. *Normal distribution, also known as the Gaussian distribution*

Let $\boldsymbol{x}$ *be a random vector with dimension* d *following a gaussian law. The distribution is defined by its mean* $\boldsymbol{\mu}$ *and its covariance matrix* Σ*. We write* $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$

The distribution of $\boldsymbol{x}$ *given* $\boldsymbol{\mu}, \Sigma$ *writes:*

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{A.1}$$

Proposition A.1.1. *Gaussian moments*

$$\mathbb{E}[\boldsymbol{x}] = \boldsymbol{\mu} \tag{A.2}$$

$$\mathbb{C}ov(x_i, x_j) = \mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)] = \mathbb{E}[x_i x_j] - \mu_i \mu_j = \Sigma_{i,j} \tag{A.3}$$

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] = \boldsymbol{\mu}\boldsymbol{\mu}^T + \Sigma \tag{A.4}$$

Proposition A.1.2. *Linear transformation of a normal distribution*

Let $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $Y = AX + \boldsymbol{b}$ any linear transformation.

Then $Y \sim \mathcal{N}(A\boldsymbol{\mu} + \boldsymbol{b}, A\Sigma A^T)$

Proposition A.1.3. *Inversion of a sum of two matrices*

Given A, U, C, V matrices such that $A + UCV$ is invertible:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1} \tag{A.5}$$

Proposition A.1.4. *Properties of symmetric matrices*

Let A and B be (real) symmetric matrices:

- *If A^{-1} exists, then A^{-1} is also symmetric, $(A^{-1})^T = A^{-1}$*
- *The sum of symmetric matrices is also symmetric, $\forall a, b \in \mathbb{R},\ aA + bB$ is symmetric.*

A.2 Gaussian Process Regression

Lemma A.2.1.

$$p\left(\boldsymbol{a}|\boldsymbol{X}, \boldsymbol{y}\right) \propto \exp\left(-\frac{1}{2}\left(\boldsymbol{a} - \bar{\boldsymbol{a}}\right)^T\left(\frac{1}{\sigma^2}XX^T + \Sigma^{-1}\right)\left(\boldsymbol{a} - \bar{\boldsymbol{a}}\right)\right)$$

Proof. Starting from Equation 1.6

$$\begin{aligned}
p\left(\boldsymbol{a}|\boldsymbol{X}, \boldsymbol{y}\right) &\propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y} - X^T\boldsymbol{a}\right)^T\left(\boldsymbol{y} - X^T\boldsymbol{a}\right)\right)\exp\left(-\frac{1}{2}\boldsymbol{a}^T\Sigma_p^{-1}\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y}^T - \boldsymbol{a}^TX\right)\left(\boldsymbol{y} - X^T\boldsymbol{a}\right) - \frac{1}{2}\boldsymbol{a}^T\Sigma_p^{-1}\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^TX^T\boldsymbol{a} - \boldsymbol{a}^TX\boldsymbol{y} + \boldsymbol{a}^TXX^T\boldsymbol{a}\right) - \frac{1}{2}\boldsymbol{a}^T\Sigma_p^{-1}\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^TX^T\boldsymbol{a} - \boldsymbol{a}^TX\boldsymbol{y}\right) - \frac{1}{2}\boldsymbol{a}^T\left(\frac{1}{\sigma^2}XX^T + \Sigma_p^{-1}\right)\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^TX^T\boldsymbol{a} - \boldsymbol{a}^TX\boldsymbol{y}\right) - \frac{1}{2}\boldsymbol{a}^TA\boldsymbol{a}\right) \text{ with } A = \left(\frac{1}{\sigma^2}XX^T + \Sigma^{-1}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T\boldsymbol{y} + \frac{1}{2\sigma^2}\boldsymbol{y}^TX^T\boldsymbol{a} + \frac{1}{2\sigma^2}\boldsymbol{a}^TX\boldsymbol{y} - \frac{1}{2}\boldsymbol{a}^TA\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T\boldsymbol{y} + \frac{1}{\sigma^2}(X\boldsymbol{y})^T\boldsymbol{a} - \frac{1}{2}\boldsymbol{a}^TA\boldsymbol{a}\right)
\end{aligned}$$

With the completing the square trick:

$$ax^2 + 2bx + c = a(x^2 + 2\frac{b}{a}x) + c = a(x + \frac{b}{a})^2 + c - \frac{b^2}{2}$$

and its equivalent in matrix form using M symmetric, and then $M^T = M$ and ${M^{-1}}^T = M^{-1}$ (A.1.4)

$$\begin{aligned}
\boldsymbol{x}^T M\boldsymbol{x} - 2\boldsymbol{b}^T\boldsymbol{x} &= \boldsymbol{x}^T M\boldsymbol{x} - 2\boldsymbol{b}^T M^{-1} M\boldsymbol{x} \\
&= \boldsymbol{x}^T M\boldsymbol{x} - 2\boldsymbol{u}^T M\boldsymbol{x} && \text{with } \boldsymbol{u} = M^{-1}\boldsymbol{b} \\
&= \boldsymbol{x}^T M\boldsymbol{x} - 2\boldsymbol{u}^T M\boldsymbol{x} + \boldsymbol{u}^T M\boldsymbol{u} - \boldsymbol{u}^T M\boldsymbol{u} \\
&= (\boldsymbol{x} - \boldsymbol{u})^T M(\boldsymbol{x} - \boldsymbol{u}) - \boldsymbol{u}^T M\boldsymbol{u}
\end{aligned}$$

In previous expression, replacing by the following notations $\boldsymbol{x} = \boldsymbol{a}$, $M = A$, $\boldsymbol{b} = X\boldsymbol{y}$, and using $\boldsymbol{u} = \frac{1}{\sigma^2} A^{-1} X\boldsymbol{y}$, we can derive:

$$\begin{aligned}
\frac{1}{\sigma^2}(X\boldsymbol{y})^T\boldsymbol{a} - \frac{1}{2}\boldsymbol{a}^T A\boldsymbol{a} &= -\frac{1}{2}\left(\boldsymbol{a}^T A\boldsymbol{a} - \frac{2}{\sigma^2}\boldsymbol{a}^T X\boldsymbol{y}\right) \\
&= (\boldsymbol{a} - \boldsymbol{u})^T A(\boldsymbol{a} - \boldsymbol{u}) - \boldsymbol{u}^T A\boldsymbol{u}
\end{aligned}$$

Denoting $\bar{\boldsymbol{a}} = \frac{1}{\sigma^2} A^{-1} X\boldsymbol{y}$, $A = \left(\frac{1}{\sigma^2} XX^T + \Sigma^{-1}\right)$ symmetric, we have:

$$\begin{aligned}
p\left(\boldsymbol{a}|\boldsymbol{X}, \boldsymbol{y}\right) &\propto \exp\left(-\frac{1}{2\sigma^2}\boldsymbol{y}^T\boldsymbol{y} + \frac{1}{\sigma^2}(X\boldsymbol{y})^T\boldsymbol{a} - \frac{1}{2}\boldsymbol{a}^T A\boldsymbol{a}\right) \\
&\propto \exp\left(-\frac{1}{2}(\boldsymbol{a} - \bar{\boldsymbol{a}})^T A(\boldsymbol{a} - \bar{\boldsymbol{a}}) - \bar{\boldsymbol{a}}^T A\bar{\boldsymbol{a}} - \frac{1}{2\sigma^2}\boldsymbol{y}^T\boldsymbol{y}\right) \\
&\propto \exp\left(-\frac{1}{2}(\boldsymbol{a} - \bar{\boldsymbol{a}})^T A(\boldsymbol{a} - \bar{\boldsymbol{a}})\right) \\
&\propto \exp\left(-\frac{1}{2}\left(\boldsymbol{a} - \bar{\boldsymbol{a}}\right)^T \left(\frac{1}{\sigma^2} XX^T + \Sigma^{-1}\right)\left(\boldsymbol{a} - \bar{\boldsymbol{a}}\right)\right)
\end{aligned}$$

□

Lemma A.2.2. *Equivalent form a the predictive law*

$$\begin{aligned}
y^*|\Phi, \boldsymbol{y}, \boldsymbol{x}^* &\sim \mathcal{N}\left(\frac{1}{\sigma^2}\boldsymbol{\phi}^{*T} A^{-1}\Phi\boldsymbol{y}, \boldsymbol{\phi}^{*T} A^{-1}\boldsymbol{\phi}^*\right) \\
&= \mathcal{N}(\boldsymbol{\phi}^{*T}\Sigma\Phi(K + \sigma^2 I)^{-1}\boldsymbol{y}, \boldsymbol{\phi}^{*T}\Sigma\boldsymbol{\phi}^* - \boldsymbol{\phi}^{*T}\Sigma\Phi(K + \sigma^2 I)^{-1}\Phi^T\Sigma\boldsymbol{\phi}^*)
\end{aligned}$$

Proof. Let us set $K = \Phi^T\Sigma\Phi$, and recall $A = \left(\frac{1}{\sigma^2}\Phi\Phi^T + \Sigma^{-1}\right)$.

$$\begin{aligned}
A\Sigma\Phi &= \left(\frac{1}{\sigma^2}\Phi\Phi^T + \Sigma^{-1}\right)\Sigma\Phi \\
&= \frac{1}{\sigma^2}\Phi\Phi^T\Sigma\Phi + \Phi \\
&= \frac{1}{\sigma^2}\Phi K + \Phi \\
&= \frac{1}{\sigma^2}\Phi(K + \sigma^2 I)
\end{aligned}$$

Then, multiplying by A^{-1} on the left and $(K + \sigma^2 I)^{-1}$ on the right:

$$\begin{aligned} A\Sigma\Phi &= \frac{1}{\sigma^2}\Phi(K + \sigma^2 I) \\ \Leftrightarrow \qquad \Sigma\Phi(K + \sigma^2 I)^{-1} &= \frac{1}{\sigma^2}A^{-1}\Phi \end{aligned}$$

so for the mean:

$$\frac{1}{\sigma^2}\boldsymbol{\phi}^{*T}A^{-1}\Phi\boldsymbol{y} = \boldsymbol{\phi}^{*T}\Sigma\Phi(K + \sigma^2 I)^{-1}\boldsymbol{y}$$

The equivalence regarding the variance expression uses the matrix inversion lemma A.1.3 with $U = \Phi$ and $V = \Phi^T$, $A = \Sigma^{-1}$, $C = I$:

$$\begin{aligned} A^{-1} &= \left(\frac{1}{\sigma^2}\Phi\Phi^T + \Sigma^{-1}\right)^{-1} \\ &= \Sigma - \Sigma\Phi(\sigma^2 I + \Phi\Sigma\Phi^T)^{-1}\Phi^T\Sigma \\ &= \Sigma - \Sigma\Phi(\sigma^2 I + K)^{-1}\Phi^T\Sigma \end{aligned}$$

Multiplying by $\boldsymbol{\phi}^{*T}$ and $\boldsymbol{\phi}^*$ directly gives the desired expression. □

Appendix B

Multi-Objective Optimization

B.1 Multi-Objective Formulation

Definition B.1.1. *Formulation of multi-objective problem*

For a given number of objectives, we consider the multi-dimensional function $F = (f_1, \ldots, f_n)$ *where* $f_i : \Omega \subset \mathbb{R}^d \to \mathbb{R}$*. Assuming a minimization problem, it writes:*

$$\min_{\boldsymbol{x} \in \Omega} \left(f_i(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}) \right).$$

Definition B.1.2. *Pareto Dominance*

Given $\boldsymbol{x}^{(1)}$ *and* $\boldsymbol{x}^{(2)}$*, we say that* $\boldsymbol{x}^{(1)}$ *dominates* $\boldsymbol{x}^{(2)}$ *if:*

$$\begin{aligned} &\forall i \in \{1, \ldots, n\}, f_i\left(\boldsymbol{x}^{(1)}\right) \leq f_i\left(\boldsymbol{x}^{(2)}\right), \\ &\exists i \in \{1, \ldots, n\}, f_i\left(\boldsymbol{x}^{(1)}\right) < f_i\left(\boldsymbol{x}^{(2)}\right). \end{aligned}$$

$\boldsymbol{x}^{(1)}$ *is better than* $\boldsymbol{x}^{(2)}$ *in at least one objective, without deteriorating the other ones.*

We write $\boldsymbol{x}^{(1)} \prec \boldsymbol{x}^{(2)}$ *in case of minimization, meaning* $\boldsymbol{x}^{(1)}$ *is preferable to* $\boldsymbol{x}^{(2)}$*.*

Definition B.1.3. *Pareto Optimal Set*

Given a population $\mathcal{P}$*,* $\boldsymbol{x}^* \in \mathcal{P}$ *is said non-dominated if:*

$$\forall \boldsymbol{x} \in \mathcal{P}, \boldsymbol{x}^* \prec \boldsymbol{x}.$$

The Pareto set, or Pareto front, refers to the set of non-dominated solution.

Example: In a two dimensional objective space, assuming the minimization of both objectives (f_1 and f_2) the Pareto set can be visualized as shown in Figure B.1.

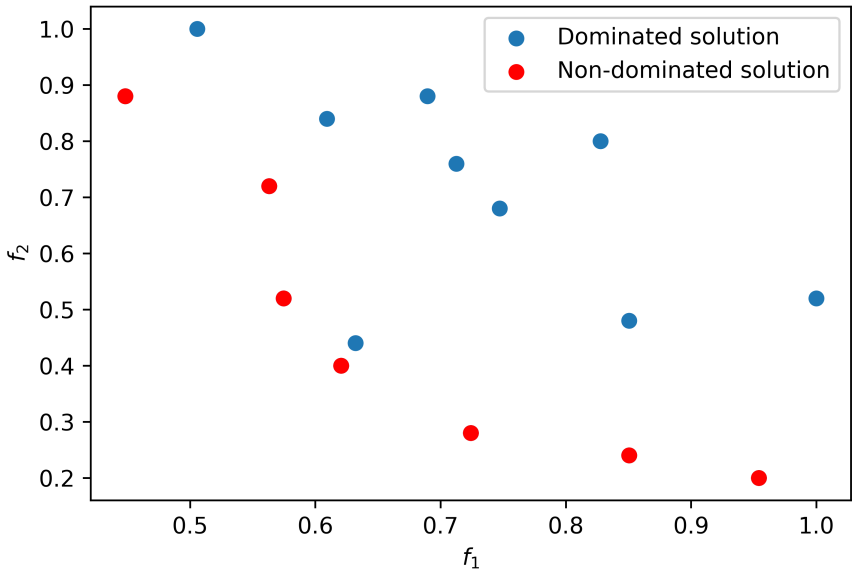


Figure B.1: Example of a Pareto front in a two dimensional objective space.

B.2 Multi-Objective Algorithms

Non-dominated Sorting Genetic Algorithm (NSGA)

NSGA [187] is an EA where the selection operator takes into account the diversity of the solution. First, the Pareto set is identified and all its individuals are attributed a very good dummy fitness value. The same value is assigned to give an equal reproductive potential. Then a sharing method is used for each individual in the Pareto front to degrade the fitness of individuals around it.

The previous steps are repeated after removing the already-treated individuals from the population. This gives another Pareto set that is attributed to a lower fitness than the previous set. The steps are repeated until all individuals from the population are classified into several fronts. This is the non-dominated sorting procedure and each front is referred to as F_i, where $i = 1$ is the best one.

In NSGA-II, the sorting strategy is improved to be faster. The sharing strategy is replaced by a crowded-comparison that does not require any user-defined parameter and has a better computational complexity. NSGA-II is summarized in Algorithm 11 where we can see the classical operators with the special non-dominated sorting procedure inserted at line 3 before the generation of the offspring population. The parents are selected as shown in line 4 according to the rank of their Pareto front, and the crowding-distance. The rest of the algorithm operates as the EA described in Algorithm 3.

Algorithm 11 Non-dominated Sorting Genetic Algorithm II

Input
f: objective function
Ω: design space
p_{size}: population size

1: $\mathcal{P} = sampling(f, \Omega, p_{size})$
2: **while** Budget available **do**
3: $\mathcal{F} =$ non-dominated_sorting$(\mathcal{P})$
4: $\mathcal{P}_{par} =$ selection$(\mathcal{F}, p_{size})$ ▷ based on the crowding-distance and front's rank
5: $\mathcal{P}_{off} =$ reproduction$(\mathcal{P}_{par})$
6: $\mathcal{P}_{off} =$ evaluation(f, P_{off})
7: $\mathcal{P} = \mathcal{P}_{par} \cup \mathcal{P}_{off}$
8: **end while**
9: **return** best_individual$(\mathcal{P})$

Appendix C

Benchmarking Optimization Algorithms

This chapter is dedicated to present the benchmark functions used in the analysis of Section 3.3. We also give additional results helping for a finer analysis of the results.

C.1 Usual Benchmark Functions

The following functions are used in Chapter 3 to conduct our experimentation. We provide their analytical expression as well as the domain they are optimized on. The domain may vary from the literature as we wanted to avoid symmetrical domains when the global optimum is located in its center. Indeed, in such configuration, the splitting scheme of BSP-based algorithms happens to divide the domain exactly on the optimum.

Definition C.1.1. *Rosenbrock function*

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1} \left[100(x_{i=1} - x_i^2)^2 + x_i - 1)^2\right] \tag{C.1}$$

with $\boldsymbol{x} \in [-5, 10]^d$*, and* $\boldsymbol{x}[i] = x_i,\ \forall i \in \{1, \dots d\}$.

Definition C.1.2. *Ackley function*

$$f(\boldsymbol{x}) = -20exp(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(2\pi x_i)\right) + 20 + e \tag{C.2}$$

with $\boldsymbol{x} \in [-15, 30]^d$.

Definition C.1.3. *Schwefel function*

$$f(\boldsymbol{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{|x_i|}\right) \tag{C.3}$$

with $\boldsymbol{x} \in [-500, 500]^d$.

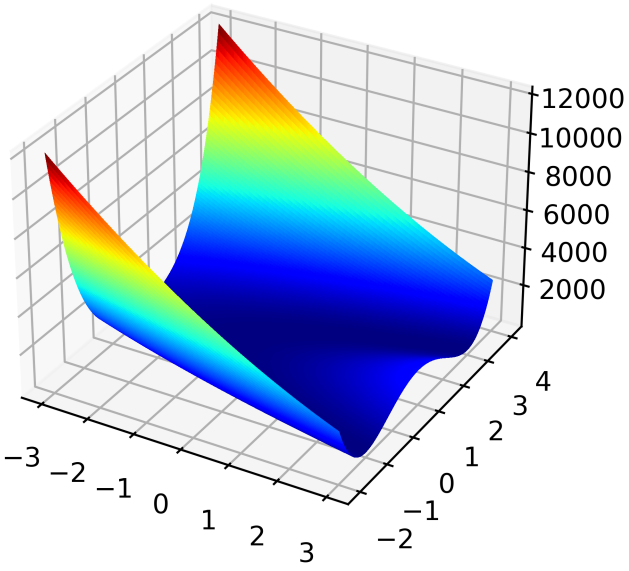


Figure C.1: Rosenbrock 2D landscape

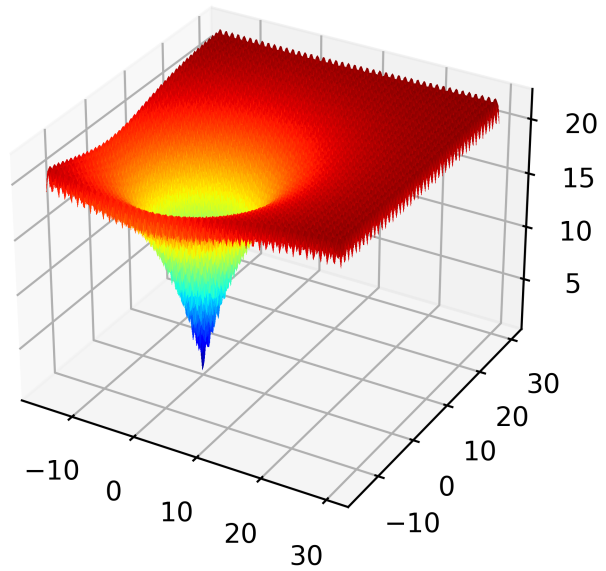


Figure C.2: Ackley 2D landscape

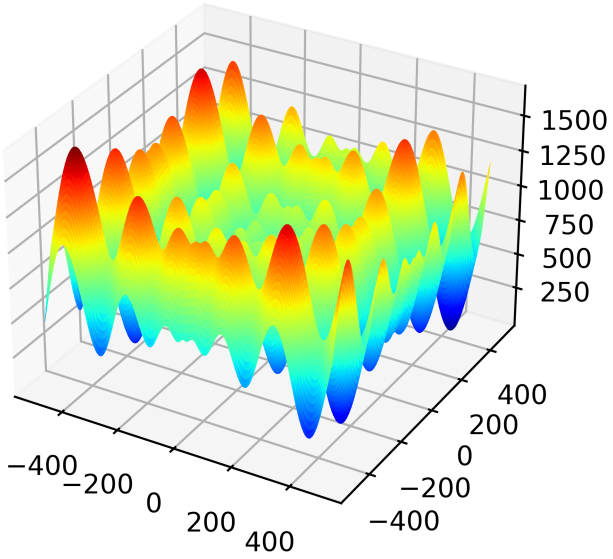


Figure C.3: Schwefel 2D landscape

Definition C.1.4. *Alpine02 function*

$$f(\boldsymbol{x}) = \prod_{i=1}^{d} \sqrt{x_i} \sin(x_i) \tag{C.4}$$

with $\boldsymbol{x} \in [-100, 100]^d$*, re-scaled into* $\boldsymbol{x} \in [0, 10]^d$.

Definition C.1.5. *Rastrigin function*

$$f(\boldsymbol{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{C.5}$$

with $\boldsymbol{x} \in [-4.12, 7.12]^d$.

C.2 Additional Results of the Benchmark Analysis

The following tables present the results for the five benchmark functions investigated in Chapter 3. Each score between 0 and 1 is computed by scaling the outcome using the known minimum, and best initial sample. Consequently a score close to 0 indicates a good performance, and a score close to 1 indicates a poor improvement compared to the initial sampling. A color scale is added to the tables to be visually interpretable. The color scale is adapted for each table so that only the best performing algorithms are highlighted.

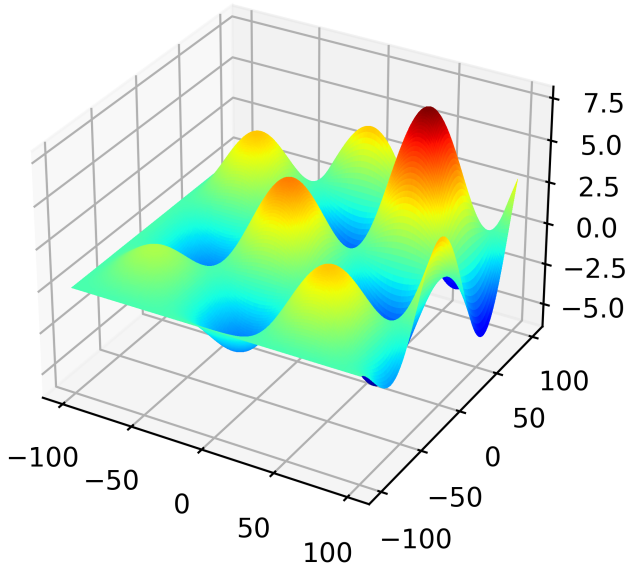


Figure C.4: Alpine02 2D landscape

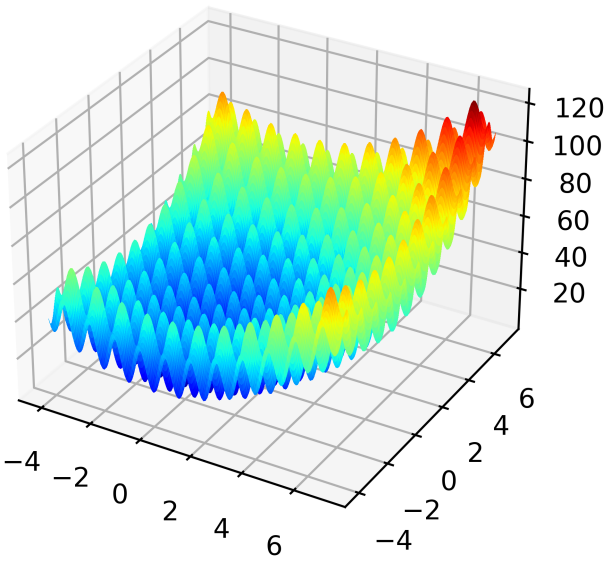


Figure C.5: Rastrigin 2D landscape

Table C.1: Scaled outcome for the **6d-Rosenbrock** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

		0	30	60	120	180	300	600	900	1200
Method	q									
qEGO	2	1	0.0869	0.0353	0.0228	0.0182	0.0135	0.0076	0.0056	0.0043
	4	1	0.1005	0.0510	0.0281	0.0207	0.0137	0.0080	0.0055	0.0045
	8	1	0.0949	0.0408	0.0267	0.0228	0.0150	0.0103	0.0077	0.0066
	16	1	0.1037	0.0540	0.0335	0.0260	0.0171	0.0119	0.0093	0.0092
	32	1	0.1160	0.0548	0.0377	0.0312	0.0255	0.0204	0.0138	0.0107
TS-qEGO	2	1	0.3283	0.2085	0.0896	0.0533	0.0281	0.0149	0.0119	0.0096
	4	1	0.2342	0.1110	0.0482	0.0316	0.0204	0.0126	0.0105	0.0093
	8	1	0.1548	0.0756	0.0297	0.0219	0.0158	0.0128	0.0107	0.0103
	16	1	0.0931	0.0443	0.0213	0.0178	0.0148	0.0120	0.0117	0.0113
	32	1	0.0377	0.0287	0.0224	0.0191	0.0167	0.0150	0.0136	0.0132
KB-qEGO	2	1	0.1206	0.0491	0.0172	0.0113	0.0051	0.0022	0.0014	0.0012
	4	1	0.0958	0.0513	0.0153	0.0085	0.0042	0.0018	0.0014	0.0011
	8	1	0.1159	0.0530	0.0187	0.0092	0.0046	0.0021	0.0013	0.0010
	16	1	0.1292	0.0733	0.0263	0.0165	0.0066	0.0033	0.0022	0.0018
	32	1	0.1629	0.0747	0.0367	0.0180	0.0117	0.0050	0.0035	0.0028
MIC-qEGO	2	1	0.1171	0.0408	0.0165	0.0097	0.0052	0.0019	0.0014	0.0012
	4	1	0.0943	0.0391	0.0128	0.0066	0.0021	0.0008	0.0005	0.0004
	8	1	0.1016	0.0338	0.0130	0.0068	0.0026	0.0010	0.0005	0.0004
	16	1	0.1561	0.0566	0.0205	0.0120	0.0040	0.0018	0.0012	0.0007
	32	1	0.1437	0.0729	0.0333	0.0138	0.0056	0.0020	0.0013	0.0010
MACE	2	1	0.3161	0.2245	0.0784	0.0414	0.0100	0.0037	0.0027	0.0022
	4	1	0.2221	0.1429	0.0236	0.0097	0.0056	0.0021	0.0016	0.0014
	8	1	0.1697	0.0982	0.0168	0.0072	0.0028	0.0015	0.0010	0.0010
	16	1	0.1364	0.0648	0.0141	0.0045	0.0024	0.0013	0.0012	0.0012
	32	1	0.0915	0.0417	0.0167	0.0102	0.0054	0.0013	0.0008	0.0006
BSP-EGO	2	1	0.1801	0.0674	0.0169	0.0104	0.0047	0.0018	0.0013	0.0007
	4	1	0.0954	0.0406	0.0115	0.0071	0.0037	0.0014	0.0009	0.0007
	8	1	0.1104	0.0303	0.0102	0.0051	0.0023	0.0009	0.0007	0.0006
	16	1	0.0997	0.0319	0.0104	0.0062	0.0027	0.0009	0.0004	0.0004
	32	1	0.1406	0.0585	0.0198	0.0161	0.0087	0.0035	0.0024	0.0017
ℓBSP-EGO	2	1	0.0902	0.0210	0.0032	0.0016	0.0005	0.0002	0.0001	0.0001
	4	1	0.0400	0.0058	0.0005	0.0003	0.0002	0.0001	0.0000	0.0000
	8	1	0.0196	0.0040	0.0005	0.0002	0.0001	0.0000	0.0000	0.0000
	16	1	0.0184	0.0040	0.0007	0.0003	0.0002	0.0001	0.0001	0.0001
	32	1	0.0274	0.0114	0.0037	0.0012	0.0004	0.0002	0.0001	0.0001
TuRBO	2	1	0.0192	0.0032	0.0010	0.0005	0.0004	0.0003	0.0003	0.0003
	4	1	0.0070	0.0013	0.0009	0.0008	0.0005	0.0004	0.0004	0.0003
	8	1	0.0043	0.0024	0.0018	0.0013	0.0009	0.0004	0.0003	0.0003
	16	1	0.0046	0.0014	0.0008	0.0007	0.0005	0.0003	0.0003	0.0003
	32	1	0.0146	0.0027	0.0008	0.0004	0.0003	0.0003	0.0003	0.0003

Table C.2: Scaled outcome for the **6d-Alpine02** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.8569	0.8000	0.7911	0.7839	0.7724	0.7678	0.7602	0.7602
	4	1	0.8514	0.8358	0.8330	0.8291	0.8249	0.8217	0.8217	0.8217
	8	1	0.8268	0.7985	0.7920	0.7843	0.7833	0.7831	0.7652	0.7646
	16	1	0.8491	0.8021	0.7756	0.7694	0.7656	0.7597	0.7520	0.7520
	32	1	0.8376	0.8264	0.7866	0.7840	0.7636	0.7529	0.7429	0.7144
TS-qEGO	2	1	0.9816	0.9185	0.9032	0.8794	0.8486	0.8214	0.7853	0.7400
	4	1	0.9184	0.8772	0.8388	0.8333	0.8035	0.7885	0.7826	0.7657
	8	1	0.9074	0.8234	0.7895	0.7846	0.7830	0.7448	0.7316	0.7306
	16	1	0.8895	0.8549	0.8268	0.7802	0.7512	0.7004	0.6852	0.6827
	32	1	0.8717	0.8649	0.8226	0.8029	0.7946	0.7946	0.7607	0.7607
KB-qEGO	2	1	0.7310	0.6796	0.5939	0.5933	0.5571	0.5301	0.4744	0.4682
	4	1	0.7285	0.6629	0.5952	0.5550	0.5099	0.4476	0.4265	0.4200
	8	1	0.7561	0.6806	0.6079	0.5966	0.5443	0.4766	0.4357	0.4221
	16	1	0.8280	0.6671	0.6424	0.5787	0.5420	0.5104	0.4854	0.4854
	32	1	0.8785	0.7835	0.6756	0.6692	0.6471	0.5766	0.5678	0.5678
MIC-qEGO	2	1	0.7557	0.6432	0.6131	0.5912	0.5911	0.5710	0.5259	0.5144
	4	1	0.6349	0.5453	0.4913	0.4689	0.4193	0.3809	0.3806	0.3804
	8	1	0.6537	0.5559	0.4839	0.4481	0.4245	0.3630	0.3481	0.3468
	16	1	0.7456	0.5555	0.4879	0.4738	0.4593	0.4232	0.4186	0.4110
	32	1	0.8661	0.7427	0.5642	0.5021	0.4269	0.3824	0.3572	0.3463
MACE	2	1	0.9273	0.8499	0.7572	0.6489	0.4789	0.2760	0.2194	0.2000
	4	1	0.9234	0.8430	0.6977	0.5394	0.3700	0.3141	0.2824	0.2321
	8	1	0.8730	0.6914	0.4662	0.3291	0.2749	0.1831	0.1638	0.1297
	16	1	0.8399	0.6202	0.4136	0.3482	0.2973	0.2702	0.2491	0.2409
	32	1	0.6134	0.3743	0.3472	0.3467	0.3466	0.3284	0.3073	0.2791
BSP-EGO	2	1	0.6106	0.5376	0.4729	0.4356	0.3906	0.3580	0.3483	0.3483
	4	1	0.5686	0.5553	0.4866	0.4567	0.4438	0.4057	0.4051	0.4051
	8	1	0.5282	0.4923	0.4666	0.4564	0.4507	0.4501	0.4501	0.4501
	16	1	0.5940	0.5564	0.5276	0.5227	0.4976	0.4864	0.4859	0.4859
	32	1	0.6012	0.5602	0.5031	0.4743	0.4666	0.4184	0.3960	0.3768
ℓBSP-EGO	2	1	0.6455	0.6455	0.6449	0.6445	0.6438	0.6429	0.6427	0.6426
	4	1	0.5946	0.5945	0.5945	0.5944	0.5944	0.5939	0.5937	0.5937
	8	1	0.5955	0.5790	0.5550	0.5539	0.5531	0.5523	0.5448	0.5448
	16	1	0.6059	0.5929	0.5559	0.5442	0.5165	0.4222	0.4119	0.4109
	32	1	0.5983	0.5419	0.4718	0.4310	0.3764	0.3432	0.3415	0.3414
TuRBO	2	1	0.6193	0.6051	0.5937	0.5799	0.5767	0.5767	0.5767	0.5767
	4	1	0.5603	0.5574	0.5532	0.5086	0.4936	0.4605	0.4362	0.4243
	8	1	0.6368	0.6349	0.6261	0.6261	0.6261	0.5381	0.5270	0.5177
	16	1	0.5447	0.5446	0.5446	0.5446	0.5446	0.5446	0.5083	0.5041
	32	1	0.3766	0.3240	0.3237	0.3237	0.3237	0.3208	0.2934	0.2724

Table C.3: Scaled outcome for the **6d-Ackley** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.5735	0.5735	0.5663	0.5512	0.5343	0.5343	0.5343	0.5343
	4	1	0.5623	0.5225	0.5119	0.5119	0.5119	0.4912	0.4912	0.4912
	8	1	0.5326	0.5168	0.5168	0.5168	0.5168	0.5168	0.5168	0.5168
	16	1	0.5670	0.5375	0.5375	0.5375	0.5375	0.5375	0.5375	0.5375
	32	1	0.6242	0.6242	0.5700	0.5700	0.5700	0.5700	0.5700	0.5700
TS-qEGO	2	1	0.5797	0.5141	0.4680	0.4403	0.4209	0.3996	0.3915	0.3821
	4	1	0.5328	0.4978	0.4657	0.4574	0.4457	0.4235	0.4144	0.4027
	8	1	0.5365	0.4907	0.4662	0.4583	0.4448	0.4270	0.4157	0.4110
	16	1	0.5583	0.5122	0.4872	0.4761	0.4731	0.4556	0.4377	0.4368
	32	1	0.5443	0.5054	0.4855	0.4650	0.4578	0.4562	0.4442	0.4442
KB-qEGO	2	1	0.3741	0.3741	0.3729	0.3729	0.3729	0.3674	0.3669	0.3669
	4	1	0.4741	0.4130	0.4097	0.4052	0.4052	0.4052	0.4052	0.4052
	8	1	0.4714	0.4589	0.4212	0.4018	0.4018	0.4018	0.4018	0.4018
	16	1	0.5897	0.4739	0.4522	0.4333	0.3943	0.3943	0.3910	0.3910
	32	1	0.7020	0.6040	0.5103	0.4762	0.4762	0.4762	0.4762	0.4762
MIC-qEGO	2	1	0.4165	0.4042	0.3627	0.3627	0.3627	0.3383	0.3383	0.3383
	4	1	0.2126	0.1993	0.1951	0.1943	0.1898	0.1894	0.1888	0.1871
	8	1	0.2041	0.1834	0.1830	0.1810	0.1754	0.1732	0.1722	0.1712
	16	1	0.3176	0.2134	0.1985	0.1976	0.1916	0.1879	0.1871	0.1857
	32	1	0.5134	0.2916	0.1857	0.1829	0.1803	0.1772	0.1766	0.1766
MACE	2	1	0.6418	0.3892	0.2924	0.2323	0.2028	0.1541	0.1340	0.1327
	4	1	0.4282	0.2566	0.2045	0.1852	0.1559	0.1442	0.1383	0.1383
	8	1	0.3105	0.2065	0.1819	0.1579	0.1465	0.1356	0.1348	0.1348
	16	1	0.2529	0.1750	0.1560	0.1456	0.1361	0.1335	0.1322	0.1322
	32	1	0.2051	0.1537	0.1286	0.1252	0.1195	0.1177	0.1177	0.1177
BSP-EGO	2	1	0.2474	0.2474	0.2474	0.2474	0.2474	0.2474	0.2474	0.2448
	4	1	0.1752	0.1752	0.1752	0.1752	0.1752	0.1752	0.1752	0.1752
	8	1	0.1485	0.1485	0.1485	0.1485	0.1485	0.1485	0.1485	0.1485
	16	1	0.1919	0.1553	0.1509	0.1509	0.1509	0.1509	0.1509	0.1509
	32	1	0.2655	0.2540	0.2230	0.1981	0.1793	0.1579	0.1579	0.1579
ℓBSP-EGO	2	1	0.0777	0.0189	0.0146	0.0146	0.0145	0.0145	0.0145	0.0145
	4	1	0.0489	0.0049	0.0026	0.0025	0.0024	0.0024	0.0024	0.0024
	8	1	0.0486	0.0116	0.0011	0.0005	0.0003	0.0001	0.0000	0.0000
	16	1	0.0689	0.0241	0.0055	0.0042	0.0022	0.0007	0.0004	0.0002
	32	1	0.1265	0.0825	0.0451	0.0229	0.0108	0.0050	0.0033	0.0022
TuRBO	2	1	0.1233	0.0964	0.0822	0.0749	0.0699	0.0681	0.0666	0.0647
	4	1	0.0916	0.0774	0.0693	0.0637	0.0565	0.0481	0.0470	0.0458
	8	1	0.0904	0.0784	0.0690	0.0654	0.0593	0.0580	0.0560	0.0542
	16	1	0.0911	0.0679	0.0629	0.0619	0.0580	0.0547	0.0542	0.0541
	32	1	0.1643	0.0888	0.0644	0.0608	0.0600	0.0544	0.0529	0.0528

Table C.4: Scaled outcome for the **6d-Schwefel** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

		0	30	60	120	180	300	600	900	1200
Method	q									
qEGO	2	1	0.6076	0.4811	0.4179	0.4067	0.3760	0.3479	0.3341	0.3192
	4	1	0.6454	0.5090	0.4410	0.4103	0.3838	0.3588	0.3240	0.3202
	8	1	0.6573	0.5724	0.4893	0.4580	0.4414	0.3999	0.3897	0.3878
	16	1	0.6665	0.5882	0.5222	0.5139	0.4856	0.4575	0.4448	0.4271
	32	1	0.7438	0.7025	0.6732	0.6415	0.6231	0.6049	0.5695	0.5565
TS-qEGO	2	1	0.9531	0.8732	0.8245	0.7881	0.7350	0.6423	0.6295	0.6158
	4	1	0.8372	0.8002	0.7194	0.6879	0.6224	0.5884	0.5800	0.5624
	8	1	0.8365	0.7241	0.6935	0.6647	0.6190	0.5800	0.5660	0.5616
	16	1	0.7550	0.6958	0.6384	0.6036	0.5765	0.5633	0.5452	0.5302
	32	1	0.7134	0.6423	0.6161	0.5929	0.5728	0.5373	0.5050	0.4769
KB-qEGO	2	1	0.6736	0.5070	0.4232	0.4064	0.3807	0.3505	0.3251	0.3197
	4	1	0.6588	0.5424	0.4337	0.4091	0.3743	0.3434	0.3319	0.3183
	8	1	0.8099	0.7057	0.5907	0.5251	0.4932	0.4346	0.4170	0.4005
	16	1	0.8985	0.7845	0.6801	0.6545	0.5826	0.4767	0.4565	0.4459
	32	1	0.9308	0.7786	0.6835	0.6362	0.5725	0.5352	0.4925	0.4828
MIC-qEGO	2	1	0.6150	0.4811	0.4139	0.3954	0.3633	0.3205	0.3064	0.3002
	4	1	0.5457	0.4260	0.3719	0.3597	0.3404	0.3274	0.3116	0.2946
	8	1	0.6917	0.4933	0.4325	0.3891	0.3557	0.3361	0.3227	0.3104
	16	1	0.8129	0.6301	0.4402	0.4090	0.3861	0.3587	0.3542	0.3517
	32	1	0.8515	0.6946	0.5480	0.5070	0.4679	0.4348	0.4113	0.4031
MACE	2	1	0.8956	0.8450	0.7716	0.7373	0.5952	0.4652	0.4065	0.3909
	4	1	0.8006	0.7275	0.6426	0.5747	0.4597	0.3337	0.2897	0.2716
	8	1	0.7724	0.6807	0.5366	0.4520	0.4049	0.3254	0.2934	0.2797
	16	1	0.7291	0.6141	0.4300	0.3574	0.3313	0.2622	0.2553	0.2428
	32	1	0.6020	0.4052	0.3430	0.3261	0.2847	0.2480	0.2251	0.2034
BSP-EGO	2	1	0.6143	0.4523	0.3743	0.3390	0.3245	0.2911	0.2814	0.2731
	4	1	0.6326	0.4851	0.4433	0.4191	0.3756	0.3413	0.3244	0.3144
	8	1	0.6048	0.4851	0.4206	0.4034	0.3504	0.3155	0.2893	0.2791
	16	1	0.6397	0.4934	0.4199	0.3980	0.3657	0.3310	0.3118	0.2997
	32	1	0.7432	0.6580	0.5540	0.5068	0.4634	0.3931	0.3834	0.3653
ℓBSP-EGO	2	1	0.5083	0.4307	0.4195	0.4175	0.4142	0.4120	0.4119	0.4111
	4	1	0.4458	0.3772	0.3404	0.3397	0.3397	0.3397	0.3397	0.3397
	8	1	0.4817	0.3511	0.3085	0.2963	0.2838	0.2838	0.2838	0.2838
	16	1	0.5098	0.3419	0.2694	0.2512	0.2469	0.2182	0.2182	0.2182
	32	1	0.6512	0.4777	0.3197	0.2806	0.2297	0.2085	0.2030	0.1889
TuRBO	2	1	0.3699	0.3318	0.3230	0.3168	0.2995	0.2979	0.2979	0.2979
	4	1	0.3929	0.3419	0.3086	0.3080	0.3080	0.2877	0.2876	0.2779
	8	1	0.3732	0.3367	0.3343	0.3343	0.3343	0.3258	0.3202	0.3187
	16	1	0.3383	0.2953	0.2842	0.2842	0.2771	0.2727	0.2722	0.2722
	32	1	0.3365	0.2627	0.2616	0.2563	0.2550	0.2540	0.2540	0.2481

Table C.5: Scaled outcome for the **6d-Rastrigin** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.0869	0.0353	0.0228	0.0182	0.0135	0.0076	0.0056	0.0043
	4	1	0.1005	0.0510	0.0281	0.0207	0.0137	0.0080	0.0055	0.0045
	8	1	0.0949	0.0408	0.0267	0.0228	0.0150	0.0103	0.0077	0.0066
	16	1	0.1037	0.0540	0.0335	0.0260	0.0171	0.0119	0.0093	0.0092
	32	1	0.1160	0.0548	0.0377	0.0312	0.0255	0.0204	0.0138	0.0107
TS-qEGO	2	1	0.3283	0.2085	0.0896	0.0533	0.0281	0.0149	0.0119	0.0096
	4	1	0.2342	0.1110	0.0482	0.0316	0.0204	0.0126	0.0105	0.0093
	8	1	0.1548	0.0756	0.0297	0.0219	0.0158	0.0128	0.0107	0.0103
	16	1	0.0931	0.0443	0.0213	0.0178	0.0148	0.0120	0.0117	0.0113
	32	1	0.0377	0.0287	0.0224	0.0191	0.0167	0.0150	0.0136	0.0132
KB-qEGO	2	1	0.1206	0.0491	0.0172	0.0113	0.0051	0.0022	0.0014	0.0012
	4	1	0.0958	0.0513	0.0153	0.0085	0.0042	0.0018	0.0014	0.0011
	8	1	0.1159	0.0530	0.0187	0.0092	0.0046	0.0021	0.0013	0.0010
	16	1	0.1292	0.0733	0.0263	0.0165	0.0066	0.0033	0.0022	0.0018
	32	1	0.1629	0.0747	0.0367	0.0180	0.0117	0.0050	0.0035	0.0028
MIC-qEGO	2	1	0.1171	0.0408	0.0165	0.0097	0.0052	0.0019	0.0014	0.0012
	4	1	0.0943	0.0391	0.0128	0.0066	0.0021	0.0008	0.0005	0.0004
	8	1	0.1016	0.0338	0.0130	0.0068	0.0026	0.0010	0.0005	0.0004
	16	1	0.1561	0.0566	0.0205	0.0120	0.0040	0.0018	0.0012	0.0007
	32	1	0.1437	0.0729	0.0333	0.0138	0.0056	0.0020	0.0013	0.0010
MACE	2	1	0.3161	0.2245	0.0784	0.0414	0.0100	0.0037	0.0027	0.0022
	4	1	0.2221	0.1429	0.0236	0.0097	0.0056	0.0021	0.0016	0.0014
	8	1	0.1697	0.0982	0.0168	0.0072	0.0028	0.0015	0.0010	0.0010
	16	1	0.1364	0.0648	0.0141	0.0045	0.0024	0.0013	0.0012	0.0012
	32	1	0.0915	0.0417	0.0167	0.0102	0.0054	0.0013	0.0008	0.0006
BSP-EGO	2	1	0.1801	0.0674	0.0169	0.0104	0.0047	0.0018	0.0013	0.0007
	4	1	0.0954	0.0406	0.0115	0.0071	0.0037	0.0014	0.0009	0.0007
	8	1	0.1104	0.0303	0.0102	0.0051	0.0023	0.0009	0.0007	0.0006
	16	1	0.0997	0.0319	0.0104	0.0062	0.0027	0.0009	0.0004	0.0004
	32	1	0.1406	0.0585	0.0198	0.0161	0.0087	0.0035	0.0024	0.0017
ℓBSP-EGO	2	1	0.0902	0.0210	0.0032	0.0016	0.0005	0.0002	0.0001	0.0001
	4	1	0.0400	0.0058	0.0005	0.0003	0.0002	0.0001	0.0000	0.0000
	8	1	0.0196	0.0040	0.0005	0.0002	0.0001	0.0000	0.0000	0.0000
	16	1	0.0184	0.0040	0.0007	0.0003	0.0002	0.0001	0.0001	0.0001
	32	1	0.0274	0.0114	0.0037	0.0012	0.0004	0.0002	0.0001	0.0001
TuRBO	2	1	0.0192	0.0032	0.0010	0.0005	0.0004	0.0003	0.0003	0.0003
	4	1	0.0070	0.0013	0.0009	0.0008	0.0005	0.0004	0.0004	0.0003
	8	1	0.0043	0.0024	0.0018	0.0013	0.0009	0.0004	0.0003	0.0003
	16	1	0.0046	0.0014	0.0008	0.0007	0.0005	0.0003	0.0003	0.0003
	32	1	0.0146	0.0027	0.0008	0.0004	0.0003	0.0003	0.0003	0.0003

Table C.6: Scaled outcome for the **Rosenbrock-12d** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.0265	0.0202	0.0162	0.0160	0.0151	0.0143	0.0124	0.0118
	4	1	0.0281	0.0254	0.0216	0.0208	0.0170	0.0163	0.0162	0.0157
	8	1	0.0319	0.0300	0.0300	0.0278	0.0261	0.0223	0.0219	0.0215
	16	1	0.0360	0.0360	0.0360	0.0360	0.0351	0.0336	0.0319	0.0305
	32	1	0.0340	0.0340	0.0340	0.0340	0.0340	0.0340	0.0340	0.0340
TS-qEGO	2	1	0.1915	0.1588	0.1057	0.0957	0.0846	0.0734	0.0694	0.0660
	4	1	0.1870	0.1483	0.1055	0.0963	0.0793	0.0692	0.0646	0.0617
	8	1	0.1489	0.1351	0.1045	0.0960	0.0914	0.0849	0.0774	0.0746
	16	1	0.1481	0.1213	0.1108	0.0999	0.0928	0.0868	0.0837	0.0817
	32	1	0.1568	0.1268	0.1099	0.0984	0.0925	0.0863	0.0863	0.0843
KB-qEGO	2	1	0.0303	0.0235	0.0147	0.0129	0.0092	0.0078	0.0073	0.0069
	4	1	0.0287	0.0247	0.0153	0.0138	0.0100	0.0084	0.0082	0.0079
	8	1	0.0286	0.0231	0.0169	0.0121	0.0099	0.0084	0.0080	0.0077
	16	1	0.0347	0.0306	0.0224	0.0159	0.0132	0.0100	0.0095	0.0092
	32	1	0.0368	0.0340	0.0227	0.0196	0.0129	0.0107	0.0095	0.0093
MIC-qEGO	2	1	0.0299	0.0247	0.0141	0.0116	0.0098	0.0072	0.0068	0.0063
	4	1	0.0316	0.0220	0.0147	0.0115	0.0087	0.0070	0.0051	0.0045
	8	1	0.0299	0.0238	0.0172	0.0155	0.0115	0.0079	0.0068	0.0063
	16	1	0.0377	0.0296	0.0196	0.0153	0.0113	0.0096	0.0087	0.0083
	32	1	0.0419	0.0312	0.0261	0.0195	0.0140	0.0098	0.0085	0.0081
MACE	2	1	0.1109	0.0704	0.0321	0.0266	0.0167	0.0100	0.0068	0.0058
	4	1	0.0477	0.0366	0.0285	0.0243	0.0173	0.0103	0.0068	0.0052
	8	1	0.0457	0.0385	0.0345	0.0307	0.0234	0.0136	0.0088	0.0078
	16	1	0.0352	0.0334	0.0262	0.0224	0.0190	0.0159	0.0145	0.0119
	32	1	0.0296	0.0278	0.0269	0.0244	0.0190	0.0170	0.0139	0.0130
BSP-EGO	2	1	0.0370	0.0240	0.0157	0.0112	0.0080	0.0060	0.0050	0.0045
	4	1	0.0383	0.0262	0.0130	0.0101	0.0067	0.0045	0.0031	0.0026
	8	1	0.0362	0.0261	0.0117	0.0102	0.0065	0.0042	0.0030	0.0021
	16	1	0.0394	0.0296	0.0212	0.0115	0.0084	0.0049	0.0037	0.0032
	32	1	0.0465	0.0430	0.0340	0.0275	0.0210	0.0123	0.0110	0.0098
ℓBSP-EGO	2	1	0.0376	0.0311	0.0108	0.0045	0.0016	0.0009	0.0008	0.0007
	4	1	0.0355	0.0168	0.0035	0.0022	0.0009	0.0006	0.0004	0.0004
	8	1	0.0334	0.0124	0.0054	0.0035	0.0015	0.0008	0.0006	0.0004
	16	1	0.0365	0.0165	0.0066	0.0052	0.0043	0.0026	0.0018	0.0016
	32	1	0.0445	0.0374	0.0210	0.0111	0.0059	0.0046	0.0035	0.0030
TuRBO	2	1	0.0111	0.0048	0.0024	0.0018	0.0015	0.0011	0.0009	0.0008
	4	1	0.0064	0.0025	0.0015	0.0013	0.0009	0.0006	0.0005	0.0005
	8	1	0.0048	0.0018	0.0010	0.0008	0.0006	0.0004	0.0004	0.0004
	16	1	0.0038	0.0014	0.0007	0.0007	0.0005	0.0004	0.0004	0.0003
	32	1	0.0096	0.0040	0.0013	0.0009	0.0007	0.0004	0.0004	0.0003

Table C.7: Scaled outcome for the **12d-Alpine02** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.9980	0.9979	0.9978	0.9973	0.9971	0.9969	0.9969	0.9965
	4	1	0.9993	0.9992	0.9987	0.9986	0.9983	0.9977	0.9977	0.9976
	8	1	0.9982	0.9975	0.9954	0.9950	0.9939	0.9925	0.9919	0.9919
	16	1	0.9976	0.9969	0.9961	0.9957	0.9954	0.9930	0.9929	0.9929
	32	1	0.9982	0.9973	0.9972	0.9971	0.9965	0.9962	0.9961	0.9961
TS-qEGO	2	1	0.9997	0.9997	0.9995	0.9986	0.9983	0.9981	0.9967	0.9964
	4	1	0.9971	0.9969	0.9966	0.9958	0.9952	0.9938	0.9938	0.9937
	8	1	0.9973	0.9943	0.9930	0.9929	0.9915	0.9909	0.9909	0.9909
	16	1	0.9987	0.9980	0.9979	0.9968	0.9967	0.9958	0.9953	0.9945
	32	1	0.9948	0.9941	0.9920	0.9910	0.9909	0.9904	0.9904	0.9904
KB-qEGO	2	1	0.9982	0.9955	0.9952	0.9952	0.9945	0.9943	0.9924	0.9922
	4	1	0.9997	0.9996	0.9995	0.9995	0.9991	0.9989	0.9989	0.9988
	8	1	0.9997	0.9987	0.9986	0.9985	0.9977	0.9977	0.9976	0.9976
	16	1	0.9995	0.9995	0.9992	0.9992	0.9991	0.9991	0.9991	0.9991
	32	1	0.9992	0.9990	0.9980	0.9980	0.9978	0.9972	0.9972	0.9972
MIC-qEGO	2	1	0.9987	0.9967	0.9957	0.9955	0.9955	0.9952	0.9952	0.9946
	4	1	0.9926	0.9839	0.9776	0.9744	0.9639	0.9426	0.9338	0.9293
	8	1	0.9856	0.9729	0.9555	0.9462	0.9324	0.8987	0.8831	0.8773
	16	1	0.9932	0.9818	0.9649	0.9528	0.9317	0.8996	0.8709	0.8595
	32	1	0.9976	0.9932	0.9832	0.9757	0.9634	0.9535	0.9491	0.9436
MACE	2	1	0.9990	0.9948	0.9917	0.9890	0.9847	0.9685	0.9379	0.9102
	4	1	0.9988	0.9964	0.9918	0.9892	0.9805	0.9663	0.9602	0.9517
	8	1	0.9985	0.9904	0.9847	0.9773	0.9566	0.9242	0.8673	0.8554
	16	1	0.9953	0.9913	0.9690	0.9542	0.9204	0.8458	0.8072	0.7657
	32	1	0.9837	0.9650	0.9393	0.9238	0.8792	0.8117	0.7658	0.7117
BSP-EGO	2	1	0.9969	0.9950	0.9927	0.9927	0.9925	0.9915	0.9915	0.9915
	4	1	0.9956	0.9903	0.9900	0.9880	0.9848	0.9830	0.9828	0.9802
	8	1	0.9946	0.9926	0.9867	0.9858	0.9854	0.9850	0.9844	0.9831
	16	1	0.9884	0.9868	0.9858	0.9858	0.9857	0.9852	0.9848	0.9822
	32	1	0.9869	0.9859	0.9859	0.9859	0.9859	0.9854	0.9854	0.9841
ℓBSP-EGO	2	1	0.9757	0.9314	0.9158	0.9124	0.9123	0.9113	0.9105	0.9101
	4	1	0.9421	0.8959	0.8838	0.8828	0.8826	0.8819	0.8766	0.8721
	8	1	0.9191	0.8734	0.8543	0.8419	0.8373	0.8278	0.8234	0.8201
	16	1	0.9102	0.8832	0.8521	0.8383	0.8212	0.7930	0.7784	0.7482
	32	1	0.9404	0.8826	0.8507	0.8419	0.8334	0.8240	0.8107	0.8078
TuRBO	2	1	0.9903	0.9634	0.8917	0.8642	0.8626	0.8626	0.8626	0.8626
	4	1	0.9813	0.9070	0.8578	0.8422	0.8266	0.8266	0.8266	0.8266
	8	1	0.9895	0.9307	0.8822	0.8807	0.8775	0.8771	0.8771	0.8771
	16	1	0.9769	0.9244	0.8876	0.8802	0.8769	0.8769	0.8769	0.8769
	32	1	0.9856	0.9485	0.8946	0.8666	0.8534	0.8409	0.8389	0.8387

Table C.8: Scaled outcome for the **12d-Ackley** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

		0	30	60	120	180	300	600	900	1200
Method	q									
qEGO	2	1	0.7801	0.7763	0.7718	0.7598	0.7480	0.7480	0.7480	0.7480
	4	1	0.7626	0.7540	0.7540	0.7540	0.7540	0.7540	0.7537	0.7381
	8	1	0.7742	0.7742	0.7742	0.7197	0.7197	0.7197	0.7197	0.7197
	16	1	0.7707	0.7707	0.7448	0.7448	0.7448	0.7448	0.7226	0.7208
	32	1	0.7442	0.7442	0.7342	0.7193	0.7193	0.6997	0.6997	0.6997
TS-qEGO	2	1	0.8339	0.8009	0.7686	0.7448	0.7297	0.7090	0.7002	0.6979
	4	1	0.8181	0.7790	0.7485	0.7428	0.7359	0.7204	0.7202	0.7078
	8	1	0.7918	0.7797	0.7690	0.7601	0.7498	0.7393	0.7347	0.7266
	16	1	0.8299	0.8100	0.7869	0.7814	0.7669	0.7617	0.7519	0.7427
	32	1	0.8256	0.8175	0.7753	0.7694	0.7639	0.7512	0.7419	0.7412
KB-qEGO	2	1	0.7181	0.7177	0.6982	0.6744	0.6439	0.6432	0.6432	0.6155
	4	1	0.7497	0.7322	0.6835	0.6642	0.6128	0.5882	0.5882	0.5882
	8	1	0.7664	0.7356	0.7322	0.7293	0.7080	0.6826	0.6826	0.6826
	16	1	0.7952	0.7653	0.7479	0.7231	0.7049	0.6777	0.6777	0.6777
	32	1	0.7892	0.7892	0.7734	0.7728	0.7699	0.7699	0.7679	0.7454
MIC-qEGO	2	1	0.7111	0.6495	0.6266	0.6001	0.5815	0.5815	0.5815	0.5815
	4	1	0.4549	0.3314	0.2711	0.2600	0.2551	0.2479	0.2449	0.2424
	8	1	0.4425	0.3215	0.2872	0.2756	0.2639	0.2450	0.2372	0.2250
	16	1	0.6035	0.3847	0.3352	0.3207	0.3030	0.2912	0.2803	0.2747
	32	1	0.7708	0.5316	0.3663	0.3249	0.3217	0.3120	0.3011	0.2967
MACE	2	1	0.7949	0.7194	0.5619	0.4024	0.3027	0.2123	0.1695	0.1568
	4	1	0.7000	0.5644	0.3816	0.3086	0.2317	0.1584	0.1467	0.1451
	8	1	0.5784	0.3632	0.2516	0.2171	0.1753	0.1436	0.1430	0.1430
	16	1	0.4844	0.3004	0.2099	0.1812	0.1546	0.1378	0.1290	0.1288
	32	1	0.3979	0.2774	0.2076	0.1816	0.1570	0.1378	0.1323	0.1297
BSP-EGO	2	1	0.5544	0.4392	0.4193	0.4193	0.3980	0.3980	0.3980	0.3980
	4	1	0.3605	0.3605	0.3323	0.3323	0.3323	0.3149	0.3149	0.3149
	8	1	0.3404	0.2687	0.2040	0.2013	0.1864	0.1864	0.1862	0.1862
	16	1	0.3481	0.3081	0.2719	0.2413	0.2368	0.1857	0.1774	0.1741
	32	1	0.5317	0.4094	0.3630	0.3623	0.3502	0.2755	0.2603	0.2588
ℓBSP-EGO	2	1	0.2416	0.1492	0.1140	0.0902	0.0625	0.0563	0.0557	0.0549
	4	1	0.1298	0.0625	0.0258	0.0088	0.0012	0.0004	0.0003	0.0002
	8	1	0.1536	0.0815	0.0394	0.0268	0.0109	0.0031	0.0020	0.0018
	16	1	0.1900	0.0998	0.0619	0.0443	0.0305	0.0190	0.0127	0.0114
	32	1	0.3266	0.1873	0.1239	0.1016	0.0733	0.0555	0.0478	0.0466
TuRBO	2	1	0.2976	0.1917	0.1496	0.1325	0.1117	0.1038	0.0975	0.0926
	4	1	0.2181	0.1614	0.1310	0.1184	0.1118	0.1052	0.1006	0.0953
	8	1	0.1992	0.1444	0.1256	0.1214	0.1122	0.1015	0.1002	0.0972
	16	1	0.2107	0.1581	0.1399	0.1308	0.1233	0.1170	0.1143	0.1112
	32	1	0.3177	0.1715	0.1216	0.1100	0.1084	0.0993	0.0948	0.0938

Table C.9: Scaled outcome for the **12d-Schwefel** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

Method	q	0	30	60	120	180	300	600	900	1200
qEGO	2	1	0.9221	0.8550	0.7691	0.7498	0.7333	0.7067	0.6815	0.6695
	4	1	0.9163	0.8756	0.8201	0.7888	0.7490	0.7101	0.6895	0.6844
	8	1	0.9439	0.8995	0.8557	0.8377	0.8267	0.8007	0.7708	0.7446
	16	1	0.9346	0.9124	0.8767	0.8615	0.8429	0.7963	0.7700	0.7659
	32	1	0.9212	0.9138	0.8920	0.8657	0.8489	0.8268	0.8263	0.8193
TS-qEGO	2	1	0.9864	0.9738	0.9433	0.9116	0.9011	0.8742	0.8682	0.8632
	4	1	0.9755	0.9514	0.9438	0.9288	0.9020	0.8724	0.8710	0.8592
	8	1	0.9702	0.9457	0.9231	0.9050	0.8982	0.8649	0.8581	0.8567
	16	1	0.9597	0.9428	0.9102	0.8995	0.8864	0.8681	0.8473	0.8462
	32	1	0.8972	0.8947	0.8718	0.8571	0.8505	0.8371	0.8284	0.8242
KB-qEGO	2	1	0.9452	0.8674	0.7991	0.7798	0.7389	0.7082	0.6880	0.6802
	4	1	0.9545	0.9135	0.8471	0.8101	0.7594	0.7127	0.7046	0.6957
	8	1	0.9777	0.9633	0.9025	0.8743	0.8400	0.8101	0.7826	0.7675
	16	1	0.9884	0.9477	0.9306	0.9114	0.8884	0.8597	0.8461	0.8326
	32	1	0.9930	0.9800	0.9737	0.9508	0.9284	0.8900	0.8603	0.8504
MIC-qEGO	2	1	0.9689	0.8842	0.8301	0.7926	0.7597	0.7223	0.7114	0.7035
	4	1	0.9280	0.8509	0.7683	0.7185	0.6805	0.6273	0.5925	0.5767
	8	1	0.9296	0.8440	0.7384	0.6940	0.6563	0.6115	0.5923	0.5885
	16	1	0.9670	0.9254	0.8518	0.8025	0.7517	0.7040	0.6801	0.6690
	32	1	0.9958	0.9679	0.9475	0.9035	0.8649	0.8077	0.7846	0.7570
MACE	2	1	0.9831	0.9744	0.8983	0.8645	0.8278	0.7643	0.6916	0.6650
	4	1	0.9713	0.9603	0.8851	0.8664	0.8069	0.7453	0.7031	0.6923
	8	1	0.9449	0.8860	0.8440	0.8045	0.7679	0.7073	0.6694	0.6589
	16	1	0.9207	0.8447	0.7879	0.7626	0.6963	0.6042	0.5684	0.5201
	32	1	0.9028	0.8079	0.7408	0.6578	0.6233	0.5089	0.4549	0.4307
BSP-EGO	2	1	0.9174	0.8115	0.7013	0.6393	0.5822	0.5639	0.5451	0.5436
	4	1	0.9264	0.8460	0.7055	0.6382	0.5864	0.5389	0.5266	0.5193
	8	1	0.8921	0.7849	0.7223	0.6806	0.5945	0.5357	0.5118	0.5027
	16	1	0.9108	0.8420	0.7494	0.7070	0.6643	0.6153	0.5791	0.5681
	32	1	0.9196	0.8952	0.8428	0.8049	0.7877	0.7264	0.7023	0.6946
ℓBSP-EGO	2	1	0.8229	0.6891	0.5393	0.4842	0.4678	0.4665	0.4631	0.4628
	4	1	0.7708	0.5817	0.4364	0.3877	0.3645	0.3630	0.3630	0.3630
	8	1	0.7870	0.6452	0.4908	0.3980	0.3138	0.2754	0.2704	0.2704
	16	1	0.8038	0.6580	0.5759	0.5328	0.4441	0.3476	0.3211	0.3019
	32	1	0.8618	0.7636	0.6532	0.6165	0.5612	0.4775	0.4251	0.3916
TuRBO	2	1	0.8259	0.6683	0.4905	0.4562	0.4244	0.4186	0.4167	0.4083
	4	1	0.7759	0.5913	0.5074	0.4777	0.4529	0.4467	0.4467	0.4467
	8	1	0.7937	0.6160	0.5253	0.5107	0.5050	0.5002	0.4993	0.4993
	16	1	0.6402	0.5055	0.4739	0.4704	0.4698	0.4697	0.4697	0.4697
	32	1	0.8296	0.6503	0.5317	0.4838	0.4672	0.4625	0.4624	0.4624

Table C.10: Scaled outcome for the **12d-Rastrigin** function averaged over 20 repetitions. Values close to 1 indicate small improvement compared to initial sampling while values approaching the theoretical optimal value are close to 0 and highlighted in blue shades.

		0	30	60	120	180	300	600	900	1200
Method	q									
qEGO	2	1	0.6557	0.6442	0.6224	0.6147	0.6087	0.5948	0.5937	0.5846
	4	1	0.6556	0.6260	0.6123	0.5980	0.5839	0.5821	0.5806	0.5806
	8	1	0.6219	0.5908	0.5787	0.5753	0.5753	0.5753	0.5591	0.5591
	16	1	0.6426	0.6280	0.6280	0.6192	0.5900	0.5756	0.5716	0.5716
	32	1	0.6220	0.6086	0.5935	0.5935	0.5841	0.5832	0.5832	0.5832
TS-qEGO	2	1	0.7964	0.7566	0.7228	0.6989	0.6717	0.6373	0.6015	0.5960
	4	1	0.7688	0.7234	0.6677	0.6577	0.6479	0.6275	0.6161	0.6154
	8	1	0.7209	0.7013	0.6612	0.6463	0.6202	0.6104	0.6039	0.5989
	16	1	0.6933	0.6731	0.6520	0.6464	0.6360	0.6084	0.5963	0.5946
	32	1	0.6663	0.6409	0.6208	0.6149	0.6025	0.5936	0.5871	0.5794
KB-qEGO	2	1	0.6987	0.6635	0.6199	0.6158	0.5986	0.5915	0.5896	0.5896
	4	1	0.7011	0.6631	0.6385	0.6300	0.6172	0.6095	0.6095	0.6095
	8	1	0.7253	0.6863	0.6547	0.6445	0.6242	0.6177	0.6063	0.6004
	16	1	0.7249	0.6843	0.6676	0.6522	0.6486	0.6073	0.6007	0.6004
	32	1	0.7335	0.7287	0.7178	0.7023	0.6598	0.6507	0.6481	0.6386
MIC-qEGO	2	1	0.6795	0.6454	0.6235	0.6225	0.6085	0.6015	0.5992	0.5958
	4	1	0.6516	0.6052	0.5762	0.5393	0.5144	0.4895	0.4799	0.4799
	8	1	0.6812	0.6418	0.5982	0.5926	0.5656	0.5308	0.5018	0.4959
	16	1	0.6956	0.6404	0.6164	0.6000	0.5840	0.5571	0.5483	0.5447
	32	1	0.7124	0.6708	0.6449	0.6392	0.6118	0.5929	0.5888	0.5752
MACE	2	1	0.7205	0.6791	0.6338	0.6193	0.6056	0.5934	0.5903	0.5903
	4	1	0.7059	0.6757	0.6651	0.6261	0.6081	0.5869	0.5725	0.5592
	8	1	0.7019	0.6559	0.6331	0.6261	0.6065	0.5947	0.5947	0.5903
	16	1	0.6755	0.6422	0.6099	0.5915	0.5767	0.5649	0.5604	0.5578
	32	1	0.6702	0.6311	0.6141	0.5902	0.5591	0.5251	0.5172	0.5167
BSP-EGO	2	1	0.6729	0.6211	0.6030	0.5706	0.5506	0.5166	0.4904	0.4852
	4	1	0.6944	0.6575	0.6214	0.5840	0.5599	0.5232	0.4883	0.4649
	8	1	0.6556	0.6048	0.5813	0.5575	0.5482	0.5114	0.4855	0.4820
	16	1	0.6840	0.6129	0.5737	0.5478	0.5448	0.4964	0.4867	0.4687
	32	1	0.7072	0.6687	0.6359	0.5954	0.5646	0.5392	0.5137	0.5019
ℓBSP-EGO	2	1	0.5959	0.4028	0.2531	0.1577	0.1206	0.1174	0.1173	0.1173
	4	1	0.5696	0.3672	0.2189	0.1586	0.1030	0.0896	0.0896	0.0896
	8	1	0.4451	0.3160	0.1823	0.1401	0.1046	0.0815	0.0770	0.0757
	16	1	0.4877	0.3538	0.2060	0.1645	0.1282	0.1011	0.0848	0.0746
	32	1	0.5735	0.4660	0.3735	0.2954	0.2423	0.1862	0.1679	0.1624
TuRBO	2	1	0.5331	0.3825	0.3131	0.2753	0.2360	0.1908	0.1757	0.1610
	4	1	0.4739	0.4088	0.2954	0.2554	0.2056	0.1701	0.1573	0.1527
	8	1	0.3705	0.2988	0.2250	0.1840	0.1597	0.1347	0.1306	0.1277
	16	1	0.3812	0.3008	0.2268	0.2082	0.1906	0.1655	0.1622	0.1569
	32	1	0.5059	0.3700	0.2550	0.2211	0.1892	0.1625	0.1622	0.1610