2023 | Faculty of Business Economics

UHASSELT
KNOWLEDGE IN ACTION

GHENT UNIVERSITY

Doctoral dissertation submitted to obtain the degrees of
- Discipline UHasselt: Doctor of Business Economics
- Discipline UGent: Doctor of Computer Science Engineering

Alejandro Morales Hernández

DOCTORAL DISSERTATION

Essays on Machine Learning: Advances in Forecasting and Optimization

Promoters: Prof. Dr Inneke Van Nieuwenhuyse | Hasselt University
Prof. Dr Ivo Couckuyt | Ghent University

Co-promoter: Prof. Dr Koenraad Vanhoof | Hasselt University

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Hasselt University
Martelarenlaan 42 |BE-3500 Hasselt

2023 | Faculty of Business Economics

UHASSELT
KNOWLEDGE IN ACTION

GHENT UNIVERSITY

Doctoral dissertation submitted to obtain the degrees of
- Discipline UHasselt: Doctor of Business Economics
- Discipline UGent: Doctor of Computer Science Engineering

Alejandro Morales Hernández

DOCTORAL DISSERTATION

Essays on Machine Learning: Advances in Forecasting and Optimization

DSI DATA SCIENCE INSTITUTE
UHASSELT

Promoters: Prof. Dr Inneke Van Nieuwenhuyse | Hasselt University
Prof. Dr Ivo Couckuyt | Ghent University

Co-promoter: Prof. Dr Koenraad Vanhoof | Hasselt University

D/2023/2451/43


Beta
Research School for Operations
Management and Logistics

Essays on Machine Learning: Advances in Forecasting and Optimization

Alejandro Morales Hernández

ISBN
NUR: 984
Deposit number: D/2023/2451/43

This thesis is part of the PhD thesis series of the Beta Research School for Operations Management and Logistics in which research groups participate of CWI, Eindhoven University of Technology, Ghent University, Hasselt University, KU Leuven, Maastricht University, Tilburg University, University of Antwerp, University of Twente, VU Amsterdam, VU Brussels, and Wageningen University and Research

This research has been supported by the Flanders Artificial Intelligence Research Program (FLAIR), funded by the Flemish government.

Members of the Examination Board

Chair

Honorary Prof. Dr. Piet Pauwels, PhD, Hasselt University

Other members entitled to vote

Dr. Ir. Jolan Wauters, PhD, Ghent University
Prof. Dirk Deschrijver, PhD, Ghent University
Prof. Gonzalo Nápoles Ruiz, PhD, Tilburg University
Prof. Agnieszka Jastrzebska, PhD, Warsaw University of Technology
Prof. Isel Grau García, PhD, Technische Universiteit Eindhoven
Prof. Dirk Valkenborg, PhD, Hasselt University

Supervisors

Prof. Inneke Van Nieuwenhuyse, PhD, Hasselt University
Prof. Koenraad Vanhoof, PhD, Hasselt University
Prof. Ivo Couckuyt, PhD, Ghent University

"*There is nothing to be feared from a body, Harry, any more than there is anything to be feared from the darkness. Lord Voldemort, who of course secretly fears both, disagrees. But once again he reveals his own lack of wisdom.* ***It is the unknown we fear when we look upon death and darkness****, nothing more.*

Albus Percival Wulfric Brian Dumbledore
Harry Potter and the Half-Blood Prince, Chapter 26

Acknowledgements

A WOODEN BENCH has just appeared around the turn of the road. Thank god because walking on these dunes in Hechtel-Eksel has not been easy. One would think that going for a walk every weekend prepares you for this type of terrain, but no. It doesn't matter having done more than 70 hiking trails, there are some places that take your breath away, literally. This bench has seen better days but my feet are killing me. So, let's take a break. An old oak is some meters behind me. A gentle breeze of early autumn plays with some yellow leaves. Soon the ground will be covered with them and winter will begin.

Oh boy! What a journey! Almost four years already. It seems like yesterday when Gonzalo suggested this opportunity to me. 2019. The year of big changes. I remember that I was preparing to go to Russia for 10 months. Fortunately, that changed. Because I wouldn't be here if it wasn't for you, and for being you and Isel like family all these years, my most sincere and deep thanks.

A fine drizzle is beginning to fall now. Classic Belgian weather. Nothing terrible, and definitely "much better" than the suffocating heat of Cuba. It was a particularly hot afternoon when I told my mom about the possibility of leaving the country. She looked at me and said, "You can count on me for everything". She has been always like that. Strong, independent, determined. Despite the more than 7000 km separating me from my family, we have always been close. No one could have prepared us for the difficult years that were to come. Yet, we survived. We always do. Thanks.

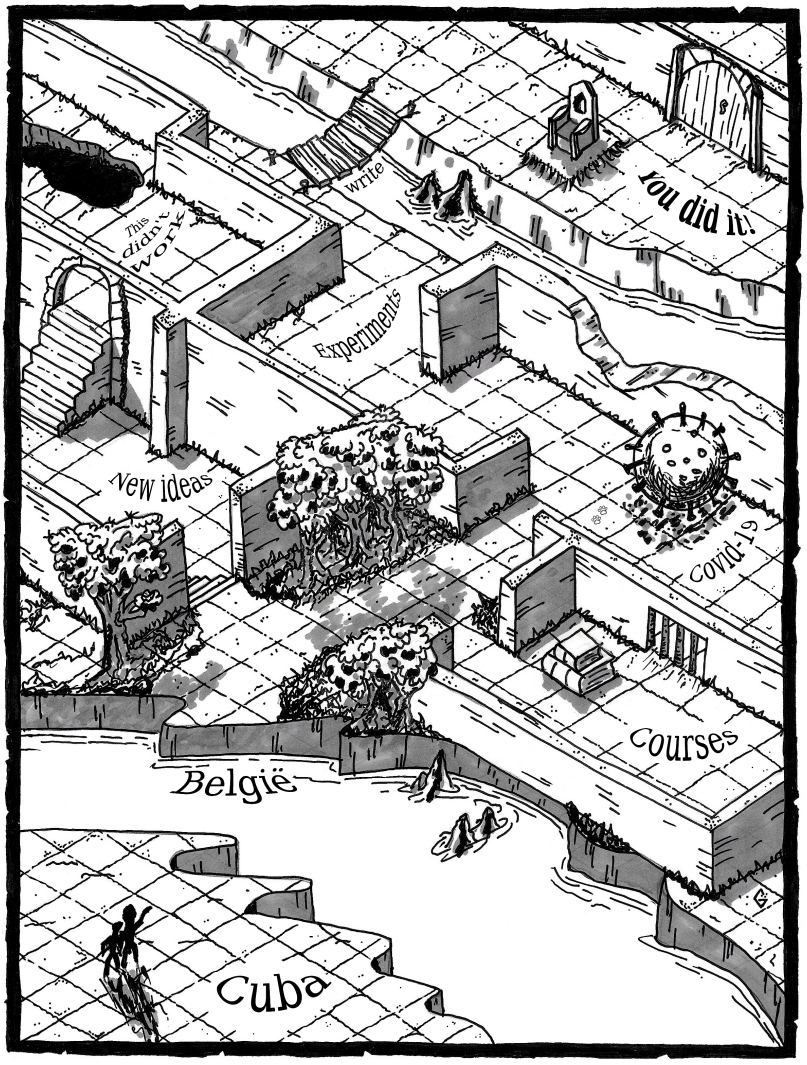
A notification appeared on my phone. Something about a deadline approaching. If I have learned anything over the years, it's that being a PhD student doesn't stop when you are out of the office. Oh shoot! It's that draft for Inneke I still have to finish. I better take care of it asap. Sometimes we remember people for small details that may go unnoticed. Patience, for instance, is what I got for Inneke. Thank you for your willingness to listen, understand and guide this "timid", sometimes stubborn, sometimes clueless Cuban. You have been of great support and motivation so that today I end this adventure.

All right, break time's over. I should hurry up to take the next bus to Hasselt. I still have some kilometers to walk before reaching the bus stop, and the rain is getting stronger. As I take the cape out of the backpack, I think of all the people who have been with me over the years. That kind soul that is Sasan. Thanks for being my friend. Thanks to my other supervisors, Koen and Ivo, for trusting that I could wear bigger shoes than I was entitled to and helping me with all kinds of bureaucratic procedures. Of course my gratitude to all my colleagues and friends; to Sebastian, because I could not have had all the experimental results without your laptop; to Gert, because this document would have been less elegant without his help; to Lilo, because we cried together; and GG why not, because fate brought

us together to achieve world domination (her idea, not mine).

The forest is very quiet now. Before, you could hear the chirping of some birds, but now the rain has silenced them. The road begins to head slightly towards the town and you can already see the back yard of some houses. There is still a lot to explore, to live. This is not the end, for sure. Another day will come tomorrow and a new adventure with it. Will we be ready?

Alejandro Morales
Hasselt, April 2023



Contents

List of Figures

List of Tables

Symbols and notations

Vectors are in bold type and matrices are capitalized with a subscript indicating their dimension. If needed, the letter is replaced by an equation between box brackets to indicate how the elements are obtained.

$f(\cdot)$	=	The true function, which is assumed to be unknown
$G(\cdot)$	=	Function representing an inequality constraint
$H(\cdot)$	=	Function representing an equality constraint
p	=	Number of inequality constraints
q	=	Number of equality constraints
$\mathbf{x}$	=	Input vector
d	=	Dimension of the input search space or number of variables in a multivariate time series
$\mathcal{X}$	=	Configuration space domain
n	=	Number of observed input configurations
T	=	Number of observations in a univariate time series
$\mathbf{X}$	=	Set of observed input configurations, $\mathbf{X}_{n \times d}$. For multivariate time series, $\mathbf{X}_{d \times T}$ is a sequence of d variables observed T times
m	=	Number of unknown functions to model. In multi-objective optimization $1 < m \leq 3$
$\mathbf{Y}$	=	Set of true function values, $\mathbf{Y}_{n \times m}$
$\mathbf{x}^{(i)}$	=	Configuration i in $\mathbf{X}$, or i-th time series in the multivariate time series $\mathbf{X}$
$x_j^{(i)}$	=	Input parameter j of i-th configuration in $\mathbf{X}$, or j-th variable observed at time i
$\mathbf{y}^{(i)}$	=	True function value of i-th configuration in $\mathbf{Y}$
$y_j^{(i)}$	=	True value of the j-th function for configuration i in $\mathbf{Y}$
r	=	Number of simulation replications in a noisy problem
ε_i	=	Noise in function value i
$\widetilde{y}_j^{(r,i)}$	=	Observed j-th function value in the r-th simulation replication of i-th point $\widetilde{y}_j^{(r,i)} = y_j^i + \varepsilon^{(i)}$
$\overline{y}$	=	Function value estimate $\overline{y} = \frac{\sum_{i=1}^{r} \widetilde{y}}{r}$
$k(\cdot, \cdot)$	=	Kernel function

$\widehat{y}_{\_}(\cdot)$	=	Surrogate prediction. Underscore is replaced by s or o to indicate the function prediction of a stochastic or deterministic surrogate that models an optimization objective. Alternatively, a subscript c refers to the function prediction of a deterministic surrogate for a constraint function. Although constraints can also present noise, we only used a deterministic surrogate, given the nature of the only constraint considered in this research (Chapter 5).
$\widehat{s}^2_{\_}(\cdot)$	=	Variance estimate of $\widehat{y}_{\_}(\cdot)$ or surrogate prediction uncertainty. Underscore is replaced by s or o to indicate the function prediction of a stochastic or deterministic surrogate that models an optimization objective. Alternatively, a subscript c refers to the function prediction of a deterministic surrogate for a constraint function. Although constraints can also present noise, we only used a deterministic surrogate, given the nature of the only constraint considered in this research (Chapter 5).
$\phi(\cdot)$	=	Standard normal density function
$\Phi(\cdot)$	=	Standard normal distribution function
$l(\cdot)$	=	Density function estimated in the input space using the configurations associated to good function values
$g(\cdot)$	=	Density function estimated in the input space using the configurations associated to bad function values
$Z_{\boldsymbol{\lambda}}$	=	Augmented Tchebycheff scalarization function using weights $\boldsymbol{\lambda}$
R	=	If NOW is the current time, R is the number of past observations until NOW
L	=	If NOW is the current time, L is the number of future observations after NOW

Abstract

Machine Learning (ML) has become a critical tool in solving complex problems. However, there are some challenges when applying ML to real-world problems, including the need for large amounts of high-quality data, the potential for errors in algorithms and data, the interpretability of model decisions, and the high computational and storage costs of training and deploying models. Addressing these challenges requires a multidisciplinary approach, as well as the need for ongoing research and development to improve the accuracy, transparency, and efficiency of ML models.

ML is about learning a model that can predict or classify new data points. To do this, we typically formulate the problem as an optimization problem, where we seek to find the optimal parameters of a model that minimizes a cost function. Additionally, the performance of ML algorithms heavily relies on the choice of their hyperparameters. Hyperparameters in ML are parameters that are set prior to the training process and they cannot be learned directly from the training data. Instead, they are set manually or with the help of optimization methods. Increasing attention is being paid to performance uncertainties in evaluating ML algorithms, as they provide information on the reliability and robustness of the predictions. Overall, a computationally efficient and robust hyperparameter optimization (HPO) method is key because training and evaluating these algorithms can be expensive.

Similar challenges are present in industrial applications where the optimization of process parameters is characterized as being multi-objective, constrained, and uncertain. Traditional evolutionary approaches, such as genetic algorithms, are unsuitable for solving these problems, as they require a prohibitive number of experiments for evaluation. Bayesian optimization-based algorithms are preferred for such expensive problems but, as is the case for HPO, few methods consider multiple noisy objectives and several constraints simultaneously.

Although power forecasting of windmill time series in online learning settings seems to be far from the aforementioned optimization challenges, they both share in common the need to use the available data as efficiently as possible. In our opinion, the amount of data generated by windmill farms make online learning the most viable strategy to follow, requiring retraining the model each time a new batch of data is available. However, updating the model with new information can be expensive when using traditional Recurrent Neural Networks (RNNs).

This thesis aims to develop efficient algorithms to overcome common challenges such as time constraints, data sparsity, and uncertainty in Forecasting and Optimization problems. This suggests that the proposed algorithms need to work smartly with the (uncertain) information they have and provide better optimums/forecasting results than existing ap-

proaches. In this thesis, the term *uncertainty* refers to the phenomena of observing different performance/function values when the same input configuration (hyperparameters, process configuration, etc) is evaluated. In turn, the quality of being efficient is directly related to the smart choice of candidates to evaluate next in the optimization process and the short training/testing times of ML models in forecasting tasks.

In order to accomplish our research goal, this thesis explores the power of Bayesian Optimization and Recurrent Neural Networks to solve expensive and complex problems. A multi-objective HPO algorithm is proposed to handle the uncertainty in performance evaluations of ML algorithms. The approach combines the Multi-objective Tree Parzen Estimators (MOTPE) sampling strategy with a Gaussian Process Regression (GPR) trained with heterogeneous noise. In this way, the algorithm should suggest new points that are likely to be non-dominated, and that are expected to cause the maximum improvement in the scalarized objective function. The proposed algorithm has shown better hypervolume compared to stand-alone multi-objective MOTPE and GPR HPO methods, which are translated into accurate ML algorithms.

Additionally, we implemented a modification to the TPE algorithm for single-objective HPO to account for performance variability without the need for any other models. In contrast to the original TPE, our method considers the uncertainty surrounding the performance evaluations and incorporates weights into the kernel density estimators used to generate the density functions. This enables us to assign a probability to each hyperparameter configuration, indicating that they may be both "*good*" and "*bad*" at the same time. Therefore, the splitting procedure used by the noiseless TPE is no longer needed. The aforementioned probabilities of being "*good*" and "*bad*" use the performance distributions observed in a cross-validation protocol and reflect the influence of each point on the density function estimate used to suggest a new input configuration. This modification proved effective in finding better hyperparameter configurations in terms of classification error of the ML algorithm.

Although BO-based algorithms are preferred to solve expensive problems such as HPO and adhesive bonding processes, few methods consider the optimization of more than one (noisy) objective and several constraints at the same time. In this research, we successfully applied GPR to emulate the objective and constraint functions in the optimization of an adhesive bonding process with a *limited* amount of experimental data. The suggested BO framework succeeded in detecting optimal process settings in a highly efficient way (i.e., requiring a few physical experiments). The difference with respect to evolutionary algorithms is that the experimental design in our approach is guided throughout the search: the Bayesian-based algorithms select infill points based on an (explainable) acquisition function, which is related to the expected merit of the new infill point for optimization. The BO model ensures that the search focuses on infill points that have a high probability of being feasible. Moreover, the GP model used to approximate the objective(s) accounts for the output (heterogenous) noise, whereas the evolutionary algorithms rely simply on the (uncertain) sample means as performance approximations.

Lastly, we designed a pipeline based on the Long Short-term Cognitive Network (LSTCN) to address the problem of data volatility and short processing times in the power forecasting of windmills. The Short-term Cognitive Network blocks that compose an LSTCN process a temporal chunk of data with a fast and deterministic learning rule that makes the algorithm suitable for online learning tasks. The network showed the lowest forecasting errors and training/testing times compared to other state-of-the-art recurrent models.

Samenvatting

Machinaal Leren (ML) is essentieel geworden voor het oplossen van complexe problemen. Er zijn echter een aantal uitdagingen bij de toepassing van ML op echte problemen, zoals de noodzaak voor grote hoeveelheden gegevens van hoge kwaliteit, mogelijke inconsistenties in algoritmen en fouten in het verwerken van de gegevens, alsook een gebrek aan de interpreteerbaarheid van voorspellingen, en de hoge reken- en opslagkosten voor het trainen en inzetten van grote ML modellen. Het aanpakken van deze uitdagingen vereist een multidisciplinaire aanpak met verder onderzoek om de nauwkeurigheid, transparantie en efficiëntie van ML-modellen te verbeteren.

Bij ML gaat het om het leren van een model op basis van een data set. Dit ML model kan voorspellingen doen voor nieuwe data (bv. regressie of classificatiemodellen). Het leren van zulk een ML model is een optimalisatieprobleem, waarbij we zoeken naar de optimale parameters dat een of meerdere kostenfuncties minimaliseert. De prestaties van ML-algoritmen zijn immers sterk afhankelijk van de keuze van hun *hyperparameters*. *Hyperparameters* in ML zijn parameters die voorafgaand aan het trainingsproces worden ingesteld en die niet rechtstreeks (= exact en analytisch) uit de trainingsgegevens kunnen worden geleerd. In plaats daarvan worden ze vaak handmatig of met behulp van optimalisatiemethoden ingesteld. Bijkomend wordt bij de evaluatie van ML-algoritmen steeds meer aandacht besteed aan de onzekerheidskwantificatie, omdat dit informatie verschaft over de betrouwbaarheid en robuustheid van de voorspellingen. In het algemeen is een computationeel efficiënte en robuuste methode voor hyperparameter optimalisatie (HPO) essentieel, omdat het trainen en evalueren van deze algoritmen anders (te) duur kan zijn.

Vergelijkbare uitdagingen doen zich bijvoorbeeld voor in industriële toepassingen waar de optimalisatie van procesparameters gekenmerkt wordt door meerdere kostenfuncties, of objectieven, met beperkingen en onzekerheid (of ruis) in de objectieven. Traditionele evolutionaire benaderingen, zoals genetische algoritmen, zijn ongeschikt om deze problemen op te lossen, omdat zij heel veel experimenten vereisen. Op Bayesiaanse optimalisatie (BO) gebaseerde algoritmen genieten de voorkeur voor dergelijke dure problemen, maar, zoals het geval is voor HPO, zijn er maar weinig methoden die geschikt zijn voor de optimalisatie van meerdere onzekere objectieven met bijhorende beperkingen.

Hoewel het voorspellen van het vermogen van windmolens in *online* leeromgevingen ver af lijkt te staan van de bovengenoemde optimalisatie-uitdaging, hebben ze beide gemeen dat de beschikbare gegevens zo efficiënt mogelijk moeten worden gebruikt. Naar mijn mening is met de hoeveelheid gegevens die windmolenparken genereren, het *online* leren de meest haalbare strategie, waarbij het model opnieuw moet worden getraind telkens wanneer een nieuwe reeks gegevens beschikbaar komt. Het bijwerken van het model met nieuwe informatie

kan echter duur zijn bij het gebruik van traditionele Recurrente Neurale Netwerken (RNNs).

Dit proefschrift beoogt effiruiciënte algoritmen te ontwikkelen om gemeenschappelijke uitdagingen zoals tijdsbeperkingen, weinig data, en onzekerheid in voorspellings- en optimalisatieproblemen te overwinnen. Dit betekent dat de voorgestelde algoritmen slim moeten werken met de (onzekere) informatie die ze hebben en betere voorspellingen moeten opleveren dan bestaande benaderingen. In dit proefschrift verwijst de term onzekerheid naar de verschillende prestatie-/functiewaarden die worden waargenomen wanneer dezelfde inputconfiguratie (*hyperparameters*, procesconfiguratie, enz.) wordt geëvalueerd. De efficiëntie van de algoritmen is rechtstreeks gerelateerd aan de slimme keuze van nieuwe data punten (kandidaten) om vervolgens te evalueren in het optimalisatieproces, en de korte trainings-/testtijden van ML-modellen bij voorspellingstaken.

Om mijn onderzoeksdoel te bereiken, verkent dit proefschrift de kracht van Bayesiaanse optimalisatie en RNNs om dure en complexe problemen op te lossen. Een multi-objectief HPO-algoritme wordt voorgesteld om de onzekerheid in prestatie-evaluaties van ML-algoritmen te hanteren. De aanpak combineert *Tree Parzen Estimators* (TPE) met een *Gaussian Process Regression* (GPR) getraind met heterogene ruis. Op die manier moet het algoritme nieuwe punten voorstellen die waarschijnlijk niet gedomineerd worden, en die naar verwachting een maximale verbetering van de doelfunctie brengen. Het voorgestelde algoritme leidt tot een betere *hypervolume* score dan de multi-objectieve TPE- en GPR HPO-methoden uit de literatuur, wat zich vertaalt in nauwkeurige ML-algoritmen.

Bovendien heb ik een wijziging aangebracht in het TPE-algoritme voor single-objective HPO, om rekening te houden met prestatievariabiliteit zonder hiertoe GPR (of andere methoden) te combineren. In tegenstelling tot het oorspronkelijke TPE algoritme houdt onze methode rekening met de onzekerheid rond de prestatie-evaluaties, gemeten via cross-validatie, en neemt het gewichten in aanmerking om waarschijnlijkheidsfuncties te genereren waaruit op sequentiële wijze nieuwe invoerconfiguraties worden gegenereerd. Dankzij deze wijziging bleek het algoritme in staat hyperparameterconfiguraties te vinden met lagere classificatiefout.

Hoewel op BO gebaseerde algoritmen de voorkeur genieten om computationeel veeleisende problemen (zoals HPO en simulatie-optimalisatie) op te lossen, zijn er maar weinig methoden die de optimalisatie van meer dan één onzekere doelstelling in combinatie met meerdere simultane beperkingen in aanmerking nemen. In dit onderzoek hebben wij met succes GPR toegepast om de objectieven en beperkingen te emuleren bij de optimalisatie van een reëel lijmverbindingsproces, op basis van een beperkte hoeveelheid experimentele gegevens. Het voorgestelde BO-kader slaagde erin de optimale procesinstellingen op een zeer efficiënte manier (d.w.z. met een klein aantal bijkomende experimenten) te detecteren. Het verschil met evolutionaire algoritmen is dat het experimentele ontwerp bij het gebruik van BO algoritmen wordt aangestuurd op basis van een (interpreteerbare) acquisitiefunctie, die gerelateerd is aan de kansverdelingen voor de outputs (objectieven en constraints) in de nog niet geëvalueerde datapunten. Het BO algoritme richt zich op datapunten waar een zo groot mogelijke verbetering in doelfunctie wordt verwacht en waar de beperkingen met grote zekerheid voldaan zijn, hierbij rekening houdend met (heterogene) ruis (dit in tegenstelling tot evolutionaire algoritmen, die eenvoudigweg vertrouwen op onzekere steekproefgemiddelden als benadering).

Tenslotte ontwierp ik een methode gebaseerd op het *Long Short-term Cognitive Network* (LSTCN) om het probleem van gegevensvolatiliteit en korte verwerkingstijden in de vermo-

gensvoorspelling van windmolens aan te pakken. De *Short-term Cognitive Network* (STCN) blokken waaruit een LSTCN is opgebouwd, verwerken een tijdelijke brok gegevens met een snelle en deterministische leerregel die het algoritme geschikt maakt voor *online* leertaken. Het netwerk vertoonde de laagste voorspellingsfouten in vergelijking met andere state-of-the-art recurrente modellen. Bovendien was de LSTCN-aanpak aanzienlijk sneller dan deze alternatieve modellen.

— CHAPTER 1 —

A NECESSARY INTRODUCTION

Nowadays, Artificial Intelligence (AI) is omnipresent in everyday life. Current technological advances allow us to analyze huge amounts of data to generate knowledge that is used in many different ways, e.g. for automatic user recommendations [24], image recognition [121, 4], and supporting healthcare-related tasks [77]. In general, AI can be seen as a computer technology capable of carrying out functions that traditionally required human intelligence [45]. Although learning is a key element in many areas of AI, the very concept of learning is mainly studied in the Machine Learning (ML) subfield. According to [112], "a computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P** if its performance at tasks in **T**, as measured by **P**, improves with experience **E**". This general definition shares similarities with an optimization process where the algorithm uses some knowledge about the search space (the experience **E**; e.g., an initial set of observations, gradients, etc) to minimize or maximize an objective function (the task **T**) while paying attention to some quality metric (performance measure **P**; e.g., hypervolume, expected improvement, candidates' diversity, etc).

The terms **input** and **output** are present both in ML and optimization algorithms. For ML researchers an input is usually known as a set of features and the output as the label(s) of each problem instance (at least for supervised learning). Then, the learning algorithm is responsible for finding the relationship between the input and output. On the other hand, the optimization field adopts the term *decision variables* to refer to the input space where the optimization is performed, and *objectives* or function values to describe the output or response once a configuration is evaluated. Here, the goal of the optimization algorithm is not to find the relationship between input and output but to discover the input location where the *optimal* output can be observed.

Optimization algorithms may benefit from ML approaches, as these allow optimization algorithms to emulate complex input-output relationships from the observed data, informing them about potentially interesting areas in the search space. Conversely, optimization algorithms are needed in the ML field, for instance during model training or hyperparameter optimization (HPO). Optimization problems in many critical applications typically involve solving models with a large number of decision variables, and where the decisions are affected by uncertainty. The same challenge is present in modern ML models, where Deep Learning algorithms usually demand lots of (often uncertain) data and long training times. This motivates the scientific community to develop (or modify) algorithms to cover a wide range of challenging tasks from real-world problems.

1.1 Motivation and challenges

Time and cost constraints, data sparsity, and uncertainty are three common challenges to consider when applying ML and optimization algorithms to real-world problems. The former is relevant, for instance, during the training or inference of the ML model, the evaluation of the optimization objective(s), and even during the optimization's search process. Data sparsity is a natural consequence of these constraints (and/or other limitations, such as expert availability), and necessitates the development of efficient algorithms, that can make the most of the available data. Data uncertainty can manifest itself in different ways and have different causes, but broadly speaking, three main sources of uncertainty can be recognized [81, 15, 26]: (1) imprecise information or knowledge, (2) incomplete information, and (3) concepts or words that are inherently inaccurate. These sources have been widely studied in the subfield of Soft Computing in Artificial Intelligence [3]. Uncertainty can be present both at the input side (input uncertainty) and the output side (output uncertainty). In this dissertation, we focus on the latter, i.e., we consider uncertainty in the function evaluations of objective(s) and constraints (e.g., due to measurement errors, transmission errors, randomness, etc.). Consequently, different function values may be observed for repeated evaluations of the same input configuration.

ML algorithms and their parameters must be intelligently configured to make the most of the data. Those parameters that need to be specified *before* training the algorithm are usually referred to as *hyperparameters*: they influence the learning process but are not optimized as part of the training algorithm. Hyperparameter optimization (HPO) evidences the aforementioned challenges. Training an ML algorithm with a single hyperparameter configuration can take hours for many real-world problems. Consequently, we may start the optimization with a reduced number of hyperparameter configurations and use them as smartly as possible to find the "optimal" configuration. As with many optimization problems, the uncertainty in HPO is usually given by the performance (the objective) variability that is observed when a new configuration is tested. Despite the advances in optimization algorithms that account for uncertain objectives [132], the application of such algorithms in HPO has been less studied [143, 115].

Bayesian approaches have emerged as a powerful alternative to deal with expensive optimization problems and uncertainty modeling. Bayesian Optimization (BO) applications range from hyperparameter tuning of deep learning models [46, 16], to design optimization in engineering, and stochastic optimization in operational research (see [53] for a comprehensive review). Several BO methods have been proposed to solve complex decision-making problems involving one or multiple expensive and noisy objectives [134]. However, only a few methods have considered the constrained case [62, 58, 56], and even fewer have considered noisy objectives and/or constraints [48, 132].

Data sparsity and the need for short training times are challenges that are also present in settings such as online learning. In this case, the ML algorithm must adapt the knowledge acquired from previous training steps when new information enters the system. Power prediction of windmills is an example of online learning, as new data are constantly generated. Wind-based power generation has some peculiar characteristics, which need to be considered when designing new forecasting solutions. Firstly, it can be heavily affected by weather variability. Weather events are unavoidable, but their impact can be minimized when anticipated in advance. Secondly, wind turbines are dynamic systems that behave differently

over time (i.e., due to wear of turbine components, maintenance, etc). These characteristics make traditional ML methods (e.g., Recurrent Neural Networks or autoregressive models) inadequate to properly model these systems' dynamics. This means that new approaches are needed to improve the prediction of wind generation [103].

1.2 Scope and research goals

The goal of this research is *to develop efficient optimization and forecasting algorithms, capable of working with scarce data which may be volatile (in case of forecasting) or uncertain (in case of optimization).* In this thesis, the term *uncertainty* refers to the phenomena of observing different performance/function values when the same input configuration (hyperparameters, process configuration, etc) is evaluated. The efficiency is quantified by means of the training/testing times of the models in the forecasting tasks and by means of the number of required experiments/observations in the optimization tasks. This general objective can be split up into several research goals that frame the different challenges overcome by this research:

1. To develop single- and multi-objective optimization algorithms for problems where the objectives are expensive to evaluate, and are affected by noise;
2. To extend recurrent neural systems for forecasting tasks in online learning settings, where data might be volatile;
3. To evaluate the optimization or forecasting capabilities of the proposed algorithms using different study cases (hyperparameter optimization, adhesive bonding process optimization, and power forecasting in windmills) and algorithm configurations;
4. To compare the optimization or forecasting capabilities of the proposed algorithms with state-of-the-art methods.

1.3 Main contributions and thesis organization

This thesis comprises four main contributions: (1) the combination of GPR-based metamodeling and the sampling strategy of MOTPE in multi-objective HPO, (2) the modification of TPE to handle performance variability in single-objective HPO, (3) the (noisy and constrained) optimization of a novel adhesive bonding process, and (4) an LSTCN-based pipeline for power forecasting of windmills. Chapter 2 is devoted to the theory associated with the algorithms presented in this research. It describes the fundamentals of Bayesian optimization as a popular technique for solving expensive optimization problems, summarizes the most relevant concepts of multi-objective optimization, and shows how performance variability may be considered in ML algorithms. Then, each chapter details the aforementioned research contributions as follows:

1. Most HPO approaches take a deterministic perspective using the mean value of the performance observed in subsets of data, obtained from the cross-validation protocol. However, depending on the chosen split, the outcome may differ: a single HP configuration may thus yield different performance in different splits, for the same HP

configuration (i.e., the performance is *noisy*). Chapter 3 presents a novel algorithm to solve the multi-objective hyperparameter optimization problem of ML algorithms. This chapter discusses the main strengths of GPR-based optimization and MOTPE, and how they can be combined to account for performance variability while aiming to discover non-dominated hyperparameter configurations in multi-objective HPO. The combination of these methods is studied in the HPO of three ML algorithms in 12 different classification problems.

2. The use of GPR in HPO procedures has limitations, as HPO problems typically have a mixed search space. Tree Parzen Estimator (TPE) algorithm can accommodate nonreal variables very well and has been used in conjunction with low-fidelity methods to achieve state-of-the-art performance on several hyperparameter optimization problems [46]. However, TPE assumes that the performance metric corresponding to a given HPO configuration (e.g., accuracy) is deterministic. Chapter 4 analyzes the concepts of "*probability of being good*" and "*probability of being bad*" for the TPE algorithm for single-objective HPO. These concepts are the foundation of the adjustments proposed later to account for the performance *variability* of the hyperparameter configurations. We show that the resulting algorithm is capable of detecting better hyperparameter configurations (lower classification errors).

3. Several Bayesian Multi-objective Optimization (BMO) methods have been proposed to solve complex decision problems where the objectives are expensive to evaluate. Yet, few of these have considered noisy objectives and/or constraints. Chapter 5 illustrates the power of Bayesian Optimization approaches for optimizing a real-life adhesive bonding process. The chapter details the formulation of an acquisition function that combines the expected improvement over the objectives and the constraint feasibility, the use of a GPR that explicitly accounts for the heterogenous noise that is present in the outcomes of the real experiments, and a full Bayesian Multi-objective Optimization algorithm. The relevance of the presented approach is specifically analyzed for settings where the analyst can only afford a very limited number of observations (as is the case with costly physical experiments in a lab). The algorithms presented in this research illustrated the effectiveness of BMO to obtain *better* process configurations than state-of-the-art Evolutionary Multi-objective algorithms (EMOAs) and surrogate-assisted EMOAs. Additionally, HV-based optimization proved to be superior to the analyzed scalarization approach.

4. The LSTCN model has proven its forecasting capabilities, intrinsic interpretability, and short training/testing time for general problems [118, 117]. However, little is known about this model in online learning settings, where data might be available for a short time. Chapter 6 presents the application of Long Short-term Cognitive Networks (LSTCN) for online learning settings. In the LSTCN-based pipeline presented here, each iteration processes a data chunk using a Short-term Cognitive Network (STCN) block that operates with the knowledge transferred from the previous block. The numerical simulations presented in this chapter are focused on showing the improvement in terms of forecasting error and training/testing times compared to state-of-the-art recurrent neural networks.

This manuscript ends by summarizing the main conclusions of this research and possible topics in need of further investigation.

FIRST PART

THEORETICAL CONTRIBUTIONS

— CHAPTER 2 —

TALE OF METHODS AND COINS

Optimization algorithms and performance variability modeling

FOR A LONG time, tossing coins have been the classical example to represent the chance of occurrence of an event. This definition of *probability* is in the foundations of several areas of study such as statistics, science, machine learning, game theory, and others. That simple 50% chance of having "heads" or "tails" also shows us that many real actions are *uncertain* and we must assume them as such. This chapter outlines the terminology and notation used throughout this investigation, where the *probability* concept is always present. Section 2.1 presents the main characteristics of two Bayesian optimization methods for single-objective optimization: Gaussian Process Regression-based optimization and Tree Parzen Estimators. Then, Section 2.2 overviews the theory on multi-objective optimization to give way to the analysis of some multi-objective optimization algorithms in Section 2.3. The methods discussed in these sections were the starting point of this research. Lastly, Section 2.4 analyses how we can account for performance variability in the validation of Machine Learning algorithms.

2.1 Bayesian optimization in a nutshell

Let $f_m : \mathcal{X} \to \mathbb{R}^m$ be a mapping between an input search space $\mathcal{X}$ of dimension d and m *true* functions. Depending on the value of m, we may have a single-objective ($m = 1$), multi-objective ($1 < m \leq 3$) or many-objective ($m > 3$) optimization problem. This research is focused on solving problems falling in the first two categories and where f_m is expensive to evaluate. For simplicity's sake, we will use $y_j^{(i)}$ to refer to the j-th true function value of the i-th input configuration. Consequently, $\mathbf{y}^{(i)}$ refers to the vector of the true m functions' values for the i-th input configuration. Assume now that we would like to find the optimal configuration that minimizes the set of m functions:

$$\begin{aligned} \min \quad & [y_1, \ldots, y_m] \\ \text{s.t.} \quad & G_i \leq 0, i = 1, 2, \ldots, p, \\ & H_j = 0, j = 1, 2, \ldots, q \end{aligned} \tag{2.1}$$

where the functions G and H define p inequality and q equality constraints, respectively.

In addition, the function values can only be observed through a simulation model or real experimentation, and their outcomes are affected by *noise*. Thus, we only have access to noisy observations $\widetilde{\mathbf{y}}_j^{(i)} = \mathbf{y}_j^{(i)} + \varepsilon_j^{(i)}$, where $\widetilde{\mathbf{y}}_j^{(i)}$ represents the vector of observed goal

values in the j-th simulation replication at configuration i. We assume that the noise is heterogeneous, thus depending on the configuration i, $\mathbf{y}$ is then usually estimated as the mean value of r simulation replications: $\overline{\mathbf{y}}^{(i)} = \frac{\sum_{j=1}^{r^i} \tilde{\mathbf{y}}_j^{(i)}}{r^i}$.

Bayesian optimization (BO) is an iterative procedure with two key components: a probabilistic surrogate model and an acquisition function to determine a point in the search space that is worth evaluating next. The evaluation of the acquisition function is cheaper than the true function, which allows us to evaluate more input configurations than the amount we can evaluate with the expensive function. Then, traditional search methods, such as evolutionary algorithms [111] or gradient-based methods [119] can be used to solve the optimization of this acquisition function. BO approaches are most relevant for problems where the target function is expensive to evaluate. The search using this acquisition function should focus on new points located in promising regions or where the surrogate uncertainty is high. Consequently, the sampling behavior automatically trades off exploration and exploitation of the configuration search space.

Although many acquisition functions exist [135], the Expected Improvement (EI) [78] is the best-known criterion in BO [51, 60, 158]. The EI of a point $\mathbf{x}^{(*)}$ is given by

$$\mathrm{EI}(\mathbf{x}^*) = [y_{\min} - \widehat{y}(\mathbf{x}^*)]\, \Phi\left(\frac{y_{\min} - \widehat{y}(\mathbf{x}^*)}{\widehat{s}(\mathbf{x}^*)}\right) + \widehat{s}(\mathbf{x}^*)\phi\left(\frac{y_{\min} - \widehat{y}(\mathbf{x}^*)}{\widehat{s}(\mathbf{x}^*)}\right) \tag{2.2}$$

where ϕ and Φ are the standard normal density and standard normal distribution function, $y_{\min}$ is the best-observed value so far, $\widehat{y}$ is the surrogate prediction, and $\widehat{s}$ is the mean squared error (MSE) of the prediction.

Figure 2.1 shows the typical steps in BO for single-objective optimization. The algorithm starts with an initial design of points that can be obtained through random sampling or space-filling strategies (e.g., Latin hypercube sampling). In each iteration, a surrogate model is trained using the observed points, for which the true function values were obtained. Then, the acquisition function uses the predictive distribution of the probabilistic model to determine the usefulness of different candidate points, without having to evaluate them with the (expensive) true function. The optimal infill point obtained in this inner optimization is evaluated with the expensive true function and added to the set of observed configurations. This sampling/update process is repeated until the computational budget is depleted. Figure 2.2 shows one iteration in a BO algorithm assuming that EI is used as an infill criterion.

As for the surrogate model, Gaussian Process Regression [152] is often used as it provides both a prediction and an uncertainty estimate. In theory, we may use any Machine Learning model to model the true function, provided that it yields both estimators (function value and uncertainty) and does not require a lot of data to train (given the limited budget of expensive function evaluations). Recently, new methods have been proposed to find optimal configurations without modeling the target function. Tree Parzen Estimators (TPE) [16] is an optimization method proposed originally for hyperparameter optimization (HPO) of Machine Learning algorithms. Unlike GPR, TPE does not estimate the function value of a new configuration. Instead, it models the probability of sampling a "better" configuration than the one you have in an initial design. This research focuses on these two models and the following sections outline their main characteristics.

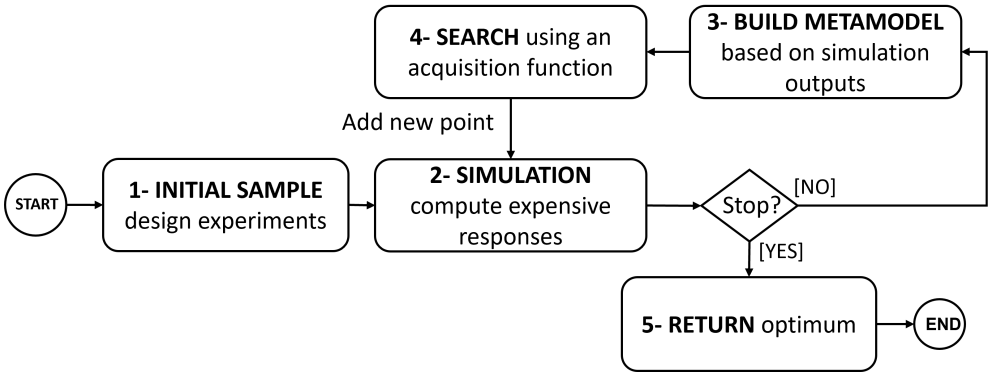
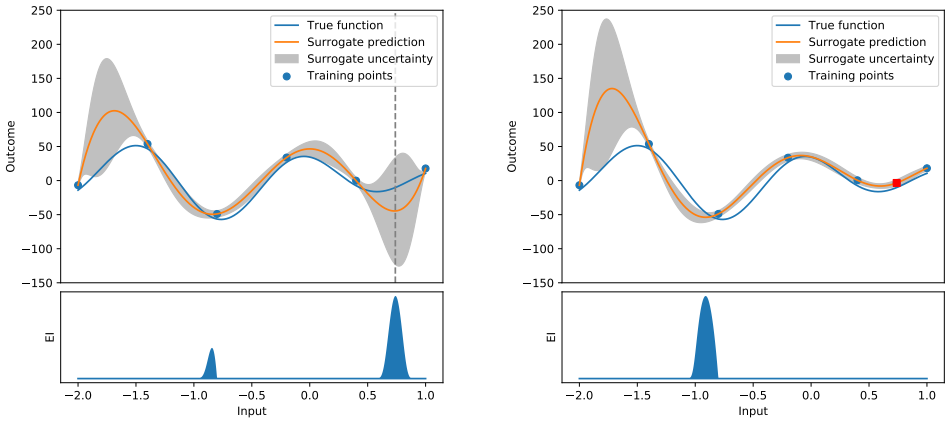


Figure 2.1: Typical steps in BO for single-objective optimization



(a) **(b)**

Figure 2.2: Selection of infill points in Bayesian optimization. At a given iteration, the solution that maximizes the acquisition function is shown as a vertical dotted line in (a). After sampling that point, the surrogate is updated including the new information (red point on (b))

2.1.1 Gaussian Process Regression: deterministic versus noisy observations

Gaussian Process Regression (GPR) or *kriging* [78, 152, 51] is a supervised learning technique. GPR is not only able to predict the outcomes at new configurations but can also approximate the uncertainty around these predictions.

More formally, the prediction of the target function at a new configuration $\mathbf{x}^{(*)}$ is obtained through the conditional probability $P(y^{(*)} \mid \mathbf{x}^{(*)}, \mathbf{X}, \mathbf{Y})$ that represents how likely the response $y^{(*)}$ is, given that we observed the target function at n input locations (contained in matrix $\mathbf{X}_{n\times d}$), yielding function values contained in matrix $\mathbf{Y}_{n\times 1}$. The initial set of points should have good space-filling properties (e.g., Latin hypercube sampling or quasi-random sequences). In GPR, the key assumption is that the unknown response function follows a Gaussian process, such that

$$y(\mathbf{x}) = m(\mathbf{x}) + M(\mathbf{x}) \tag{2.3}$$

where $m(\mathbf{x})$ represents the mean of the process, and $M(\mathbf{x})$ is a realization of a Gaussian random field with mean zero (also referred to as the *extrinsic uncertainty* [6]). Popular choices for $m(\mathbf{x})$ are known linear or nonlinear functions combinations of $\mathbf{x}$; an unknown constant β_0; or $m(\mathbf{x}) = 0$. $M(\mathbf{x})$ can be seen as a function, randomly sampled from a space of functions that, by assumption, exhibit spatial correlation according to a covariance function $k(\cdot, \cdot)$ (also referred to as *kernel*):

$$Cov(\mathbf{y}^{(i)}, \mathbf{y}^{(j)}) = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \tag{2.4}$$

Often, this covariance function is assumed to be stationary, meaning that its outcome only depends on the *distance* between the input locations. There exist multiple kernel functions in the literature [152], both the *squared exponential* or Gaussian kernel and the Mátern kernel are widely used in the literature.

The ordinary GPR prediction at an arbitrary unobserved location $\mathbf{x}^{(*)}$ is then given by:

$$\widehat{y}_o(\mathbf{x}^{(*)}) = \beta_0 + k_*[K_n]^{-1}(\mathbf{Y} - \mathbf{1}_n\beta_0) \tag{2.5}$$

where $\mathbf{1}_n$ is a $n \times 1$ vector of ones. The mean squared error on the prediction (MSE, also referred to as *kriging variance*) is given by [78]:

$$\widehat{s}_o^2(\mathbf{x}^{(*)}) = k_{**} - k_* K_n^{-1} k_*^T \tag{2.6}$$

where

$$K_n = \left[k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\right]_{n\times n}, \quad i, j \in \{1, \ldots, n\} \tag{2.7}$$

$$k_* = \left[k(\mathbf{x}^{(*)}, \mathbf{x}^{(i)})\right]_{1\times n}, \quad i \in \{1, \ldots, n\} \tag{2.8}$$

$$k_{**} = k(\mathbf{x}^{(*)}, \mathbf{x}^{(*)}) \tag{2.9}$$

Note that the extrinsic uncertainty in Equation (2.3) is imposed on the problem by assumption, to aid the construction of the predictive model. It does not account for the intrinsic uncertainty (i.e., the noise on the outcomes, which stems from the experiments). In our problem setting, replications of the same input location $\mathbf{x}^{(*)}$ may yield a different output $y_r^{(*)}$ at each replication r and the magnitude of the noise changes from location to location. In the literature, the model in Equation (2.3) is often applied to the mean of these replications [51, 32, 95]. In that way, though, the analyst is neglecting the stochastic nature of the experiment, which may lead to overconfident predictions [66, 69]. To avoid this, some GPR models have been proposed to consider the heteroscedastic nature of the noise in the problems [116, 89, 5, 18, 60]. For instance, Ankenman *et. al.* [6] provides a GPR model (referred to as *stochastic kriging*) that takes into account the heterogeneous (also referred to as heteroscedastic) noise observed in the data, and models the response values in the r-th replication at design point $\mathbf{x}^{(i)}$ as:

$$y_r(\mathbf{x}^{(*)}) = m(\mathbf{x}^{(*)}) + M(\mathbf{x}^{(*)}) + \varepsilon_r(\mathbf{x}^{(*)}) \tag{2.10}$$

where m and M are defined as in Equation (2.3), and $\varepsilon_r(\mathbf{x}^{(*)})$ is the intrinsic uncertainty observed in replication r.

The stochastic GPR prediction at an unobserved location $\mathbf{x}^{(*)}$ is then given by [6]:

$$\widehat{y}_s(\mathbf{x}^{(*)}) = \beta_0 + k_*[K_n + \Sigma_\varepsilon]^{-1}(\mathbf{Y} - \mathbf{1}_n\beta_0) \tag{2.11}$$

with $\mathbf{1}_n$ as a $n \times 1$ vector of ones, k_{**} and K_n defined as in Equation (2.9) and Equation (2.7) respectively, and

$$\Sigma_\varepsilon = diag\left[\frac{\frac{1}{r_i - 1}\sum_j^{r_i}\left[\widetilde{y}_j^{(i)} - \overline{y}^{(i)}\right]^2}{r_i}\right]_{n\times n}, \; i \in \{1, \ldots, n\} \tag{2.12}$$

The diagonal matrix Σ_ε contains the sample variance (noise) on the mean outcome of each observed configuration $\mathbf{x}^{(i)}$ sampled with r_i replications.

As pointed out previously, the GPR prediction error can be used to quantify the uncertainty of the prediction. For a GPR with noisy observations, it is computed as

$$\widehat{s}^2(\mathbf{x}^*) = k_{**} - k_*[K_n + \Sigma_\varepsilon]^{-1}k_*^T \tag{2.13}$$

Note that, if we remove the intrinsic noise added in Equation (2.11) and Equation (2.13), we obtain the prediction and the uncertainty estimate of the GPR with deterministic observations. For the sake of clarity, we will use $\widehat{y}_s$ and $\widehat{s}_s^2$ to refer to the predictors obtained with a GPR with noisy observations, and $\widehat{y}_o$ and $\widehat{s}_o^2$ for the predictors obtained with a GPR with deterministic observations.

2.1.2 Tree Parzen Estimators

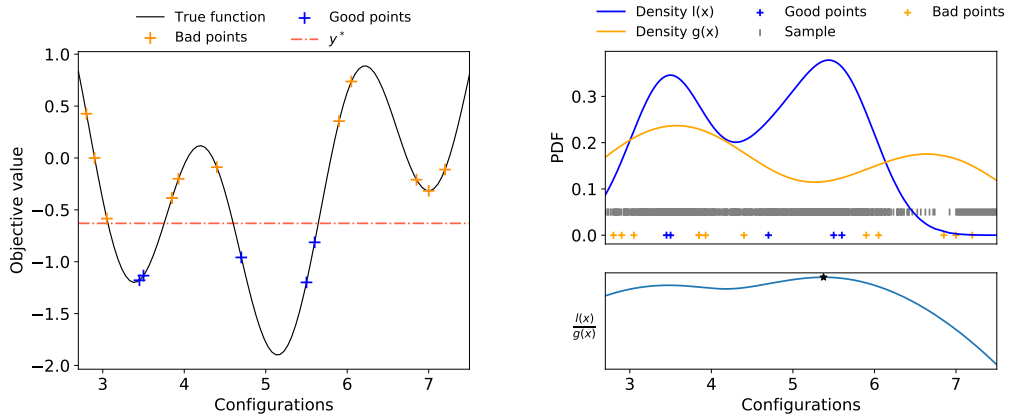
Whereas GPR models the probability distribution $P(y \mid \mathbf{x}, \mathbf{X}, \mathbf{Y})$, Tree Parzen Estimators (TPE) tries to model the probability $P(\mathbf{x} \mid \mathbf{X}, \mathbf{Y})$ of sampling a point given the set of observed responses [16]. TPE defines these probabilities using two densities:

$$p(x \mid y) = \begin{cases} l(x) & if \quad y(x) < y^*, x \in \mathbf{X} \\ g(x) & o.w \end{cases} \tag{2.14}$$

where $l(x)$ is the density estimated using the points $x^{(i)}$ for which $y^{(i)} < y^*$, and $g(x)$ is the density estimated using the remaining points. The value y^* is a user-defined quantile γ of the observed y values, so that $P(y(\mathbf{x}) < y^*) = \gamma$. Here, we can see l as the density of the configurations that may have the best response.

Figure 2.3 shows a toy example of TPE using densities $l(x)$ and $g(x)$ to suggest the next configuration to evaluate with the true function. Given the set of observed responses, we compute y^* as the percentile value for the γ quantile (dash red line). Then, all the points with an observed response lower than y^* will be considered as "good points", and the rest as "bad points" (Figure 2.3a). After this splitting procedure, the densities $l(x)$ and $g(x)$ are estimated using the set of "good" and "bad" points respectively, and use them to suggest the next point to evaluate (Figure 2.3b).

Algorithm 2.1 outlines the steps of TPE optimization. After the splitting process, the density of the configurations is estimated per dimension for both sets, and a number of



(a) **(b)**

Figure 2.3: Acquisition function evaluation for (a) $f(x) = sin(x) + sin(\frac{10}{3}x)$ (black line) in the range $[2.7, 7.5]$. The density estimation of the configurations in the *good* (orange) and *bad* (blue) set are used to suggest the next point to evaluate with the real function. The candidate set (gray dots) is sampled from density $l(x)$

samples are drawn from $l(x)$ (Equation 2.16). TPE employs a different EI formulation to suggest the next point to evaluate. Based on the densities $l(x)$ and $g(x)$, the EI in TPE is defined as

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y \mid x)dy \propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1} \tag{2.15}$$

This EI definition shows that we would like points x with a high probability under $l(x)$ and a low probability under $g(x)$ to maximize improvement. This EI formulation allows the model to balance exploration versus exploitation. Given that $l(x)$ is an estimated distribution and not a single value, the configurations drawn are likely close but not necessarily at the maximum of the expected improvement. Moreover, because the densities were estimated from some observed points, the selected configuration may not actually yield an improvement when evaluated and the densities will have to be updated.

Kernel Density Estimation (KDE) [140] is used to estimate a probability density function of a random variable. Let $\mathbf{X} = \left[x^{(1)}, x^{(2)}, \ldots, x^{(n)}\right]$ be independent and identically distributed samples drawn from some univariate distribution with an unknown density l at any given point x. We are interested in estimating the shape of this function l. Its kernel density estimator is

$$l(x) = \frac{1}{nh}\sum_{i=1}^{n} k\left(\frac{x - x^{(i)}}{h}\right) \tag{2.16}$$

where k is the kernel (a non-negative function) and $h > 0$ is a smoothing parameter known as *bandwidth*. The bandwidth here controls the quality of the model. A large bandwidth leads to a very smooth (i.e. high error in the density estimation) density distribution (orange line in Figure 2.4). On the other hand, a small bandwidth leads to an unsmooth (i.e. overfitting the

Algorithm 2.1 Tree Parzen Estimators for (noiseless) single-objective optimization

Require: $D = \{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times 1})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, γ: quantile parameter, c: number of candidates per iteration

for $i \leftarrow 1, \ldots, N$ **do**
  $\mathbf{T}_{1\times d} \leftarrow \{\}$
  for $j \leftarrow 1, \ldots, d$ **do**
    $D_l \leftarrow \left\{ \left(\mathbf{X}_j^{(k)}, \mathbf{Y}^{(k)} \right) \mid y^{(k)} < y^* \wedge p(y < y^*) = \gamma \right\}$
    $D_g \leftarrow D_j \setminus D_l$
    $l_j(x) \leftarrow \text{KDE}(D_l)$ ▷ KDE of good configurations
    $g_j(x) \leftarrow \text{KDE}(D_g)$ ▷ KDE of bad configurations
    $\mathbf{C}_{1\times c} \leftarrow \left\{ x^{(k)} \sim l_j(x) \mid k = 1, \ldots, c \right\}$ ▷ Sample candidates in dimension j
    $x^* \leftarrow \arg\max_{x\in\mathbf{C}} \text{EI}_{y^*}(l_j(x),\ g_j(x),\ x)$ ▷ Acquisition function maximization
    $\mathbf{T} \leftarrow \mathbf{T} \cup x^*$
  end for
  $\mathbf{Y}^* \leftarrow \text{SIMULATE}(\mathbf{T})$ ▷ Expensive evaluation of $\mathbf{T}$
  $D \leftarrow D \cup \{(\mathbf{T},\ \mathbf{Y}^*)\}$
end for
return the best-observed configuration contained in D

estimated density to the training points) density distribution (dark blue in Figure 2.4). One can either set manually this parameter or use Scott's and Silvermann's estimation methods [137].

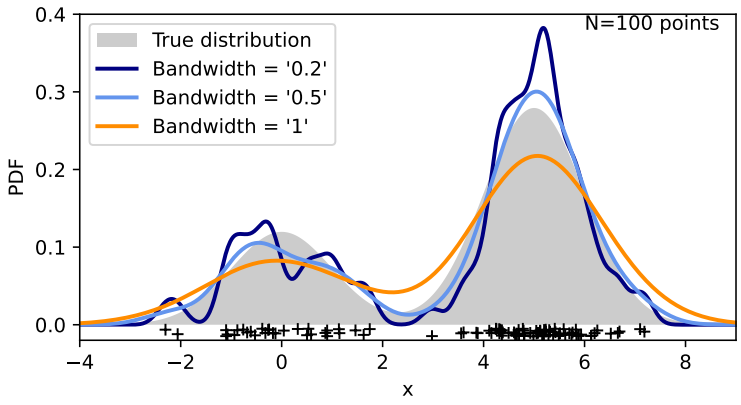


Figure 2.4: Influence of the bandwidth parameter in KDE

2.2 Overview of multi-objective optimization concepts

It is often required to consider the trade-off between two or more objectives ($m > 1$) in practical applications; such as the error-based performance measures and training time in Hyperparameter optimization of Machine Learning algorithms [83, 130, 114], ensemble error and diversity measures of classifier ensembles [23], break strength and production cost of

adhesive bonding process [79], and others. The goal in multi-objective optimization is to obtain a set of *Pareto-optimal* solutions, i.e., those configurations for which none of the objective values can be improved without negatively affecting any other [107]. More formally, for $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ two vectors in $\mathcal{X}$:

- $\mathbf{x}^{(1)} \preceq \mathbf{x}^{(2)}$ means $\mathbf{x}^{(1)}$ weakly dominates $\mathbf{x}^{(2)}$ (or $\mathbf{y}^{(1)} \preceq \mathbf{y}^{(2)}$) iff $y_j^{(1)} \leq y_j^{(2)}, \forall j \in \{1, \ldots, m\}$, and $\exists j \in \{1, \ldots, m\}$ such that $y_j^{(1)} < y_j^{(2)}$
- $\mathbf{x}^{(1)} \prec \mathbf{x}^{(2)}$ (or $\mathbf{y}^{(1)} \prec \mathbf{y}^{(2)}$) means $\mathbf{x}^{(1)}$ strictly dominates $\mathbf{x}^{(2)}$ iff $y_j^{(1)} < y_j^{(2)}, \forall j \in \{1, \ldots, m\}$

For a set of vectors $\mathbf{Y}^{n \times m}$ representing the evaluation of m objectives of n observations, the non-domination rank of a vector $\mathbf{y} \in \mathbf{Y}$ (denoted $rank(\mathbf{y}) \in \mathbb{N}$) is defined as follows:

- $rank(\mathbf{y}) = 1$ iff $\nexists \mathbf{y}' \in \mathbf{Y} : \mathbf{y}' \prec \mathbf{y}$
- $\forall k > 1 : rank(\mathbf{y}) = k$ iff $\nexists \mathbf{y}' \in \mathbf{Y} : \mathbf{y}' \prec \mathbf{y} \wedge \mathbf{y}' \notin \bigcup_{i<k} \{\mathbf{y}' \in \mathbf{Y} \mid rank(\mathbf{y}') = i\}$

Additionally, we denote $\mathbf{Y}_{rank(k)}$ the set $\{\mathbf{y} \in \mathbf{Y} \mid rank(\mathbf{y}) = k\}$. Vector $\mathbf{y} \in \mathbf{Y}$ is non-dominated iff $rank(\mathbf{y}) = 1$. Figure 2.5 shows the first three non-domination ranks in a set of points of a multi-objective problem with two objectives.

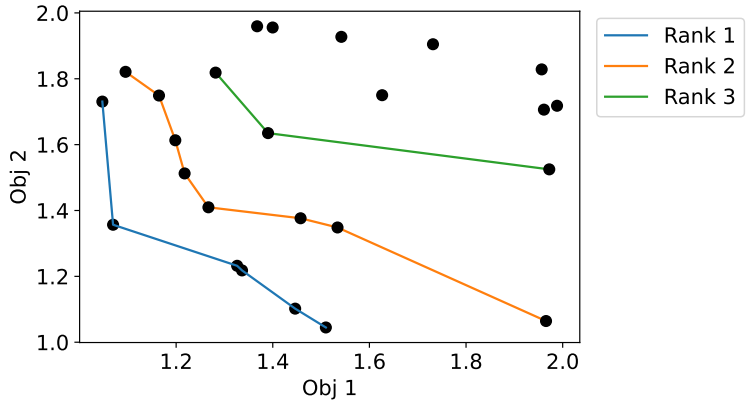


Figure 2.5: Example of non-domination ranks in an optimization problem of two objectives. Points not connected by a line are dominated by the points included in the first three ranks.

Classically, multi-objective optimization problems are often solved using scalarization techniques where the multi-objective optimization problem is transformed into a single-objective problem by the aggregation (or reformulation as constraints) of the objectives [108]. However, we should be cautious when using such techniques and analyze if the optimization results in Pareto points and if we can obtain all Pareto points on the Pareto front by changing the parameters of the scalarization [42].

A simple scalarization technique is the minimization of the weighted sum of objective functions. Then, the multi-objective optimization problem is reformulated to:

$$\min \sum_{i=1}^{m} w_i y_i \tag{2.17}$$

where $\mathbf{w} \in \mathbb{R}^{+m}$ is a vector of weights. The solution of a linear scalarization problem is on the Pareto front, no matter which weights in $\mathbb{R}^{+m}$ are chosen. However, if the Pareto front is non-convex, then, in general, there can be points on the Pareto front which are not found. In practice, a linear scalarization will tend to give only extreme solutions in concave fronts; that is, solutions that are optimal in one of the objectives. Figure 2.6 shows two multi-objective optimization problems with different types of Pareto fronts. Each red point represents the solution obtained by minimizing a linear scalarization problem, for different weight choices. In the case of the non-convex Pareto front (Figure 2.6b), even equal weights cannot lead to a solution in the middle part of the Pareto front.

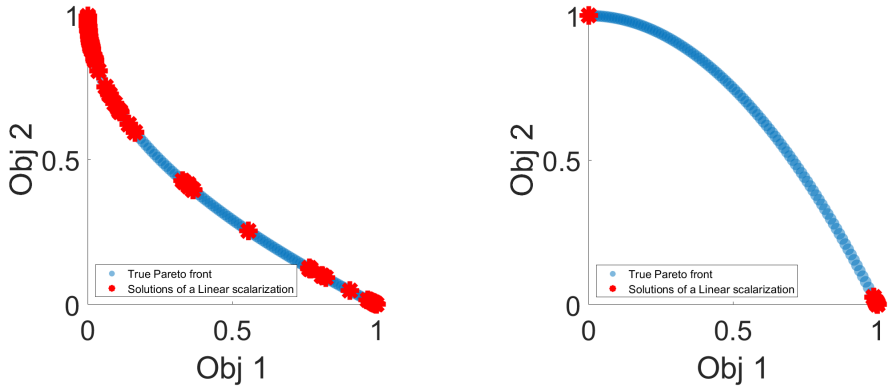


Figure 2.6: Linear scalarization function with different weights for (a) ZDT1 function with convex Pareto front, and (b) ZDT2 function with concave Pareto front

As an alternative to this scalarization, the augmented Chebychev scalarization function guarantees that all the points on the Pareto front can be obtained by minimizing the resulting scalarization problem. The augmented Chebychev scalarization function is defined as

$$\min \max_{i=\{1,\ldots,m\}} w_i y_i + \rho \sum_{i=1}^{m} w_i y_i \tag{2.18}$$

where $\mathbf{w} \in \mathbb{R}^{+m}$ is a vector of weights, and ρ is a small positive value (e.g., $\rho = 0.05$). By changing the weights, all points of the Pareto front can, in principle, occur as minimizers of the scalarization problem. Such points are potentially found in convex parts of Pareto fronts (Figure 2.7a) as well as in concave parts (Figure 2.7b). Note that the right-hand side of Equation 2.18 avoids the suggestion of weakly dominated points as minimizers of the scalarization problem, which for other functions, such as the (original) Chebychev scalarization, cannot be guaranteed. Other scalarization functions can be reviewed in [30].

In the context of multi-objective optimization problems, a performance indicator (or just indicator) is defined as a scalar measure of the quality of a Pareto front approximation. It can be obtained simply by looking at the Pareto front approximation or how much better

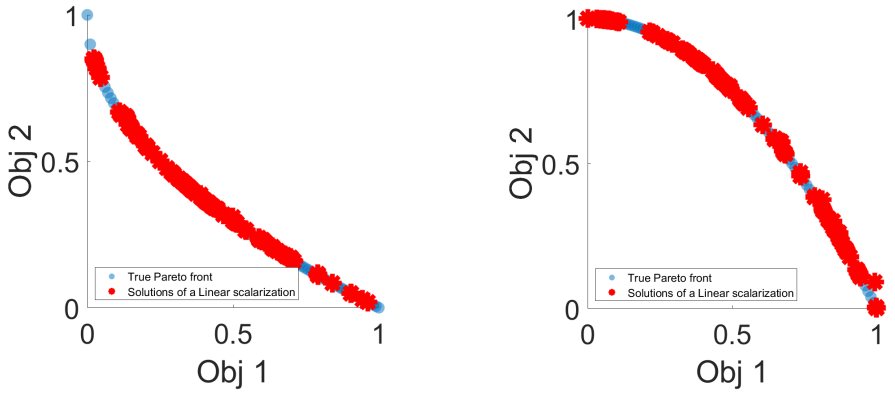


Figure 2.7: Augmented Chebychev scalarization function with different weights for (a) ZDT1 function with convex Pareto front, and (b) ZDT2 function with concave Pareto front

one Pareto front approximation is relative to another Pareto front approximation. The hypervolume indicator or s-metric [7] is an example of the former and IGD2+ of the latter.

The hypervolume is defined as follow for an approximation set $A \subset \mathbb{R}^m$:

$$\mathcal{H}(A) = Vol(\{\mathbf{y} \in \mathbb{R}^m : \mathbf{y} \preceq \mathbf{r} \wedge \exists \mathbf{a} \in A : \mathbf{a} \preceq \mathbf{y}\}) \tag{2.19}$$

where, $\mathbf{r}$ is a reference point. Practically speaking, the hypervolume is the area (or volume if $m \geq 3$) of the objective space that is dominated by the front obtained, w.r.t. a reference point. The optimal points move to the Pareto front, and the more they distribute along the Pareto front, the more space gets dominated, the higher the hypervolume is and the better the Pareto front obtained. Figure 2.8 show an example of the hypervolume computed for a minimization problem of two objectives.

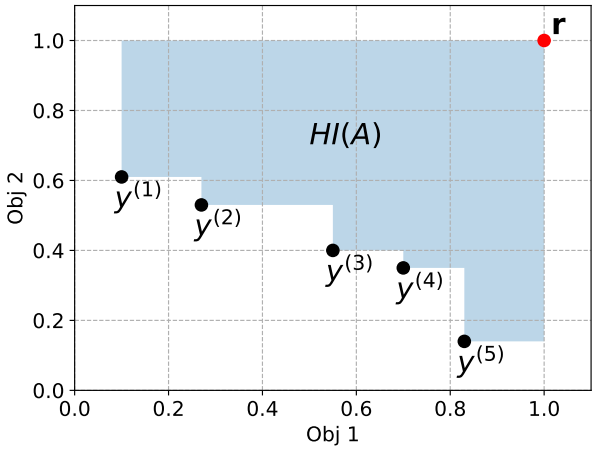


Figure 2.8: Illustration of a 2-D hypervolume surface for $A = \{\mathbf{y^{(1)}}, \mathbf{y^{(2)}}, \mathbf{y^{(3)}}, \mathbf{y^{(4)}}, \mathbf{y^{(5)}}\}$ and $\mathbf{r} = [1, 1]$

The basic idea of distance-based quality indicators is to measure the distance of the Pareto front to the solution set under consideration. As such, a reference set that well represents the Pareto front is required. A solution set that is close to every member of the reference set can have a good evaluation value. This idea can be materialized by averaging (or summing up) the distances of the reference set's members to their closest solution in the solution set, or finding the maximum value from these distances. The Inverted Generational Distance plus (IGD+) [71, 96] is a representative example, which considers the average Euclidean distance between each point of the reference and solution set. The IGD+ indicator can be formulated as:

$$IGD^{+}(A, Z) = \frac{1}{|Z|} \sum_{j=1}^{|Z|} \min_{\mathbf{a}_i \in A} d(\mathbf{a}_i, \mathbf{z}_j) \tag{2.20}$$

$a \in A \subset \mathbb{R}^m$, $z \in Z \subset \mathbb{R}^m$, A is the Pareto front approximation and Z is the reference Pareto front. The distance $d(\mathbf{a}, \mathbf{z})$ is defined for minimization problems as:

$$d(\mathbf{a}, \mathbf{z}) = \sqrt{\sum_{k=1}^{m} (\max\{a_k - z_k, 0\})^2} \tag{2.21}$$

When the solution $\mathbf{a}$ is dominated by the reference point z, this is exactly the same as the Euclidean distance since $a_k \geq z_k$ for all k. In Equation (2.21), when the solution $\mathbf{a}$ is not inferior to the reference point $\mathbf{z}$ with respect to the k-th objective (i.e., when $a_k \leq z_k$ for minimization problems), the k-th objective has no effect on the distance calculation. Figure 2.9 illustrates the distance calculation in the IGD+ indicator for a minimization problem with six reference points and three solutions given.

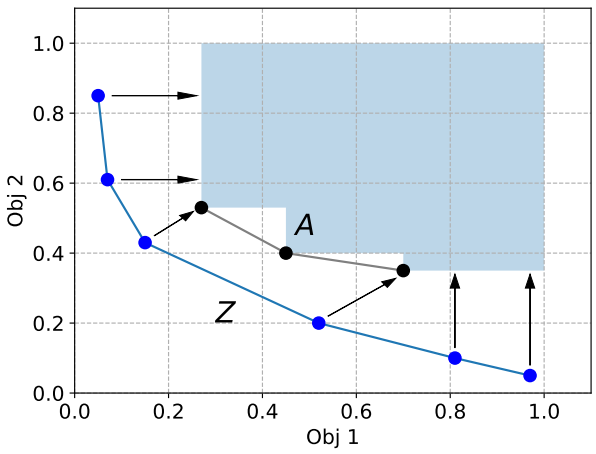


Figure 2.9: Illustration of the IGD+ indicator for a minimization problem of two objectives. The blue line is the reference Pareto front and the black line represents the approximated Pareto front.

2.3 Multi-objective optimization algorithms for expensive and/or noisy problems

The optimizers of the individual objectives are not usually of particular interest to the decision-maker, since these only reflect the extremes of the so-called *Pareto front* (i.e., the evaluation of the Pareto set in the objective space). The reader is referred to [105] for a comprehensive survey on *deterministic* multi-objective methods for engineering problems. In such methods, it is assumed that the objective (and, potentially, constraint) functions can be observed without noise (e.g., through a deterministic simulator, or noise-free experiments). However, we are interested in those problems where the objectives and/or constraints are *noisy*. The literature on *stochastic* multi-objective optimization algorithms is scarce, and consists mainly of evolutionary and Bayesian multi-objective optimization algorithms [32, 134, 115], with few results in constrained settings. This section details three metamodels-based multi-objective optimization algorithms adopted in our research (ParEGO, Multi-EGO, MOTPE).

ParEGO [82] is one of the algorithms that use a scalarization technique and exhibit a promising performance for multi-objective optimization problems where evaluations are expensive. ParEGO uses the augmented Chebychev scalarization function [82], which is often considered to solve general multi-objective optimization problems [59, 108, 72] given the properties described in the previous section.

ParEGO was originally proposed for noiseless multi-objective problems. However, Algorithm 2.2 presents a version of ParEGO (used in the exploratory phase of SK-MOCBA [59]) for noisy objectives, with the GPR proposed for noisy observations [6]. $Z_{\boldsymbol{\lambda}}$ (Line 3) is the result of applying the scalarization function to the objectives. The size of this matrix is $n \times r$ due to the stochastic nature of the experiment (hence, the r replications). The surrogate in this version of ParEGO is trained using the current set of configurations in $\mathbf{X}$ and the sample mean and variance of the scalarized objectives (Lines 4-5).

Instead of using a scalarization function, several algorithms exploit the advantages of an infill criterion for each of the objectives in the search of infill points [134]. Therefore, a surrogate model has to be trained for each objective individually. Multi-EGO [76] is an example of this strategy. The EIs for each of the objectives are directly used as fitness values in the multi-objective optimization. Then, a Genetic Algorithm (GA) maximizes the EIs of each objective function to find the non-dominated solutions according to the EI's values. Once performed enough iterations of the GA using the metamodels, we perform the expensive evaluation of the points in the Pareto front, and then the metamodels are updated with the new information.

Alternatively, the EI can be reformulated as a multi-objective infill criterion using hypervolume as a quality indicator. The Expected Hypervolume Improvement (EHVI) criterion uses the hypervolume indicator to measure the improvement we may expect from a new point [44, 33]. The hypervolume improvement of a point $\mathbf{y} \in \mathbb{R}^m$ is defined as the increment of the hypervolume indicator after $\mathbf{y}$ is added to the current approximation of $\mathbf{P}$ as

$$I(\mathbf{y}, \mathbf{P}) = \mathcal{H}(\mathbf{P} \cup \{\mathbf{y}\}) - \mathcal{H}(\mathbf{P}), \tag{2.22}$$

where $\mathcal{H}(.)$ is the hypervolume calculation function. Then, the EHVI is defined as the integration of the hypervolume improvement function over the non-dominated area [43]

Algorithm 2.2 ParEGO algorithm for noisy multi-objective optimization. Stochastic GPR is the surrogate model and EI is the acquisition function. The same number of simulation replications is assumed for each configuration

Require: $\{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times m\times r})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, r: number of simulation replications

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: $\boldsymbol{\lambda}_{1\times m} = [\lambda_1, \ldots, \lambda_m]\,,\ \sum_{i+1}^{m} \lambda_i = 1$ ▷ Random weight vector
3: $[Z_{\boldsymbol{\lambda}}]_{n\times 1\times r} = \left[Z_{\boldsymbol{\lambda}}^{(i)}\right],\ i = \{1, \ldots, n\}$ ▷ Augmented Chebychev scalarization
4: $\overline{\mathbf{Y}}_{n\times 1} \leftarrow \left[\frac{\sum_{k=1}^{r} Z_{\boldsymbol{\lambda}_k}^{(j)}}{r}\right]_{n\times 1},\ j = \{1, , \ldots, n+i-1\}$
5: $\overline{\mathbf{V}}_{n\times 1} \leftarrow \left[\frac{\frac{1}{r-1}\sum_{k=1}^{r}\left[Z_{\boldsymbol{\lambda}_k}^{(j)} - \overline{y}^{(j)}\right]^2}{r}\right]_{n\times 1},\ j \in \{1, \ldots, n+i-1\}$
6: $GP \leftarrow$ GP_FIT($\mathbf{X},\ \overline{\mathbf{Y}},\ \overline{\mathbf{V}}$) ▷ Surrogate training
7: $y_{min} \leftarrow \min_{\overline{y}\in\overline{\mathbf{Y}}}\ \overline{y}$
8: $\mathbf{x}^* \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}}\ \text{EI}(GP,\ y_{min},\ \mathbf{x})$ ▷ Acquisition function maximization
9: $\mathbf{Y}^*_{1\times r} \leftarrow \text{SIMULATE}(\mathbf{x}^*,\ r)$ ▷ Expensive evaluation of $\mathbf{x}^*$
10: $\mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{x}^*$
11: $\mathbf{Y} \leftarrow \mathbf{Y} \cup \mathbf{Y}^*$
12: **end for**
return the set of non-dominated solutions

$$EHVI(\mathbf{y}) = \int_{\mathbf{y}\in A} I(\mathbf{y}, \mathbf{P}) \prod_{i=1}^{m} \frac{1}{\widehat{s}_{oi}} \phi\left(\frac{y_i - \widehat{y}_{oi}}{\widehat{s}_{oi}}\right)\, dy_i \tag{2.23}$$

where A stands for the non-dominated area and $\phi(\cdot)$ is the standard normal density distribution function. The terms $\widehat{y}_{oi}$ and $\widehat{s}_{oi}$ refer to the objective and uncertainty estimators of the ordinary GPR models respectively. Previous studies have already used EHVI, often assuming noiseless objectives [33, 36, 100], or with homogeneous noise at best [37, 83].

We can also use the strategy followed by TPE to model the probability of sampling an input location given the evaluation of a set of points. However, we need to adapt the splitting procedure introduced in Section 2.1.2 to work with more than one objective. Ozaki et. al. [120] proposed to use the dominance rank of the objectives to split the observations into the "good" and "bad" sets. As far as we know, TPE for multi-objective problems (MOTPE) is the only implementation of TPE to solve multi-objective optimization problems.

MOTPE defines $P(x \mid \mathbf{X}, \mathbf{Y})$ using the following two probability density functions:

$$p(x|\mathbf{y}) = \begin{cases} l(x) & if \quad \mathbf{y} \prec \mathbf{P} \cup \mathbf{y} \preceq \mathbf{P} \\ g(x) & o.w \end{cases} \tag{2.24}$$

where $\mathbf{P}$ is a set of objective vectors such that $p(\mathbf{y} \prec \mathbf{P} \cup \mathbf{y} \preceq \mathbf{P}) = \gamma$, $l(x)$ is the probability density function estimated by using the observations $\{x^{(i)}\}$ such that $\mathbf{y}^{(i)} \prec \mathbf{P}$ (strictly dominates $\mathbf{P}$) or $\mathbf{y}^{(i)} \preceq \mathbf{P}$ (weakly dominates $\mathbf{P}$), and $g(x)$ is the probability density function estimated by using the remaining observations.

The observations are split by MOTPE for a specific γ in a greedy manner, as described in Algorithm 2.3. The splitting procedure in MOTPE, for noiseless multi-objective optimization, comprises two steps. The first step (lines 3-6) greedily appends better non-domination

ranked observations to the largest extent possible to S_l. The second step (line 7) appends the set obtained as a result of the hypervolume sub-selection problem (HSSP) [9] to S_l. The subroutine SOLVE_HSSP(S, s) (line 7 Algorithm 2.3) returns the result of HSSP for a set S and the size of subset s. Take, for instance, the example in 2.5 and suppose that MOTPE added all the points with $\mathbf{Y}_{rank(1)}$ to S_l but it still needs to select a subset of points $\mathbf{y}'$ with $\mathbf{Y}'_{rank(2)}$ to have that $|S_l| = \gamma\,|S|$. HSSP will choose those points from the second Pareto front that maximizes the hypervolume indicator given a reference point.

Algorithm 2.3 MOTPE splitting procedure (SPLIT_OBSERVATIONS method) for noiseless multi-objective problems

Require: $S = \{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times m})\}$: observations, γ: splitting parameter
1: $S_l \leftarrow \{\}$ ▷ Observations for $l(x)$
2: $i \leftarrow 1$
3: **while** $|S_l| + |S_{rank(i)}| \leq \gamma|S|$ **do**
4: $\quad S_l \leftarrow S_l \cup S_{rank(i)}$
5: $\quad i \leftarrow i + 1$
6: **end while**
7: $S_l \leftarrow S_l \cup \text{SOLVE\_HSSP}(S_{rank(i)}, \lfloor\gamma|S|\rfloor - |S_l|)$
8: $S_g \leftarrow S \setminus S_l$ ▷ Observations for $g(x)$
return S_l, S_g

MOTPE, as TPE, is originally intended to work with deterministic outcomes; i.e., the same input configuration will always yield the same outcome. The matrix $\mathbf{Y}_{n\times m}$ thus represents the deterministic values of the performance measures, evaluated for all input locations in S (no replications are required). MOTPE starts by splitting the set S using the non-domination rank of the configurations (line 4 Algorithm 2.4) and then, it constructs the probability density functions $l(x)$ and $g(x)$ in an identical manner to the original TPE algorithm.

Algorithm 2.4 Tree Parzen Estimator for (noiseless) multi-objective optimization

Require: $S = \{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times m})\}$: observations, N: number of iterations, c: number of candidates per iteration, γ: splitting parameter
1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: $\quad \mathbf{T}_{1\times d} \leftarrow \{\}$
3: $\quad$ **for** $j \leftarrow 1, \ldots, d$ **do**
4: $\quad\quad S_l, S_g \leftarrow \text{SPLIT\_OBSERVATIONS}(\{(\mathbf{X}_j, \mathbf{Y})\}, \gamma)$
5: $\quad\quad l_j(x) \leftarrow \text{KDE}(S_l)$ ▷ KDE of good configurations
6: $\quad\quad g_j(x) \leftarrow \text{KDE}(S_g)$ ▷ KDE of bad configurations
7: $\quad\quad \mathbf{C}_{1\times c} \leftarrow \{x^{(k)} \sim l_j(x) \mid k = 1, \ldots, c\}$ ▷ Sample candidates in dimension j
8: $\quad\quad x^* \leftarrow \arg\max_{x\in\mathbf{C}} \text{EHVI}_\gamma(l_j(x),\ g_j(x),\ x)$ ▷ Acquisition function maximization
9: $\quad\quad \mathbf{T} \leftarrow \mathbf{T} \cup x^*$
10: $\quad$ **end for**
11: $\quad \mathbf{Y}^* \leftarrow \text{SIMULATE}(\mathbf{T})$ ▷ Expensive evaluation of $\mathbf{T}$
12: $\quad S \leftarrow S \cup \{(\mathbf{T}, \mathbf{Y}^*)\}$
13: **end for**
return the set of non-dominated solutions in S

The acquisition function used to select the "best" sampled point (line 8 Algorithm 2.4) is the $\gamma-$ Expected Hypervolume Indicator ($EHVI_\gamma$), which is equivalent to the Expected

Improvement in TPE [120]. Using Equation (2.22), an generalizing Equation (2.23), the $EHVI_\gamma(\mathbf{x})$ is computed as follows:

$$EHVI_\gamma(\mathbf{x}) = \int_{\mathbf{y}\in A} (\mathcal{H}(\mathbf{P}\cup\{\mathbf{y}\}) - \mathcal{H}(\mathbf{P}))\, p(\mathbf{y}\mid x)\, d\mathbf{y} \tag{2.25}$$

where $\mathbf{P}$ is the current Pareto front approximation and A stands for the non-dominated area (including weakly non-dominated points). Then, applying Bayes's theorem in Equation (2.25) we obtain

$$EHVI_\gamma(\mathbf{x}) = \int_{\mathbf{y}\in A} (\mathcal{H}(\mathbf{P}\cup\{\mathbf{y}\}) - \mathcal{H}(\mathbf{P}))\, \frac{p(x\mid \mathbf{y})p(\mathbf{y})}{p(x)}\, d\mathbf{y}, \tag{2.26}$$

and based on Equation (2.24), the numerator of $EHVI_\gamma(\mathbf{x})$ is

$$\begin{aligned} EHVI_\gamma(\mathbf{x}) &= \int_{\mathbf{y}\in A} (\mathcal{H}(\mathbf{P}\cup\{\mathbf{y}\}) - \mathcal{H}(\mathbf{P}))\, p(x\mid \mathbf{y})p(\mathbf{y}\, d\mathbf{y} \\ &= l(x) \underbrace{\int_{\mathbf{y}\in A} (\mathcal{H}(\mathbf{P}\cup\{\mathbf{y}\}) - \mathcal{H}(\mathbf{P}))\, p(\mathbf{y})\, d\mathbf{y}}_{C_\gamma(\text{constant w.r.t. x})} \\ &= C_\gamma l(x) \end{aligned} \tag{2.27}$$

By construction, $\gamma = p(\mathbf{y} \prec \mathbf{P}\cup\mathbf{y} \preceq \mathbf{P})$ and $p(x) = \int p(x\mid \mathbf{y})p(\mathbf{y})\, d\mathbf{y} = \gamma l(x)+(1-\gamma)g(x)$. Therefore, we obtain the following:

$$\begin{aligned} EHVI_\gamma(\mathbf{x}) &= \frac{C_\gamma l(x)}{p(x)} \\ &= \frac{C_\gamma l(x)}{\gamma l(x) + (1-\gamma)g(x)} \\ &= \frac{C_\gamma}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}} \\ &\propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1} \end{aligned} \tag{2.28}$$

As in TPE for single-objective optimization, MOTPE should favor points x with a high probability under $l(x)$ and a low probability under $g(x)$ to maximize the improvement in the current Pareto front.

2.4 Performance variability in the validation of Machine Learning algorithms

The stochastic nature of many real problems implies that decisions are often to be made based on noisy observations (and/or, possibly, uncertain domain knowledge). This is no different for ML models and AI-based decision-making. Therefore, it is important to represent the uncertainty in any AI-based system in a trustworthy manner. Uncertainty is understood as doubt or ambiguity and it can be observed both in the data or the performance evaluation of the ML model.

The irreducible uncertainty in data that yields uncertain predictions is known as aleatoric uncertainty (or data uncertainty) [1]. This type of uncertainty is not a model property, but an inherent property of the data distribution. Hence, it is irreducible. On the other hand, epistemic uncertainty (or model uncertainty) occurs due to inadequate knowledge acquired by the model. You may have lots of data to train your ML model, but an accurate prediction will be compromised if the information quality is poor.

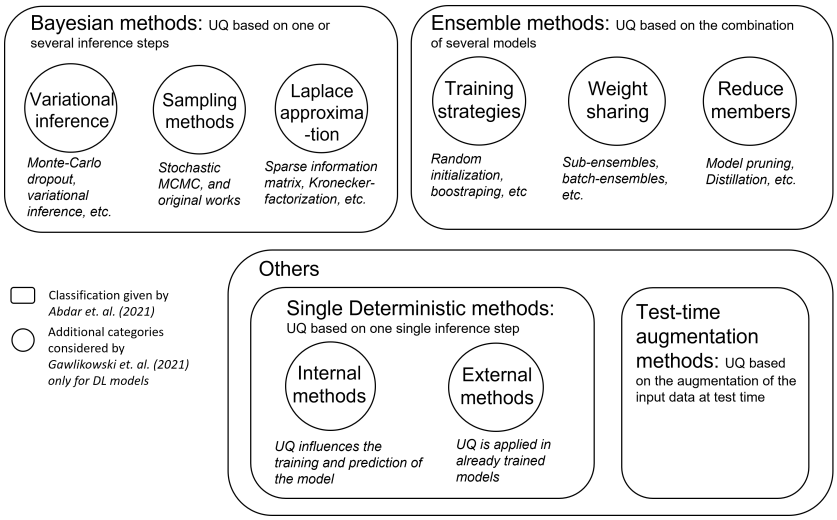


Figure 2.10: Summary of the uncertainty quantification techniques for ML and DL analyzed by Abdar *et. al.* [1] and Gawlikowski *et. al.* [57]. The categories analyzed by the latter (circles) were discussed only for DL algorithms.

The ML process life cycle involves data collection and filtering, model tuning, and model exploitation with operational data; and the uncertainty may propagate across the complete cycle. Uncertainty quantification (UQ) plays a role in each of these steps and, in general, decisions made without UQ are usually not trustworthy [8, 61]. All sources of uncertainty discussed before may occur during the first step and the UQ method should be focused on evaluating the completeness and accuracy of the training/testing data. Similarly, uncertainty corresponding to the model's performance (a combination of aleatoric and epistemic uncertainty) should be analyzed when designing/tuning ML algorithms. Lastly, once the model is deployed, a systematic analysis of the predictions in terms of UQ increases the overall confidence in the model and makes data-centric tools and methods practically useful.

Abdar *et. al.* [1] presents a detailed literature review of uncertainty quantification techniques for ML and DL algorithms. The authors advocate two main categories: Bayesian and Ensemble-based methods, and a general group of other methods that do not fall under the first two categories. Curiously, some GP-based techniques were not considered by the authors as Bayesian UQ methods, even when these find their essence in the fundamentals of Bayesian optimization. A more detailed categorization of UQ is proposed by Gawlikowski *et. al.* [57] in a pre-print focused only on Deep Neural Networks. Figure 2.10 summarizes the categorization provided by these two surveys. Unfortunately, it is not clear from these articles how these UQ methods can be used to evaluate algorithm performance. More specif-

ically, if we know how (un)certain the prediction is, how can this information be considered in the overall performance evaluation of the algorithm?

Even in sensitive tasks such as hyperparameter optimization [50], automatic model selection [104], and Neural Architecture Search (NAS) [41, 129], performance uncertainty (or performance variability) has been usually neglected by relying completely on the mean performance observed; e.g., in ensemble methods, or in cross-validation protocols. The latter has been widely used in the ML community because it is a suitable alternative for model selection in the absence of prior knowledge [136]. When evaluating different models, part of the available data is held out as a test set and the rest is dedicated to training the model. The cross-validation protocol splits the training set into k smaller sets. For each of the k "folds", the model is trained using the other $k-1$ folds, and the resulting model is then validated on the left-out fold. The performance measure reported by k-fold cross-validation is then the average of the performance observations across the different splits. Figure 2.11 shows the typical cross-validation and testing of ML models and an example of 5-fold cross-validation.

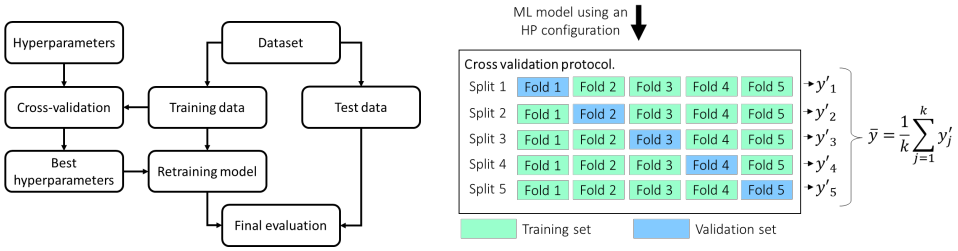


Figure 2.11: Evaluating estimator performance. (a) Typical cross-validation and testing of an ML model. (b) Example of a 5-fold cross-validation protocol. The training set is split into k folds (here: $k = 5$). The ML algorithm is trained and validated on each fold for a given HP configuration, yielding k estimates of algorithm performance. Traditionally, the average performance across splits is then used as an estimator for the overall algorithm performance, given an HP configuration.

While it is known that cross-validation provides an estimate of the expected prediction error, it is also known that its variance may be very large [21]. This may complicate model comparisons, and cause erratic behavior in the expected prediction error [14, 12]. Therefore, tasks such as hyperparameter optimization should consider the variance of the performance estimate(s) when looking for the best hyperparameter combination. Although many authors recognize the aforementioned problem, the literature on HPO accounting for the performance variability of ML algorithms (both in single-objective HPO [39, 63, 110] and in multi-objective HPO [83, 66]) remains scarce [115]. These papers have mainly explored the impact of different noise handling strategies on the results of *existing* algorithms, while it may be more beneficial to account for the noise by adjusting the metamodels (if any) used, and/or the algorithmic approach.

Lastly, we would like to emphasize that our use of the term "variance" in the following chapters should not be confused with the traditional use of this term in ML, as in the well-known "bias-variance" tradeoff. In the latter, the bias is the error made in the learning algorithm from erroneous assumptions, while the variance results from sensitivity to small fluctuations in the training set. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). High variance, by contrast, may

result from an algorithm modeling the random noise in the training data (overfitting). Lowering a model's bias (e.g., by reducing the error on the training data) may lead to overfitting, which increases the model's variance (i.e., it doesn't generalize well to new data). On the other hand, an ML algorithm that is too simplistic or that ignores existing relationships in the training data (resulting in underfitting and high bias) may generalize better to new data (low variance). How much "learning" and "forgetfulness" the algorithm should perform during training evidence the aforementioned bias-variance trade-off.

Cross-validation (CV) is a common technique in ML when training and validating ML models using a single dataset. In k-fold CV, the data is split into k parts, and each part is sequentially left out t be used as a validation set for the ML model that results from training on the $k-1$ other parts. The average error thus obtained on the entire dataset (the average of the k error estimates resulting from the CV) is then considered as *the* estimate of the true error of the ML model that we would obtain if we used the training algorithm on the entire dataset. As observed by Kohavi [85] in one of the early studies related to the bias-variance trade-off, a higher value of k leads to more folds, reducing bias but increasing variance and computational cost. Conversely, a lower value of k increases bias but reduces variance and computational cost. In this dissertation, the term variance refers to the concept known from the probability and statistics field as a *measure of dispersion* in a random variable.

The performance of an ML algorithm is typically measured by means of a single numeric value, usually obtained as a sample mean of different performance values (obtained from a cross-validation protocol, or a bootstrap ensembling model), or obtained by training and evaluating the ML algorithm only once using a single data set (typically in Deep Learning settings). Consider, for instance, the performance distribution obtained when a k-fold cross-validation protocol is applied to evaluate an ML algorithm. Just using the *sample mean* of the performances evaluated on the k different splits neglects the uncertainty in this performance; as shown later, this may lead to worse HPO solutions, or short-sighted conclusions about the difference in performance between alternative algorithms. The *sample variance* of the observed performances can be considered as an indicator of this uncertainty.

— CHAPTER 3 —

WHEN ONE BECOMES TWO

Multi-objective hyperparameter optimization with performance variability

The optimization of two or more objectives is often required in practical problems. This is the case, for instance, in hyperparameter optimization (HPO) of Machine Learning (ML) algorithms. Additionally, as training and evaluating an ML algorithm is usually expensive (e.g., because of training time, memory consumption, etc), the HPO method needs to be computationally efficient to be useful in practice. Most of the existing approaches to multi-objective HPO use evolutionary strategies and metamodel-based optimization [115], but neglecting that performance measures are, in fact, noisy. This chapter presents results on multi-objective HPO with noisy performance evaluations of the ML algorithm, using a combination of Tree Parzen Estimators (TPE) and Gaussian Process Regression (GPR) with heterogeneous noise.[1] Section 3.1 introduces the hyperparameter optimization of ML algorithms as a multi-objective problem and Section 3.2 presents the proposed optimization algorithm. Section 3.3 describes the experimental setting designed to evaluate the proposed algorithm, and Section 3.4 shows the improvement in the hypervolume obtained when compared with HPO using stand-alone multi-objective TPE and GPR-based optimization.

3.1 Multi-objective hyperparameter optimization

In mathematics and computer science, an algorithm is a finite sequence of well-defined instructions that, when fed with a set of initial inputs, eventually produces an output. Figure 3.1 shows that in HPO, the optimization algorithm forms an "outer" shell of optimization instructions; the "inner" optimization refers to the training of the target ML algorithm (e.g., Neural Networks (NN), Support Vector Machine (SVM), etc). This inner optimization trains the target algorithm to perform the task it should perform (e.g., house prices prediction). In turn, the HPO algorithm takes the hyperparameters of the target ML algorithm as input and produces a number of performance measures as output (e.g., Root Mean Square error (RMSE), training time, etc). The aim of the HPO algorithm is to optimize the set of hyperparameters, in view of obtaining the best possible outcomes for the performance measures considered.

[1] The content of this chapter has been included in the publications *"Multi-objective Hyperparameter Optimization with Performance Uncertainty". In: Proceedings of the Optimization and Learning Conference (2022)* [114] and *"A survey on multi-objective hyperparameter optimization for Machine Learning". In: Artificial Intelligence Reviews* [115].

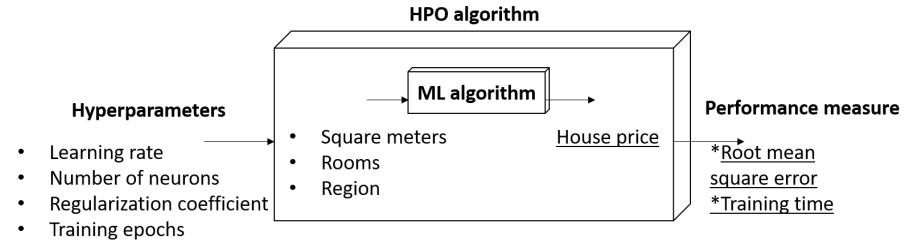


Figure 3.1: Example of the interplay between the HPO algorithm and the target ML algorithm (in this case, an ANN for predicting house prices)

Consider a target ML algorithm $\mathcal{A}$ with d hyperparameters, such that the i-th hyperparameter has a domain denoted by $\mathcal{X}_i$. The overall *hyperparameter configuration space* is denoted as $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times ... \times \mathcal{X}_d$. A vector of hyperparameters is denoted by $\mathbf{x} \in \mathcal{X}$, and an algorithm $\mathcal{A}$ with its hyperparameters set to $\mathbf{x}$ is denoted by $\mathcal{A}_\mathbf{x}$. Generally, the available data are split into a training set, a validation set, and a test set. The *learning* process of the algorithm takes place on the training set ($\mathcal{D}_{train}$) and is validated on the validation set ($\mathcal{D}_{valid}$). Lastly, the ML algorithm with the best HP configuration is evaluated using the test set. We can then formalize the *single-objective* HPO problem as [50]:

$$\min_{\mathbf{x} \in \mathcal{X}} V(\mathcal{L} \mid \mathcal{A}_\mathbf{x}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

where $V(\mathcal{L} \mid \mathcal{A}_\mathbf{x}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ is a validation protocol that uses a loss function $\mathcal{L}$ to estimate the performance of a model $\mathcal{A}_\mathbf{x}$ trained on $\mathcal{D}_{train}$ and validated on $\mathcal{D}_{valid}$. Popular choices for the validation protocol $V(\cdot)$ are the holdout and cross-validation process (see [19] for an overview of validation protocols).

The previous definition can be readily extended to multi-objective optimization (see [96]). Consider a multi-objective HPO problem with d hyperparameters and a set $\mathbf{L}$ containing m performance measures (objective functions). This generalizes the concept of the "loss function" in the single-objective formulation, to include error-based measures and other metrics such as algorithm complexity, training time, memory consumption, etc. The multi-objective HPO problem can then be formalized as follows (assuming that all performance measures should be minimized):

$$\min_{\mathbf{x} \in \mathcal{X}} V(\mathbf{L} \mid \mathcal{A}_\mathbf{x}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Trade-offs exist, for instance, between the performance of a model and its training time (increasing the accuracy of a model often requires larger amounts of data and, hence, a higher training time [128]), or between different error-based measures (e.g., between confusion matrix-based measures [145] of a binary classification problem [65]). Considering these trade-offs is often crucial: e.g., in medical diagnostics [146], the simultaneous consideration of objectives such as sensitivity and specificity is essential to determine if the ML model can be used in practice. As in any multi-objective problem, the goal in multi-objective HPO is

to obtain the *Pareto-optimal* solutions, i.e., those solutions for which none of the objectives can be improved without negatively affecting any other objective.

Most multi-objective HPO approaches take a deterministic perspective using the mean value of the performance observed in subsets of data (cross-validation protocol) [115]. However, depending on the chosen sets, the outcome may differ: a single HP configuration may thus yield different results for each performance objective, implying that the objectives are *noisy*. Apart from the work of [83] and [66], little has been done on noisy multi-objective HPO. The former uses the re-interpolation (RI) method by Forrester et. al. [52] to not use any replicates of the performance measures. Here, m GPR models for noisy observations are created for all m objective functions using the current observed points and nugget estimation (the noise). Then, the current design is re-interpolated to induce m ordinary GPR models trained with the responses predicted before. As usual, the new ordinary GPR models are finally used to optimize the EI infill criterion.

On the other hand, Horn et. al. [66] analyze several strategies to account for noisy objectives where each point may be re-evaluated a different number of times. To achieve this, the authors focus on two strategies: re-evaluate only promising settings (according to the dominance relation and the number of prior re-evaluations), while inferior settings are evaluated only once; and a combination of sometimes performing no replications at all (mostly at the beginning of the optimization) and k re-evaluations of Pareto optimal points (at the end of the optimization). In contrast to these studies, the algorithm presented in this chapter always performs k replications of the performance measures and uses the stochastic GPR proposed by [6] to handle the noise.

We conjecture that an HPO approach for noisy performance will outperform alternative approaches that assume the relationships to be deterministic. Specifically, we propose to combine MOTPE and GPR metamodeling to handle the existing uncertainty in the performance evaluation of a hyperparameter configuration. In addition, the MOTPE sampling strategy ensures that the suggested candidate set has hyperparameter configurations that are unlikely to be dominated. Then, selecting the next configuration to be evaluated from this set, reinforces the assumption that the new hyperparameter configuration maximizes the performance measure (throughout the acquisition function optimization) and is not dominated by any other configuration (thus, increasing the hypervolume measured with the Pareto front).

3.2 Using TPE sampling strategy with GPR metamodeling

We propose to combine the sampling strategy of MOTPE with GPR-based optimization. The algorithm (Algorithm 3.1) starts with an initial set of hyperparameter configurations obtained with random sampling. Then, the performance of the ML algorithm is evaluated using a fixed number of replications. In this case, this value is given by the number of folds of a cross-validation protocol (see Chapter 2), allowing to account for the performance variability of the initial and subsequent configurations. We then perform two tasks in parallel. On the one hand, we use the augmented Chebychev scalarization function [82] (with a random combination of weights) to transform the multiple objectives into a single objective problem (lines 2-5). Then, we train a (single) stochastic GPR metamodel on these scalarized objective values (line 6).

At the same time, we perform the splitting process used by [120] (Algorithm 2.3) to divide the hyperparameter configurations based on the observed mean performance, and conform the sets of observations with a "good" and "bad" performance. To that end, our approach uses the same greedy selection followed by MOTPE, and controlled by the parameter γ [2]. Thus, this splitting strategy considers HP configurations with better non-domination rank to fall into the "good" set. Subsequently, the algorithm estimates the densities $l(x)$ and $g(x)$ for each separate input dimension (Lines 12-13).

Algorithm 3.1 GPR metamodeling and TPE sampling for noisy multi-objective HPO. Stochastic GPR is the surrogate model and MEI is the acquisition function. The same number of simulation replications is assumed for each configuration

Require: $\{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times 1\times r})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, r: number of simulation replications, c: number of candidates per iteration, γ: splitting parameter

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: $\quad \boldsymbol{\lambda}_{1\times m} = [\lambda_1, \ldots, \lambda_m]\,,\ \sum_{i+1}^{m} \lambda_i = 1$ ▷ Random weight vector
3: $\quad [Z_{\boldsymbol{\lambda}}]_{n\times 1\times r} = \left[Z_{\boldsymbol{\lambda}}^{(i)}\right],\ i = \{1, \ldots, n\}$ ▷ Augmented Chebychev scalarization
4: $\quad \overline{\mathbf{Y}}_{n\times 1} \leftarrow \left[\frac{\sum_{k=1}^{r} Z_{\boldsymbol{\lambda}k}^{(j)}}{r}\right]_{n\times 1},\ j = \{1, , \ldots, n+i-1\}$
5: $\quad \overline{\mathbf{V}}_{n\times 1} \leftarrow \left[\frac{\frac{1}{r-1}\sum_{k=1}^{r}\left[Z_{\boldsymbol{\lambda}k}^{(j)} - \overline{y}^{(j)}\right]^2}{r}\right]_{n\times 1},\ j \in \{1, \ldots, n+i-1\}$
6: $\quad GP \leftarrow$ GP_FIT($\mathbf{X}$, $\overline{\mathbf{Y}}$, $\overline{\mathbf{V}}$) ▷ Surrogate training
7: $\quad \mathbf{x}_{min} \leftarrow \arg\min_{y\in\overline{\mathbf{Y}}}\ y$
8: $\quad \widehat{\mathbf{Z}}_{min} \leftarrow$ GP_PREDICT($\mathbf{x}_{min}$)
9: $\quad \mathbf{T}_{c\times d} \leftarrow \{\}$
10: $\quad$ **for** $j \leftarrow 1, \ldots, d$ **do**
11: $\quad\quad S_l, S_g \leftarrow$ SPLIT_OBSERVATIONS($\{(\mathbf{X}_j, \mathbf{Y})\}, \gamma$)
12: $\quad\quad l_j(x) \leftarrow$ KDE(S_l) ▷ KDE of good configurations
13: $\quad\quad g_j(x) \leftarrow$ KDE(S_g) ▷ KDE of bad configurations
14: $\quad\quad \mathbf{C}_{1\times c} \leftarrow \{x^{(k)} \sim l_j(x) \mid k = 1, \ldots, c\}$ ▷ Sample candidates in dimension j
15: $\quad\quad \mathbf{T}_j \leftarrow \text{SORT}\left(\left\{log\frac{l_j(\mathbf{C}^{(k)})}{g_j(\mathbf{C}^{(k)})} \mid k = 1, \ldots, c\right\}\right)$
16: $\quad$ **end for**
17: $\quad Q \leftarrow \left[T^{(k)} \mid \sum_{j=1}^{d} T_j^{(k)} > 0\right]_{\beta\times d}, k = \{1, \ldots, c\}, \beta \leq c$ ▷ Removing $AS \leq 0$
18: $\quad \mathbf{x}^* \leftarrow \arg\max_{\mathbf{q}\in Q}$ MEI(GP, $\widehat{\mathbf{Z}}_{min}$, $\mathbf{q}$) ▷ Acquisition function maximization
19: $\quad \mathbf{Y}^*_{1\times r} \leftarrow$ SIMULATE($\mathbf{x}^*$, r) ▷ Expensive evaluation of $\mathbf{x}^*$
20: $\quad \mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{x}^*$
21: $\quad \mathbf{Y} \leftarrow \mathbf{Y} \cup \mathbf{Y}^*$
22: **end for**
return the set of non-dominated solutions

Using the densities $l(x)$, we randomly sample a candidate set of c elements for each input dimension (Line 14). These individual samples are sorted according to their log-likelihood ratio $log\frac{l(x)}{g(x)}$, such that the higher this score, the larger the probability that the input value is sampled under $l(\mathbf{x})$ (and/or the lower the probability under $g(\mathbf{x})$). Both MOTPE and TPE

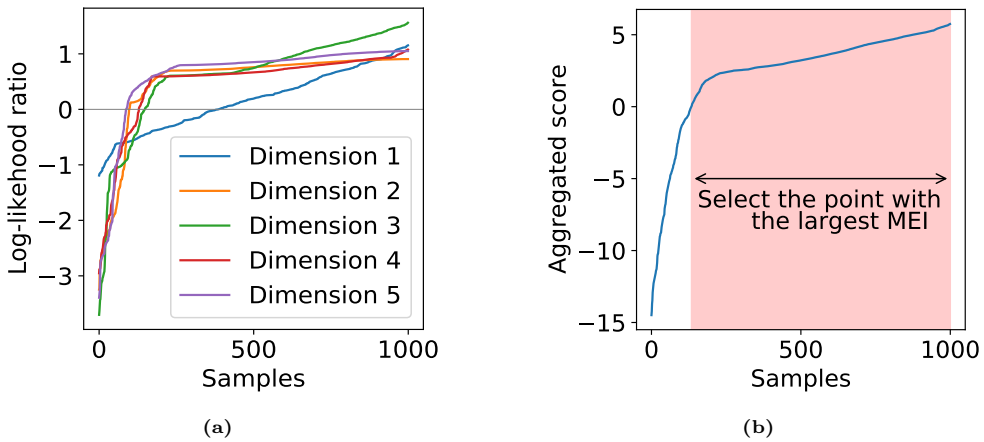
[2] Notice that both in [120] and in our algorithm, the parameter γ represents a percentage of the known observations that may be considered as "good".

select the single value with the highest score [16, 120] for each dimension. We propose to select the best hyperparameter configuration using the aggregation of the scores obtained for each dimension; this way the decision is made considering every dimension at the same time and not individually. The aggregated score is computed as $AS(\mathbf{t}) = \sum_{i=1}^{d} log\frac{l(t_i)}{g(t_i)}, \mathbf{t} \in \mathbf{T}$ for each configuration. Then, the algorithm suggests the configuration that maximizes the *Modified Expected Improvement* (MEI) [126] in the set $\mathbf{Q}$ of the configurations with a positive aggregated score. Note that a negative score here indicates that the sample/configuration is not under the density $l(x)$ for some input dimensions.

Figure 3.2 shows an example of this selection process for 1000 configurations of five dimensions. While MOTPE and TPE select the rightmost configuration in Figure 3.2a, we use the aggregation of the scores to sort the configurations and another infill criterion to suggest the candidate evaluate next (Figure 3.2b). The *Modified Expected Improvement* (MEI) [126] is used instead of the infill criterion used in TPE and MOTPE, for having shown promising results in the optimization of problems affected by heterogeneous noise [74, 133]. The MEI maximization on the set Q is formulated as follows:

$$\underset{\mathbf{q}\in Q}{\arg\max} \left[\widehat{Z}_{\min} - \widehat{Z}_{\mathbf{q}}\right] \Phi\left(\frac{\widehat{Z}_{\min} - \widehat{Z}_{\mathbf{q}}}{\widehat{s}_{\mathbf{q}}}\right) + \widehat{s}_{\mathbf{q}}\phi\left(\frac{\widehat{Z}_{\min} - \widehat{Z}_{\mathbf{q}}}{\widehat{s}_{\mathbf{q}}}\right), \; Q = \{\mathbf{t} \,|\, AS(\mathbf{t}) > 0, \mathbf{t} \in \mathbf{T}\} \tag{3.1}$$

where $\widehat{Z}_{\min}$ is the stochastic GPR prediction at $\mathbf{x}_{\min}$ (i.e. the hyperparameter configuration with the lowest sample mean among the already observed configurations), $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function respectively, the $\widehat{Z}_{\mathbf{q}}$ is the stochastic GPR prediction at configuration $\mathbf{q}, \mathbf{q} \in Q$, and $\widehat{s}_{\mathbf{q}}$ is the ordinary GPR standard deviation for that configuration [158].



(a) (b)

Figure 3.2: Infill point selection with aggregated scores (a) The log-likelihood ratio of $l(x)$ and $g(x)$ obtained for each dimension i of a set of 1000 samples obtained from density $l(x)$. (b) Aggregated score to consider all the dimensions at the moment of infill criterion maximization. Then, the MEI helps to decide which of those points with a positive aggregated score to evaluate next.

The search using MEI focuses on new points located in promising regions (i.e., with low predicted responses; recall that we assume that the scalarized objective needs to be

minimized), or in regions with high metamodel uncertainty (i.e., where little is known yet about the objective function). Consequently, the sampling behavior automatically trades off exploration and exploitation of the configuration search space.

Once a new hyperparameter configuration has been selected as an infill point, the ML algorithm is trained using this configuration hyperparameters, yielding (again) noisy estimates of the performance measures (Line 19). Following our infill strategy, we choose the configuration for which we expect the larger improvement in the scalarized objective function (the one that maximizes the MEI), among the configurations that are likely to be non-dominated (given the candidate sampling inherited from MOTPE). The new hyperparameter configuration is going to be considered in the next iteration to update both GPR and MOTPE models, and a new combination of weights is employed to scalarize the objectives to explore different areas in the Pareto front. The optimization continues until a stopping criterion is met, such as the number of iterations, or until the budget for the number of function evaluations is exhausted.

3.3 Numerical simulations

We evaluate the performance of the proposed algorithm for solving multi-objective optimization problems (GP_MOTPE), comparing the results with those that would be obtained by ParEGO [82] (with MEI as acquisition function instead of EI) and MOTPE individually. First, we analyze the performance on three well-known bi-objective problems (ZDT1, WFG4, and DTLZ7 with input dimension $d = 5$; see [68]), to which we add artificial heterogeneous noise (as in [59]). More specifically, we obtain noisy observations $\widetilde{y}_j^{(r,i)} = y_j^{(i)} + \varepsilon^{(i)},\ j = \{1, \ldots, m\}$, with $\varepsilon^{(i)} \sim \mathcal{N}(0, \tau_j^{(i)})$. The standard deviation of the noise (τ_j) varies for each objective between $0.01 \times \Omega_j$ and $0.5 \times \Omega_j$, where Ω_j is the range of objective j. In between these limits, τ_j decreases linearly with the objective value: $\tau_j = a_j(y_j + b_j), \forall j \in \{1, \ldots, m\}$, where a and b are the linear coefficients obtained from the noise range [75].

Table 3.1: Details of the ML datasets

Dataset	ID	Inst. (Feat.)	Dataset	ID	Inst. (Feat.)
Balance-scale	997	625 (4)	Delta_ailerons	803	7129 (5)
Optdigits	980	5620 (64)	Heart-statlog	53	270 (13)
Stock	841	950 (9)	Chscase_vine2	814	468 (2)
Pollen	871	6848 (5)	Ilpd	41945	583 (10)
Sylvine	41146	5124 (20)	Bodyfat	778	252 (14)
Wind	847	6574 (14)	Strikes	770	625 (6)

In a second experiment, we test our proposal on 12 OpenML datasets, shown in Table 3.1. We optimize five hyperparameters for a simple (one hidden layer) Multi-Layer Perceptron (MLP), two for a support vector machine (SVM), and five for a Decision Tree (DT) (Table 3.2). In each experiment, the goal is to find the HPO configurations that minimize classification error while simultaneously maximizing recall. We opt to consider these two metrics because they show an evident trade-off. Indeed, if True Positive instances are classified as False Negative this will increase the error while reducing the recall. On the other hand,

if the False Negatives are reduced the classification error will be reduced while the recall is increased. Hence, the conflicting objectives. Other pairs of conflicting objectives could have been chosen, such as precision versus recall (the two components of the well-known F1-score, which is a popular quality measure for classification problems with unbalanced datasets) [40, 149] or classification error versus training time [130, 66] (for expensive-to-train problems). In all experiments, we used 20% of the initial dataset as the test set and the rest for HPO. We evaluate each hyperparameter configuration by applying stratified *k-fold cross-validation* ($k = 10$).

Table 3.2: Configuration space of the ML algorithms

HP	Description	Type	Range
Multilayer Perceptron (MLP)			
max_iter	Iterations to optimize weights	Int.	$[1, 1000]$
neurons	Number of neurons in the hidden layer	Int.	$[5, 1000]$
lr_init	Initial learning rate (10^{lr_init})	Int.	$[1, 6]$
b1	First exponential decay rate	Real	$[10^{-7}, 1]$
b2	Second exponential decay rate	Real	$[10^{-7}, 1]$
Support Vector Machine (SVM)			
C	Regularization parameter	Real	$[0.1, 2]$
kernel	Kernel type to be used in the algorithm	Cat.	[linear, poly, rbf, sigmoid]
Decision Tree (DT)			
max_depth	Maximum depth of the tree. If 0, then *None* is used	Int.	$[0, 20]$
mss	Minimum number of samples required to split an internal node	Real	$[0, 0.99]$
msl	Minimum number of samples required to be at a leaf node	Int.	$[1, 10]$
max_f	Features in the best split	Cat.	[auto, sqrt, log2]
criterion	Measure the quality of a split	Cat.	[gini, entropy]

We used a fixed, small number of iterations (100) as a stopping criterion in all algorithms. This keeps optimization time low and resembles real-world optimization settings where limited resources (e.g., time, memory) may exist. The optimization of the acquisition function in Bayesian optimization tends to be non-trivial, as the function is often non-linear, non-convex, and multimodal [153]. In this work, we use the *Particle Swarm Optimization* (PSO) metaheuristic to find the infill point that maximizes MEI (i.e., the fitness function of this inner optimization) in ParEGO. Our choice is motivated by the good performance and low computational time observed in other studies with high-dimensional search space [156]. On the other hand, the infill point that maximizes the acquisition function for the other two algorithms (MEI for GP_MOTPE and $EHVI_\gamma$ for MOTPE) is selected from a candidate set obtained from density $l(\mathbf{x})$. This is motivated by the MOTPE's assumption (and subsequently for GP_MOTPE) that the current best solution can be improved by sampling a point that follows the distribution in the input space of the best-observed points. Therefore, no metaheuristic or similar algorithm is required for this inner optimization. Table 3.3 summarizes the rest of the parameters used in the experiments.

Table 3.3: Summary of the parameters for the experiments

Setting	Problem	ParEGO	MOTPE	GP_MOTPE
Initial design	Test functions	Latin Hypercube sampling: $11d - 1$		
	HPO	Random sampling: $11d - 1$		
Replications	Test functions	50		
	HPO	10 (cross-validation folds)		
Acquisition function		MEI	EHVI_γ	MEI
Acquisition function optimization		**PSO***	Maximization on a candidate set	
Number of candidates to sample		-	$c = 1000$	
Splitting parameter		-	$\gamma = 0.3$	
Kernel		Gaussian	-	Gaussian

* PSO algorithm (Pyswarm library): swarm size = 300, max iterations = 1800, cognitive parameter=0.5, social parameter=0.3, and inertia=0.9

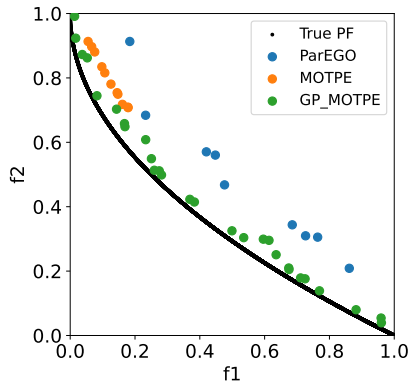
3.4 Results

Figure 3.3a, 3.3c, and 3.3e show the Pareto front found by each algorithm at the end of one of the macro-replications. Although the algorithms obtain a very good approximation of the true Pareto front, it seems that GP_MOTPE is the most successful in finding a set of well-distributed solutions on the Pareto front.

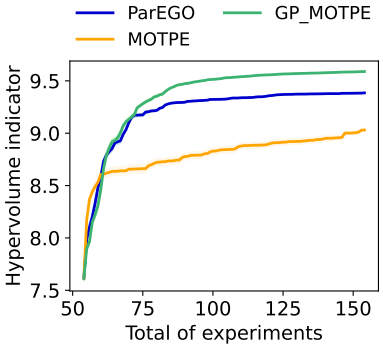
Figure 3.3b, 3.3f, and 3.3d show the evolution of the hypervolume indicator during the optimization of the test functions. The combined algorithm GP_MOTPE yields a large improvement over both ParEGO and MOTPE algorithms for the ZDT1 and DTLZ7 functions, reaching a superior hypervolume already after a small number of iterations. It was also observed that for ZDT1 and DTLZ7, the standard deviation (around the mean of 13 macro-replications) on the final hypervolume obtained by ParEGO (ZDT1: 0.0203, DTLZ7: 0.1294) and GP_MOTPE (ZDT1: 0.0082, DTLZ7: 0.0715) is small, which indicates that a Pareto front of similar quality is obtained regardless of the initial design. MOTPE, by contrast, shows higher variability in the hypervolume results at the end of the optimization of those functions (ZDT1: 0.1141, DTLZ7: 0.6154). For the concave Pareto front of WFG4, MOTPE provides the best results, while GP_MOTPE still outperforms ParEGO.

ParEGO uses a scalarization function to transform the MOO problem and in each iteration of the algorithm, different weights are selected for the scalarization. Although all points of the true Pareto front can (in principle) be found by changing the weights of the Augmented Chebychev scalarization function [42], it seems that the reduced budget of evaluations yielded the poor results of ParEGO in WFG4. Furthermore, the small gain in hypervolume may also be the result of the internal optimization algorithm getting stuck on local optima. To sum up, just considering the sampling mechanism of MOTPE to suggest the next point to evaluate may alleviate the aforementioned issues.

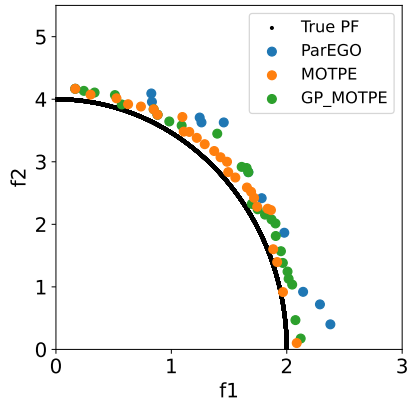
The second experimentation, conducted for HPO in the 12 OpenML datasets, did not highlight significant differences between the hypervolume obtained with GP_MOTPE, ParEGO, and MOTPE (Wilcoxon signed-rank test with Bonferroni correction; GP_MOTPE versus ParEGO: $p\_value = 0.479 > 0.05$, GP_MOTPE versus MOTPE: $p\_value = 0.987 > 0.05$, and ParEGO versus MOTPE: $p\_value = 0.295 > 0.05$). However, GP_MOTPE had the lowest average rank in the validation set, indicating that on average, the Pareto front obtained with our algorithm tends to be better than those found by ParEGO and MOTPE individually, yielding a larger hypervolume. Table 3.4 shows the average rank of the optimization algorithms according to the hypervolume indicator.



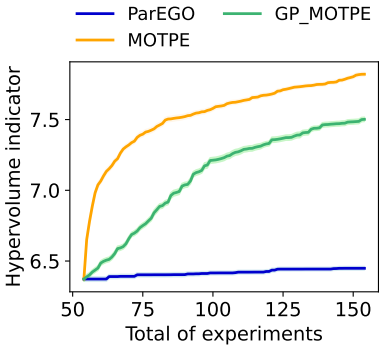
(a) ZDT1



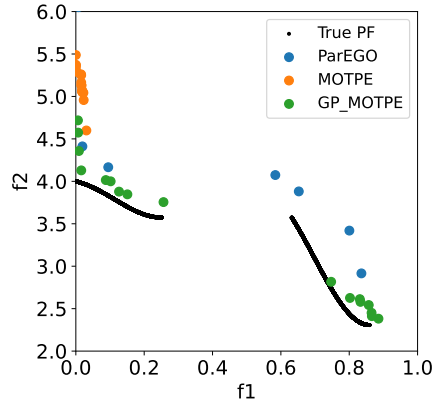
(b) ZDT1 $ref = [1, 10]$



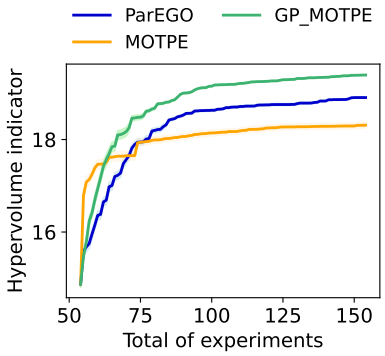
(c) WFG4



(d) WFG4 $ref = [3, 5]$



(e) DTLZ7



(f) DTLZ7 $ref = [1, 23]$

Figure 3.3: Optimization results of test analytical functions. (Left column) Observed Pareto front (PF) obtained at the end of a single macro-replication. The function value uncertainty of each solution is shown by a shaded ellipse and reflects the $mean \pm 1std/\sqrt{50}$ of the simulation replications. (Right column) Hypervolume evolution during the optimization of the analytical test functions. Shaded area represents $mean \pm 1std/\sqrt{13}$ of 13 macro-replications. Sub-captions contain the reference point used to compute the hypervolume indicator.

Table 3.4: Average rank (given by the mean hypervolume of 13 macro-replications) of each algorithm. The lower the better.

	Validation set
ParEGO	2.125
MOTPE	1.9861
GP_MOTPE	1.8889

Once the Pareto-optimal set of HP configurations has been obtained on the validation set, the ML algorithm (trained with those configurations) is evaluated on the test set. The difference between the hypervolume values obtained from the validation and test set can be used as a measure of reliability: in general, one would prefer HP configurations that generate a similar hypervolume in the test set. Figure 3.4 shows that the difference between both hypervolume values is lower when GP_MOTPE is used, for all ML algorithms and datasets. In general, MOTPE and GP_MOTPE tended to obtain solutions with lower hypervolume differences (higher rank) than the results obtained with ParEGO. The Wilcoxon signed-rank test with Bonferroni correction (as recommended by [13] when comparing ranks of multiple algorithms) detected significant differences between the results obtained with ParEGO and GP_MOTPE ($p\_value = 0.023 < 0.05$), yet no significant difference between GP_MOTPE and MOTPE ($p\_value = 0.94 > 0.05$) and between MOTPE and ParEGO ($p\_value = 0.065 > 0.05$). The results evidence once more the superiority of our proposed approach over ParEGO; there is no clear statistical evidence, though, that it is performing better than MOTPE. Appendix A.1 details the hypervolume differences and the individual ranks per each dataset and ML algorithm.

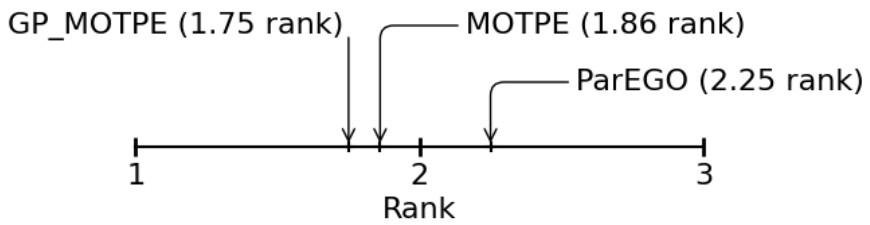


Figure 3.4: Comparison of the optimization algorithms according to the difference between the hypervolume computed using the HP evaluation in the validation set and then evaluated with the test set. The order relationship was determined by analyzing the difference between the hypervolume computed using the validation set and the hypervolume computed using the test set. The lower the rank the better.

It is somehow surprising that the combined GP_MOTPE algorithm does not always obtain an improvement over the individual MOTPE and ParEGO algorithms. By combining both approaches, we ensure that we select configurations that (1) have a high probability to be non-dominated (according to the candidate selection strategy), and (2) have the highest MEI value for the scalarized objective. In the ParEGO algorithm, (1) is neglected, which increases the probability of sampling a non-Pareto optimal point, especially at the start of the algorithm. In the original MOTPE algorithm, (2) is neglected, which may cause the algorithm to focus too much on exploitation, which increases the probability of ending up in

a local optimum.

3.5 Concluding note

A new algorithm (GP_MOTPE) was proposed for multi-objective HPO of ML algorithms. This algorithm combines the predictor information (both predictor and predictor uncertainty) obtained from a GPR model with heterogenous noise, and the sampling strategy performed by Multi-objective Tree Parzen Estimators (MOTPE). In this way, the algorithm should select new points that are likely to be non-dominated and expected to cause the maximum improvement in the scalarized objective function.

The performed experiments report that our approach performed relatively well for the analytical test functions of the study. It appears to outperform the pure GP algorithm in all analytical instances. Yet, it does not always outperform the original MOTPE algorithm. In the HPO experiments, GP_MOTPE shows the best average rank w.r.t. the hypervolume computed on the validation set. In addition, it showed good reliability properties (small changes in hypervolume when the ML algorithm is evaluated on the test set). Since our approach outperforms the pure GP-based optimization algorithm (which used PSO to maximize the infill criterion), it is useful in its own right, as the optimization of infill criteria is known to be challenging. As shown, a candidate set can be generated using MOTPE, and the point to be evaluated next suggested from this set (from these first results) appears to yield superior results when combined with GP-based metamodeling and another acquisition function.

Note that GP_MOTPE uses the current information w.r.t. the observed hyperparameter configurations to propose a *candidate set* of next points to evaluate, by sampling candidate solutions per dimension (using univariate kernel density estimation) and randomly combining the resulting input coordinates into input vectors. The choice for the next combination to evaluate is then made by exhaustively evaluating the acquisition function on this finite set, and selecting the best. This sampling approach is the same as in the original MOTPE algorithm. Algorithms such as ParEGO, on the other hand, usually apply a metaheuristic to optimize the acquisition function over the search space. Metaheuristics have their own complexity, though, which influences the overall execution time of the optimization. They also require the choice of additional parameters, that should ideally be tuned, and even retuned during the BO iterations. In our experiments, we did not tune the PSO implemented in ParEGO, and the results indicate that the resulting quality of the solutions is not superior to the ones found by the finite sampling approach in GP_MOTPE. We also expect that the use of multivariate kernel density estimation may still further improve the effectiveness of the results obtained by GP_MOTPE.

— CHAPTER 4 —

GOOD AND BAD

Tree Parzen Estimators with performance variability

TREE-*structured* PARZEN ESTIMATORS (TPE) have demonstrated their ability to find hyperparameter configurations with efficient evaluation budgets. However, as it is common in HPO procedures, TPE ignores that the algorithm's expected performance, for any given HPO configuration, varies according to a number of factors: different datasets to train, the stochastic nature of the learning algorithm, etc. Thus, this performance is *noisy*. The previous chapter demonstrated that accounting for the performance variability can suggest better HP configurations in a multi-objective optimization setting. However, accounting for this variability was done by modeling the objective with a GPR for noisy observations, which naturally handles the heteroscedastic noise. However, is it possible to handle noisy objectives directly in TPE and leave out the computational load introduced by GPR? Building on the TPE algorithm proposed by [16] for single-objective HPO, Section 4.1 proposes a strategy to account for this performance variability. Section 4.2 describes the experimental study used to evaluate our approach and Section 4.3 discusses the key findings of such experimentation.

4.1 Adjusted TPE for stochastic objectives

Traditional HPO algorithms start from observations[1] $\mathbf{\Gamma} = \{\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times r}\} = \left\{(\mathbf{x}^{(1)}, \overline{y}^{(1)}), \ldots, (\mathbf{x}^{(n)}, \overline{y}^{(n)})\right\}$ where $\mathbf{x}^{(i)}$ is a hyperparameter configuration of dimension d and $\overline{y}^{(i)}$ is the mean expected performance of the algorithm (having evaluated r times the performance), observed after the ML algorithm has been trained/validated using hyperparameter configuration i.

As mentioned in Chapter 2, TPE neglects the fact that the $\overline{y}^{(i)}$ values are *noisy*. Indeed, they typically result from a k-fold cross-validation protocol, yielding a random sample of k performance values per HP configuration considered. The overall performance of the ML algorithm, trained with the given HP configuration, is then commonly reported as the mean performance across the different splits. Figure 4.1 shows an example of HP selection using a cross-validation protocol with $k = 5$.

Notice that the resulting sample mean $\overline{y}^{(i)}$ is also a random variable of unknown distribution. Then, the set $\mathbf{\Gamma}$ can be reformulated as $\mathbf{\Gamma} = \{(\mathbf{x}^{(1)}, PDF(\mathbf{y}^{(1)})), \ldots, (\mathbf{x}^{(n)}, PDF(\mathbf{y}^{(n)}))\}$, where the performance observed for each configuration $\mathbf{x}^{(i)}$ follows a

[1] This chapter is dedicated to single-objective HPO. So, the columns of matrix $\mathbf{Y}$ refers to the replications of only one objective.

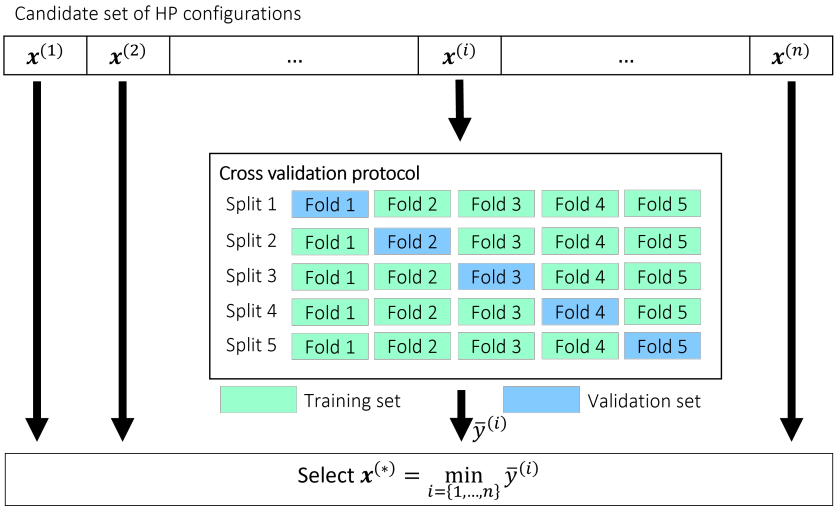


Figure 4.1: Illustration of a 5-fold cross-validation protocol for selecting the best HP configuration (with the lower error) from a candidate set (obtained, for instance, with Grid search). For cross-validation purposes, the training set is split into k folds (here: $k = 5$ folds or split). The chosen ML algorithm is trained and validated on each split for a given HP configuration, yielding k estimates of algorithm performance. Traditionally, the average performance across splits is then used as an estimator for the overall algorithm performance, given this HP configuration.

continuous distribution given by the *probability density function* (PDF) of the observed performance. The value y^* continues to be calculated as the quantile γ of the observed $\overline{y}$ performances. Figure 4.2 shows the uncertain evaluation of a toy function (simulating an unknown performance function in HPO). Here, the performance should not be reduced to a single value (i.e., the mean value) since a point can be considered as good and bad at the same time. For instance, we observed that point C in Figure 2.3 was considered as good point but after analyzing its PDF (Figure 4.2), point C is also a bad point to some extent. The same can be observed for point A.

Taking into account this variability on the performance implies that a given configuration $\mathbf{x}^{(i)}$ can no longer be classified with certainty as part of X_l versus X_g (line 2 of Algorithm 2.1). Instead, each configuration has a *probability* of yielding a "good" versus a "bad" performance outcome. More formally, the cumulative distribution function (CDF) of the performance distribution yields the probability of being "good" for an HP configuration i as $CDF(\mathbf{x}^{(i)}) = Pr[\overline{y}^{(i)} < y^*]$. Likewise, the survival function yields the probability of being "bad" for an HP configuration i as $Sr(\mathbf{x}^{(i)}) = Pr[\overline{y}^{(i)} > y^*]$ or $1 - Pr[\overline{y}^{(i)} < y^*]$. Figure 4.3 shows the performance distribution of four points (A, B, C y D in Figure 4.2). The original splitting procedure of TPE will only select points B and C to belong to X_l, according to their mean performance (less than y^*). However, point A is also likely of being "good", suggesting that they should be considered in some way during $l(x)$ estimation. Similarly, point C was not considered in X_g, but we can see that the performance distribution suggests that this point may result in a bad performance. Indeed, the CDF of these points (Figure 4.3b) establishes that points A and C are slightly good and bad respectively.

To handle noisy objectives with TPE, we consider that *all* observed data points are

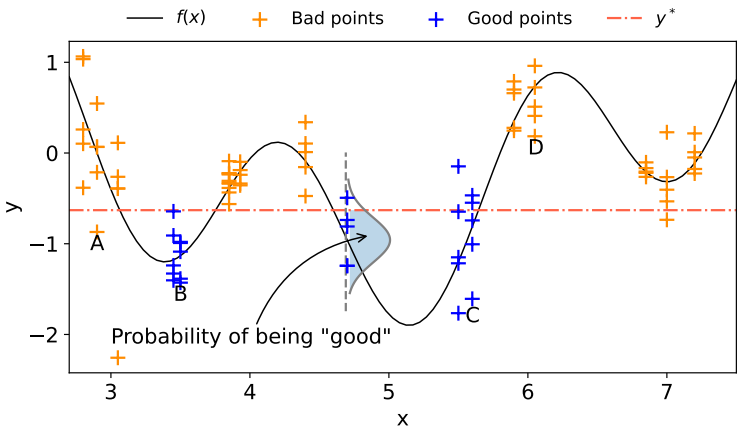
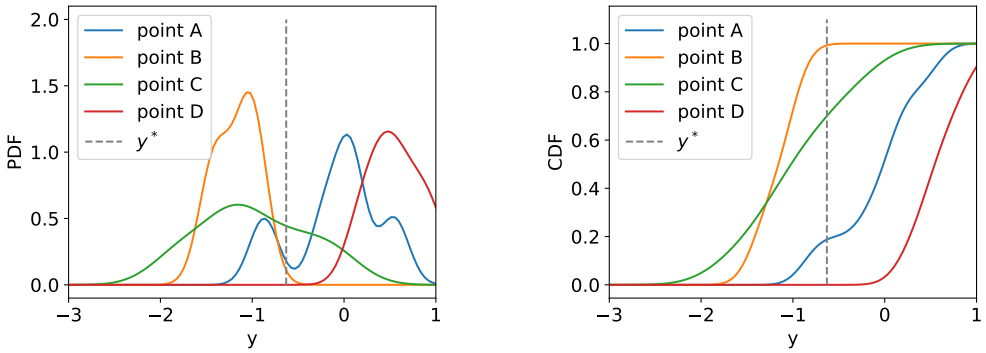


Figure 4.2: Noisy function evaluation for $f(x) = sin(x)+sin(\frac{10}{3}x)$ (black line) in the range $[2.7, 7.5]$. The objective value of any configuration $\mathbf{x^{(i)}}$ is described by a *probability density function*. The colors of each point correspond to the division observed in Figure 2.3 for good and bad points. However, each point could be considered bad and good to some extent when their response is noisy. The probability of being good is given in this case by the *cumulative density function* $CDF(\mathbf{x}^{(i)}) = Pr[\overline{y}^{(i)} < y^*]$.



(a) **(b)**

Figure 4.3: Example of probability and cumulative density function for the noisy performance of four points (Highlighted in Figure 4.2). Here, the higher the CDF value the higher the probability that the performance observed for a point will be good.

relevant for estimating $l(x)$ and $g(x)$, and their probabilities of being "good" and "bad" indicate their influence on estimating such density functions. Hence, the kernel density estimators are weighted with the respective probabilities. Equations 4.1 and 4.2 formalize the resulting weighted kernel density estimation.

$$l(x) = \frac{1}{nh}\sum_{i=1}^{n} w_i^l k\left(\frac{x - x^{(i)}}{h}\right), \sum_{i=1}^{n} w_i^l = 1, \tag{4.1}$$

$$g(x) = \frac{1}{nh}\sum_{i=1}^{n} w_i^g k\left(\frac{x - x^{(i)}}{h}\right), \sum_{i=1}^{n} w_i^g = 1, \tag{4.2}$$

where the weights $w_i^l = \frac{CDF_i(y^*)}{\sum_{j=1}^{n} CDF_j(y^*)}$ and $w_i^l = \frac{Sr_i(y^*)}{\sum_{j=1}^{n} Sr_j(y^*)}$ correspond to the normalization of the aforementioned probabilities of n configurations, and h is the bandwidth of the kernel $k(\cdot)$. The value of h was set to the greater of the distances to the left and right neighbor, but clipped to remain in a reasonable range [16]. Figure 4.4 shows the weighted KDE obtained for the TPE example shown in Figure 2.3. Here, the weights obtained with Equation (4.1) and Equation (4.2) determine the influence of each point on $l(x)$ and $g(x)$ estimation respectively. Therefore, the subset selection performed by Algorithm 2.1 (Line 4) is no longer needed in our proposal since all the points are considered now. Algorithm 4.1 details the adjusted TPE algorithm to accommodate the performance variability of HP configurations.

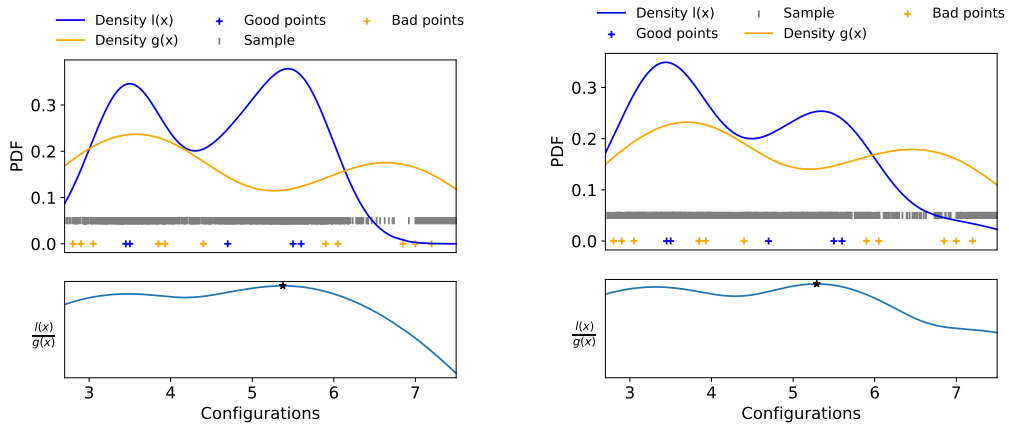


Figure 4.4: (Weighted) Kernel Density Estimation (KDE) in TPE. (a) The original TPE uses a subset of the observed points to perform KDE in the input space (see Figure 2.3). (b) Weighted KDE uses the probability of being "good" and "bad" as weights to estimate the influence of each point on $l(x)$ and $g(x)$ estimation.

4.2 Experimental settings

We focus on the optimization of five hyperparameters (Table 4.2) of a recurrent neural network with one hidden layer. Table 4.1 shows the details of the OpenML datasets used to test the proposed algorithm.

Table 4.1: OpenML datasets used in the experimentation

Dataset	OpenML ID	Features	Instances
Balance-scale	997	4	625
Optdigits	980	64	5620
Stock	841	9	950
Heart-statlog	53	13	270
Ilpd	41945	10	583

Table 4.3 summarizes the parameters used in the experiments. We are unaware of any guidelines regarding the size of the initial design for HPO. Following Jones et al. [78], we

Algorithm 4.1 Tree Parzen Estimators for noisy single-objective optimization

Require: $D = \{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times 1\times r})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, γ: quantile parameter, c: number of candidates per iteration, r: number of simulation replications

for $i \leftarrow 1, \ldots, N$ **do**
  $\mathbf{T}_{1\times d} \leftarrow \{\}$
  for $j \leftarrow 1, \ldots, d$ **do**
    $W^l = \left[\frac{CDF_t(y^*)}{\sum_{k=1}^{n} CDF_k(y^*)}\right]_{n\times 1},\ t = \{1, \ldots, n\} \wedge p(y < y^*) = \gamma$
    $W^g = \left[\frac{Sr_t(y^*)}{\sum_{k=1}^{n} Sr_k(y^*)}\right]_{n\times 1},\ t = \{1, \ldots, n\} \wedge p(y < y^*) = \gamma$
    $l_j(x) \leftarrow \text{KDE}(D_j, W^l)$ ▷ Weighted KDE with probability of being "good"
    $g_j(x) \leftarrow \text{KDE}(D_j, W^g)$ ▷ Weighted KDE with probability of being "bad"
    $\mathbf{C}_{1\times c} \leftarrow \left\{x^{(k)} \sim l_j(x) \mid k = 1, \ldots, c\right\}$ ▷ Sample candidates in dimension j
    $x^* \leftarrow \arg\max_{x\in\mathbf{C}} \text{EI}_{y^*}(l_j(x),\ g_j(x),\ x)$ ▷ Acquisition function maximization
    $\mathbf{T} \leftarrow \mathbf{T} \cup x^*$
  end for
  $\mathbf{Y}^* \leftarrow \text{SIMULATE}(\mathbf{T})$ ▷ Expensive evaluation of $\mathbf{T}$
  $D \leftarrow D \cup \{(\mathbf{T},\ \mathbf{Y}^*)\}$
end forreturn the best-observed configuration contained in D

set the number of initial design points to $n_0 = 11d - 1$, with d the dimension of the search space, as is common in GPR-based optimization. We use a random sampling method (using the python package ConfigSpace [99]) since Latin Hypercube sampling does not work well with non-real values [106, 148]. We use a limited number of iterations as a stopping criterion in our experiments. This resembles real-world optimization settings where limited resources (e.g., time) may exist.

Table 4.2: Setup of hyperparameters in the HPO experiments

HP	Description	Type	Domain
max_iter	Maximum number of iterations to optimize the weights	Int.	$[1, 1000]$
neurons	Number of neurons in the hidden layer	Int.	$[5, 1000]$
lr_init	Initial learning rate (10^{lr_init})	Real	$[1, 6]$
b1	Exponential decay rate for estimates of first moment vector in adam solver (10^{b1})	Real	$[0, 7]$
b2	Exponential decay rate for estimates of second moment vector in adam solver (10^{b2})	Real	$[0, 7]$
act	Activation function for the hidden layer	Cat.	*relu*
solver	The solver for weight optimization	Cat.	*adam*
layer	Number of hidden layers	Int.	1

We opt for *stratified* cross-validation as validation protocol, to ensure that relative class

frequencies are approximately preserved in each train and validation split. In our experiments, we consider 20% of the initial dataset as a test set and use the remainder for HPO (through a cross-validation protocol as observed in Figure 4.1). The goal is to find the hyperparameter configuration that minimizes the mean classification error in the validation set of each of the classification problems.

Table 4.3: Summary of the parameters for TPE and the proposed modification

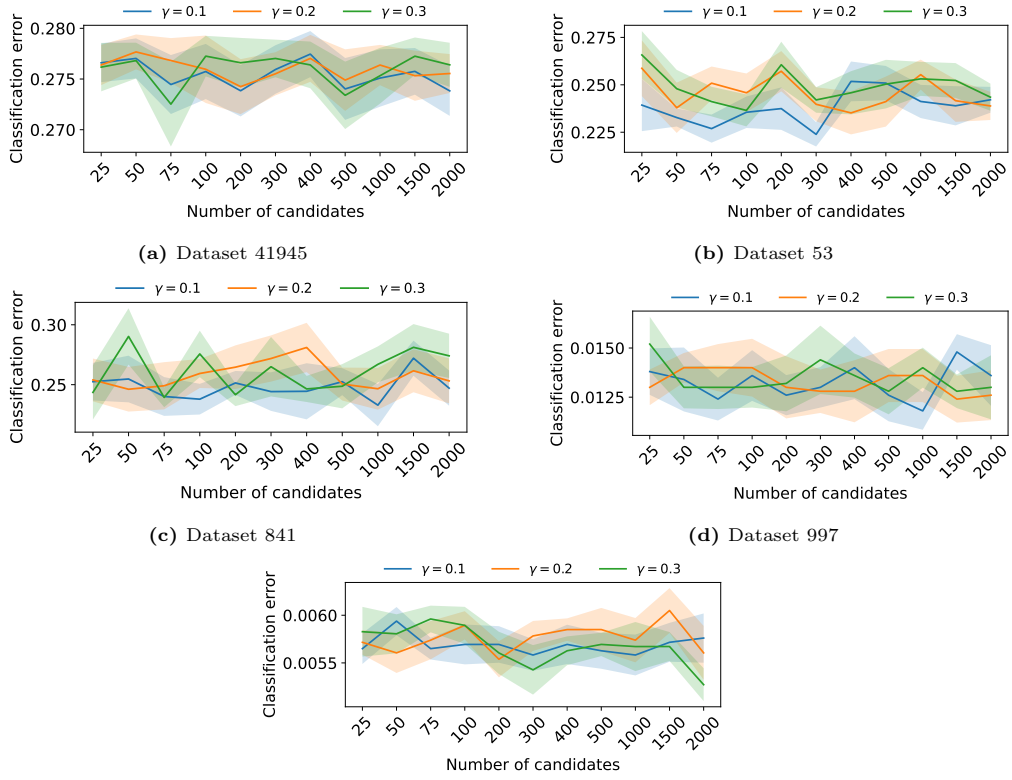
Parameter	Setting
Design space size	$11d - 1$
Replications (folds in the validation protocol)	5
Iterations	50
Candidates to sample per iteration	[25, 50, 75, 100, 200, 300, 400, 500, 1000, 1500, 2000]
TPE (γ)	$[0.1, 0.2, 0.3]$

4.3 Results

Figure 4.5 shows an analysis of the influence of the parameter γ and the number of candidates when TPE considers the noise of the observations. The parameters were varied in the discrete set $\gamma = \{0.1, 0.2, 0.3\}$ and $c = \{25, 50, 75, 100, 200, 300, 400, 500, 1000, 1500, 2000\}$. The mean classification error reached at the end of optimization in 10 macro-replications was used for comparison purposes. Recall that small values of γ will force the algorithm to focus on sampling new configurations from a probability distribution obtained for the best-observed input configurations. The results did not show a clear best setting for all the datasets. However, it seems that $\gamma = 0.1$ was consistently amongst the settings with the lowest classification error. As for the number of candidates, the results were heterogeneous. It was obvious for Dataset 53 that sampling more than 300 candidates does not help with the optimization if $\gamma = 0.1$. The opposite is observed in Dataset 980 where increasing the number of candidates and γ parameter shows the best results.

We relied on hierarchical/agglomerative clustering to select the best settings for TPE with noisy settings, instead of selecting them visually. The data-driven decision avoids specifying a threshold value to select the best single setting and gives us a global analysis of the behavior of the other parameter configurations. The clustering is made based on the Euclidian distance between the mean classification error of the settings at the end of 10 macro-replications. We generated 3 clusters; Cluster 1 with the settings with the lowest classification error (the best), Cluster 3 with setting with the largest classification error (the worst), and Cluster 2 with "medium" performances (the settings were not the worst but they were not the best either). Figure 4.6 shows a frequency analysis of the settings included in Cluster 1 and Appendix B shows the resulting dendrograms of the clustering algorithm.

As we can see, $[\gamma = 0.1, c = 75]$ was the most frequent combination with the lowest classification error in the HPO performed for each dataset. Dataset 980 was the only case where this combination of parameters was not in Cluster 1, which makes sense since for this dataset the best results (of TPE with noisy observations) were observed for $\gamma = 0.3$ and sampling more than 300 candidates (Figure 4.5e).



(a) Dataset 41945

(b) Dataset 53

(c) Dataset 841

(d) Dataset 997

(e) Dataset 980

Figure 4.5: Sensitivity analysis of the parameters γ and number of candidates when TPE considers the noise of the observations. The mean classification error reached at the end of optimization in 10 macro-replications was used for comparison purposes. Likewise, shaded area represents $mean \pm \frac{1std}{\sqrt{10}}$ of 10 macro-replications

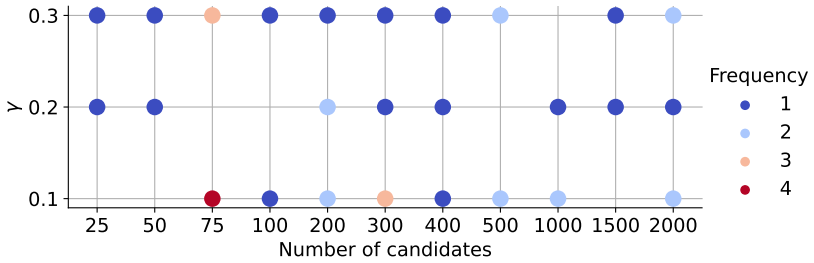
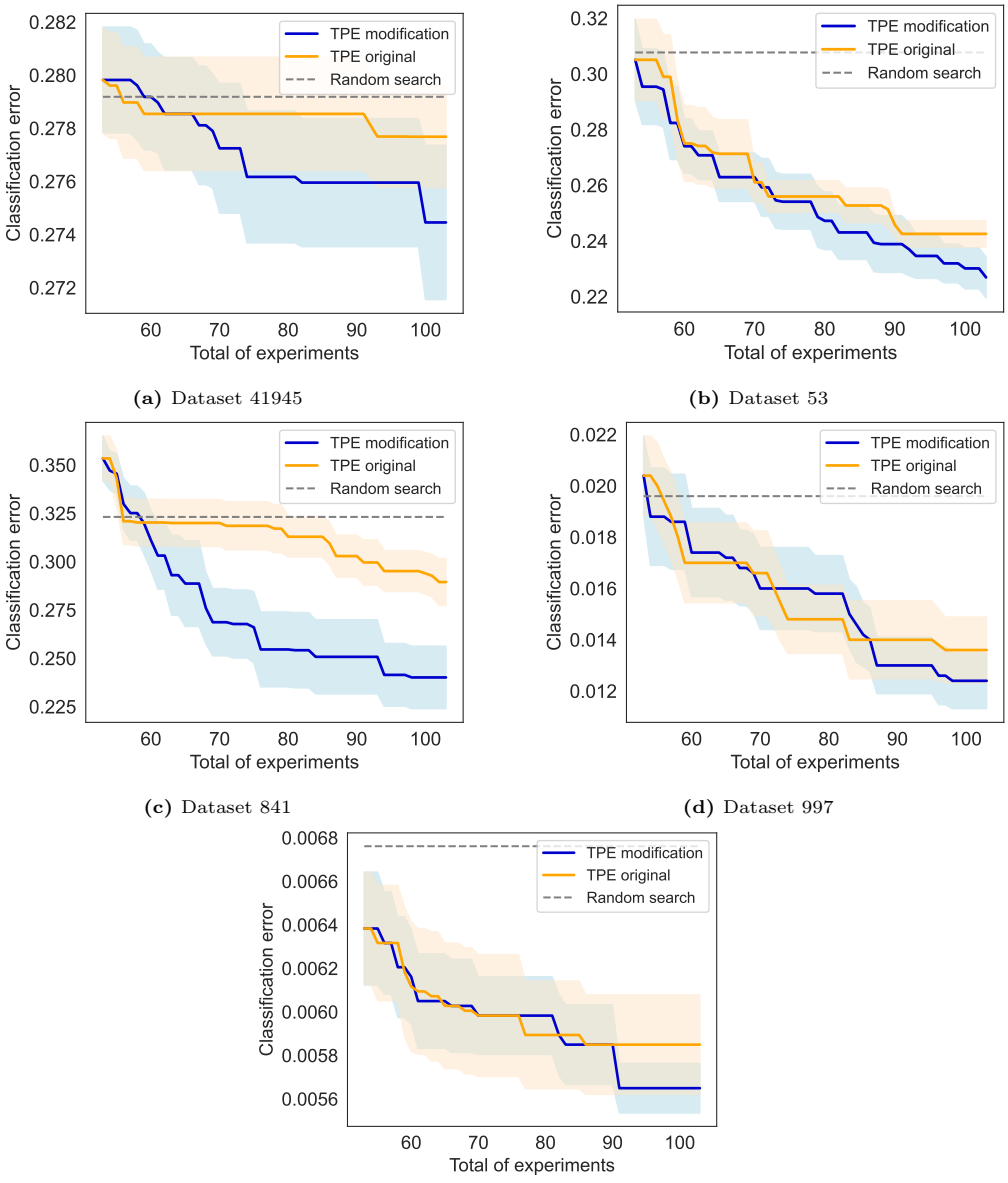


Figure 4.6: Frequency analysis of the parameters γ and number of candidates with the lowest classification error at the end of the optimization. Empty intersections represent settings that were never the best.

Figure 4.7 shows the evolution of the mean classification error based on 10 macro-replications and using parameters $[\gamma = 0.1, c = 75]$ both for TPE with deterministic and

noisy observations. Additionally, we included the results of the lowest classification error found by a random search (as a baseline) using the same budget described in Section 4.2.



(a) Dataset 41945

(b) Dataset 53

(c) Dataset 841

(d) Dataset 997

(e) Dataset 980

Figure 4.7: Evolution of the mean classification error based on 10 macro-replications and using parameters $[\gamma = 0.1, c = 75]$. Shadowed area corresponds to the $mean \pm \frac{std}{\sqrt{10}}$ of 10 macro-replications.

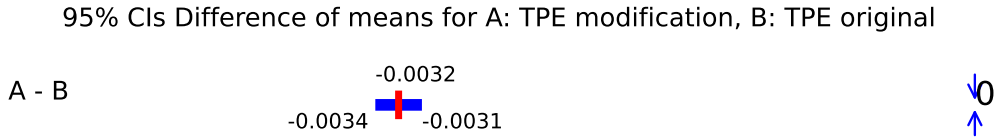
The proposed modification to TPE is able to find hyperparameter configurations that

yield lower classification errors in the datasets analyzed. Note how our proposal behaved similarly to the original TPE during most iterations in Dataset 980. Just before the budget was exhausted, TPE for noisy observations was able to suggest a better configuration than had been found until about iteration 82. Indeed, if we analyze the last 20 iterations of the optimization in Dataset 980, we can see that at some point both algorithms cannot find any hyperparameter configuration with an improvement in the classification error. This may be because the algorithm reached a local minimum, and suggesting configurations in the region where the best configurations were observed is not enough to escape it. However, because our proposal uses the information on all the points (in terms of how good they are) to estimate the densities in the input space, we have a small chance of going beyond that area where the best points were observed. For example, that may be the reason for our algorithm to find a better hyperparameter setting at iteration 90.

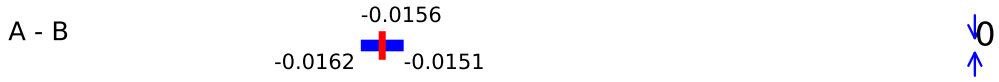
These results are corroborated by analyzing the confidence interval (CI) generated by the differences between the classification error of the configurations suggested by the algorithms at the end of the optimization (Figure 4.8). This type of CI will always agree with the 2-sample t-test. In addition to providing a simple visual assessment, the confidence interval of the difference presents crucial information that a p-value does not provide, such as which algorithm consistently suggested a better configuration. When looking at the CIs of the pairwise differences per macro-replication between both algorithms on each dataset, we can see that the proposed modification to TPE was consistently better than the original TPE (without considering the performance variability of the ML algorithm evaluation). Furthermore, the Wilcoxon signed-rank test with Bonferroni correction showed significant differences between the results of both algorithms ($p\_value = 0.0029 < 5\%$).

4.4 Concluding note

The experimentation confirmed that considering a hyperparameter evaluation's performance uncertainty yields better ML algorithm configuration. Contrary to the original algorithm, we take into account the probability of a point being simultaneously "good" and "bad", and use these as weights for the kernel density estimators that form the density functions in the input space. In general, we perform equally well as (or better than) the original TPE at the end of the optimization and during the optimization (which is interesting when the evaluation budget is even more limited).

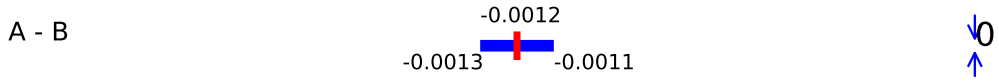


(a) Dataset 41945

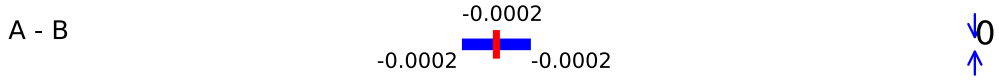


(b) Dataset 53

(c) Dataset 841



(d) Dataset 997



(e) Dataset 980

Figure 4.8: Comparison of the algorithms according to the confidence interval generated by the difference between the classification errors observed at the end of the optimization process. ci : $mean \pm t_{9,0.95} \frac{std}{\sqrt{9}}$, where $t_{n,\alpha}$ is the Student T value with n degrees of freedom and α the significance level. The proposed TPE modification is always betters than the original TPE, as the CIs are completely negative.

SECOND PART

APPLICATIONS

— CHAPTER 5 —

A GLUE TO BIND THEM ALL

Bayesian multi-objective optimization of process design parameters in constrained settings with noise: an engineering design application

Adhesive joints are increasingly used in industry for a wide variety of applications because of their favorable characteristics such as high strength-to-weight ratio, design flexibility, limited stress concentrations, planar force transfer, good damage tolerance, and fatigue resistance. Finding the optimal process parameters for an adhesive bonding process is challenging: the optimization is inherently multi-objective (aiming to maximize break strength while minimizing cost), constrained (the process should not result in any visual damage to the materials, and stress tests should not result in failures that are adhesion-related), and uncertain (measuring the same process parameters several times lead to different break strength). Real-life physical experiments in the lab are expensive to perform. Traditional evolutionary approaches (such as genetic algorithms) are then ill-suited to solve the problem, due to the prohibitive amount of experiments required for evaluation. Although Bayesian optimization-based algorithms are preferred to solve such expensive problems, few methods consider the optimization of more than one (noisy) objective and several constraints at the same time. This chapter shows the application of specific machine learning techniques (Gaussian Process Regression) to emulate the objective and constraint functions based on a *limited* amount of experimental data.[1] Section 5.1 describes the bonding process problem under study. Section 5.2 details the proposed algorithms. Section 5.3 discusses the design of experiments and Section 5.4 analyzes the main results of the experimentation.

5.1 Adhesive bonding process: problem setting

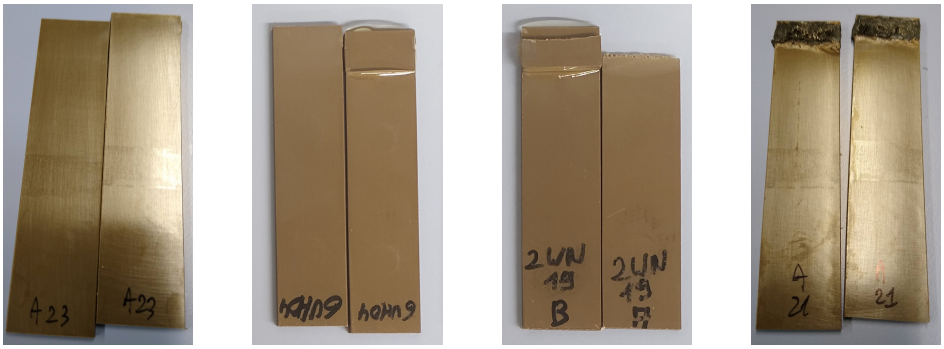
The bonding process we focus on joins two PolyPhenylene Sulfide (PPS) substrates using Araldite 2011 adhesive. Figure 5.1 shows the general procedure of the adhesive bonding process. The optimization focuses on the Plasma treatment step. The plasma treatment chemically modifies the top surface layer of the PPS substrate so that the surface energy increases, which impacts the adhesion strength (i.e., the strength of the connection between the adhesive and the substrate). In this process, six parameters play a role: (1) whether the

[1] The content of this chapter has been included in the publications *"Optimization of Plasma-Assisted Surface Treatment for Adhesive Bonding via Artificial Intelligence". In: Proceedings of the International Conference on Industrial Applications of Adhesives (2022)* [79], *"Expensive multi-objective optimization of adhesive bonding process in constrained settings". In: Proceedings of the Optimization and Learning Conference (2023)* [Accepted for publication], and *"Bayesian multi-objective optimization of process design parameters in constrained settings with noise: an engineering design application"* [Submitted].

surface is pre-processed or not (cleaning to remove dust and grease, which may prevent a good connection between the adhesive and the substrate), (2) the power setting of the plasma torch, (3) the speed at which the plasma torch moves across the samples, (4) the distance between the plasma torch nozzle and the sample, (5) the number of passes of the plasma torch over the sample, and (6) the time between the plasma treatment and the application of the glue (as the plasma effect reverses over time). The adhesion strength is very sensitive to the configuration of these parameters.

Figure 5.1: Schematic overview of the adhesive bonding process

Using lab experiments, stress tests can be performed to check the outcomes of samples that have been treated with any particular plasma parameter configuration: the lap shear strength of the sample (MPa), the failure mode (adhesive, cohesive, or substrate failure), the production cost of the sample (in euros), and the potential occurrence of visual damage (the substrates may burn when heated above their maximum allowable temperature during plasma treatment). Such physical experiments are expensive, as they require the whole process in Figure 5.1 to be performed, involving a human operator. Figure 5.2 shows different failure modes and an example of visual damage.



(a) Initial substrate **(b)** Adhesion failure **(c)** Substrate failure **(d)** Burned sample

Figure 5.2: Illustration of failure modes that may result from the stress test applied to a sample. The adhesive is applied on the substrates (a) and the failure mode can be either adhesion failure (b), substrate failure (c) or cohesive failure (not shown). In addition, visual damage might be observed after the experiment; e.g., burned sample (d)

The goal of the optimization is to set the plasma process parameters in such a way that (1) the tensile strength (TS) is maximized, (2) the production cost (PC) is minimized, and (3) adhesive failures and visual damage are avoided. As objectives (1) and (2) are in conflict, this is a *multi-objective* optimization problem. The goal is to find a set of solutions that reveal the essential trade-offs between these objectives (i.e., those solutions for which no objective

can be improved without negatively affecting the performance of any other objective) while meeting the constraints (3). Equation 5.1 formally defines this optimization problem as

$$\begin{aligned} \min \quad & [-TS(\mathbf{x}), PC(\mathbf{x})] \\ \text{s.t.} \quad & 0.5 - Pf(\mathbf{x}) \leq 0 \end{aligned} \tag{5.1}$$

where the notation $Pf(\mathbf{x})$ refers to the probability that a process configuration $\mathbf{x}$ is feasible (estimated as the fraction of replications in which the configuration resulted in a feasible outcome). As the performance evaluation is expensive, the optimization algorithm should be able to detect (nearly) Pareto-optimal solutions within a small number of experiments required; collecting large amounts of experimental data is simply financially infeasible. In the following section, we discuss how the use of Bayesian optimization to allows us to develop such a data-efficient optimization approach.

5.2 Constrained Bayesian multi-objective optimization: proposed algorithms

Each BO algorithm thus has two key elements: the type of metamodel used, and the type of acquisition function. Several acquisition functions exist (see [53] and [134] for single and multi-objective surveys respectively). Allegedly, the *expected improvement (EI)* remains one of the most commonly used ones in practice [78], and GP regressors are standard metamodels in the BO literature [152]. In this work, we propose two different algorithms for constrained multi-objective optimization using EI: cMEI-SK, and cEHVI-SK.

cMEI-SK uses the augmented Chebychev scalarization function to transform the problem into a single-objective optimization problem (Algorithm 5.1). As discussed previously, the algorithm starts with an initial set of points designed by a Latin hypercube sample, and a fixed number r of simulation replications is used to account for the uncertainty in the objectives $\mathbf{Y}$ and constraint $\mathbf{C}$. $Z_{\boldsymbol{\lambda}}^{(i)}$ corresponds to the scalarization function formalized in Equation (2.18). An independent ML model is trained (Line 7) to predict the probability that a parameter configuration is feasible (meaning that it will not entail visual damage or adhesive failure). Then, the metamodels' information is exploited to search for new process configurations.

The second algorithm we propose trains a metamodel for each objective and constraint (cEHVI-SK in Algorithm 5.2). So no scalarization function is required. In total, we train three metamodels for the adhesive optimization problem: a (stochastic) GPR to approximate the break strength objective, a (deterministic) GPR to estimate the production cost, and a (deterministic) GPR to estimate the probability of feasibility (as in Algorithm 5.1). Notice that in both algorithms, there is a preliminary step (Line 8 of Algorithm 5.1 and Line 7 of Algorithm 5.2) in which we select the better feasible observation(s) to evaluate with the metamodel and use this evaluation in the maximization of the acquisition function.

We shall notice that both MEI and EHVI are acquisition functions well-known in the BO literature. However, their application has mostly been reported in deterministic unconstrained settings [158]. A few extensions have been proposed in the literature to handle constraints [48], and to handle observational noise [132, 37]. Given that in our problem setting the feasibility of a process configuration is evaluated with physical tests, where both the

objectives and constraints not only are noisy but also of different nature (see Section 5.1), such methods cannot be used out-of-the-box. Further details on the proposed algorithms (and thus our main contributions) are given in sections 5.2.1, 5.2.2, and 5.2.3. As noted in Chapter 2, we explicitly differentiate two types of GPR models: *ordinary kriging (OK)* [78, 80] and *stochastic kriging (SK)* [6]. While any GPR model can accommodate noisy evaluations [152], OK metamodels are limited to homogeneous noise. The seminal work of Ankenman, Nelson, and Staum [6] extended OK metamodels to handle heterogeneous noise (referred to as SK metamodels). Subsequently, the algorithms presented here exploit the information extracted from both types of metamodels.

Algorithm 5.1 cMEI-SK algorithm for noisy (constrained) multi-objective optimization. GPR is the surrogate model approximating objectives (stochastic GPR) and constraints (deterministic GPR). The same number of simulation replications is assumed for each configuration

Require: $\{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times m\times r}, \mathbf{C}_{n\times 1})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, r: number of simulation replications

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: $\quad \boldsymbol{\lambda}_{1\times m} = [\lambda_1, \ldots, \lambda_m]\,,\ \sum_{i+1}^{m} \lambda_i = 1$ ▷ Random weight vector
3: $\quad [\mathbf{Z}_{\boldsymbol{\lambda}}]_{n\times 1\times r} = \left[Z_{\boldsymbol{\lambda}}^{(i)}\right],\ i = \{1, \ldots, n\}$ ▷ Chebychev scalarization
4: $\quad \overline{\mathbf{Y}}_{n\times 1} \leftarrow \left[\frac{\sum_{k=1}^{r} Z_{\boldsymbol{\lambda}_k}^{(j)}}{r}\right]_{n\times 1},\ j = \{1, \ldots, n+i-1\}$
5: $\quad \overline{\mathbf{V}}_{n\times 1} \leftarrow \left[\frac{\frac{1}{r-1}\sum_{k=1}^{r}\left[Z_{\boldsymbol{\lambda}_k}^{(j)} - \overline{y}_l^{(j)}\right]^2}{r}\right]_{n\times 1},\ j \in \{1, \ldots, n+i-1\}$
6: $\quad sGP \leftarrow$ STOCHASTIC_GP_FIT($\mathbf{X}$, $\overline{\mathbf{Y}}$, $\overline{\mathbf{V}}$) ▷ Metamodel training of the objectives
7: $\quad dGP \leftarrow$ DETERMINISTIC_GP_FIT($\mathbf{X}$, $\overline{\mathbf{C}}$) ▷ Metamodel training of the constraint
8: $\quad \mathbf{x}_{min} \leftarrow \arg\min_{\overline{y}\in\overline{\mathbf{Y}}\,|\,\overline{y}\ is\ feasible}\ \overline{y}$
9: $\quad \widehat{y}_{min} \leftarrow sGP(\mathbf{x}_{min})$
10: $\quad \mathbf{x}^* \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}}$ cMEI-SK(sGP, dGP, $\widehat{y}_{min}$, $\mathbf{x}$) ▷ Acquisition function maximization
11: $\quad \mathbf{Y}^*_{1\times r}$, $C^* \leftarrow$ SIMULATE($\mathbf{x}^*$, r) ▷ Expensive evaluation of $\mathbf{x}^*$
12: $\quad \mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{x}^*$
13: $\quad \mathbf{Y} \leftarrow \mathbf{Y} \cup \mathbf{Y}^*$
14: $\quad \mathbf{C} \leftarrow \mathbf{C} \cup C^*$
15: **end for**
return the set of non-dominated feasible solutions

5.2.1 Probability of feasibility

The *probability of feasibility* (PoF) has been well-studied in the BO literature when the constraints are deterministic [51, 33, 158]. In our problem setting, the constraint feasibility is checked physically (meaning that it will not entail visual damage or adhesive failure), which results in a classification problem with two classes: `YES` for feasible outcomes and `NO` for infeasible outcomes. Thus, for each process configuration evaluated, we have a discrete target variable $c \in \{0, 1\}$, where 1 denotes `YES` and 0 denotes `NO`. We are interested in the probability $p(c = 1 \mid \mathbf{X})$. That is, the probability of a process configuration being truly feasible is conditioned on the data collected so far.

To do this, we can train a *classification* model in Line 7 of Algorithm 5.1 (and Line 6 of Algorithm 5.2). Training a binary classifier here is actually not very helpful, since the *noise* on the constraints will affect the performance for the same input configuration (i.e., replicating the same process configuration may yield different outcomes for constraint violation). For instance, if the same input is replicated 5 times, yielding 2 `YES` and 3 `NO`, it is unlikely that training the classifier with e.g., the mode of the outcomes will yield accurate predictions. Therefore, we opt for training a regression model on the proportion of successful outcomes instead: in the example above we would use 0.4 as the expected prediction since 2 out of 5 replications were successful. Thus, for a given process configuration $\mathbf{x}^{(*)}$, we take the proportion of successful outcomes of the binary target c as

$$\bar{c}^{(*)} = \frac{\sum_{j=1}^{r^{(*)}} c_j}{r^{(*)}} \tag{5.2}$$

where r is the number of replications. We then approximate the *probability of feasibility* (PoF) of the unobserved locations by fitting an ordinary kriging model (Equation 2.5) on the target $\bar{c}$ (denoted $\widehat{y}_c(\mathbf{x})$). That is, given the observed targets $\bar{c} \in \overline{\mathbf{C}}$ at points $\mathbf{X}$, the predicted distribution of an unobserved target $c^{(*)}$ at point $\mathbf{x}^{(*)}$ is given by

$$\text{PoF}(\mathbf{x}^{(*)}) \simeq \widehat{y}_c(\mathbf{x}^{(*)}) = p(c^{(*)} = 1 \mid \mathbf{x}^{(*)}, \mathbf{X}, \overline{\mathbf{C}}) \tag{5.3}$$

Algorithm 5.2 cEHVI-SK algorithm for noisy (constrained) multi-objective optimization. GPR is the surrogate model approximating break strength (stochastic GPR), production cost (deterministic GPR), and feasibility constraint (deterministic GPR). The same number of simulation replications is assumed for each configuration

Require: $\{(\mathbf{X}_{n\times d}, \mathbf{Y}_{n\times m\times r}, \overline{\mathbf{C}}_{n\times 1})\}$: initial design, N: number of iterations, n: number of configurations in the initial design, r: number of simulation replications

1: **for** $i \leftarrow 1, \ldots, N$ **do**

2: $\quad \overline{\mathbf{Y}}_{n\times m} \leftarrow \left[\frac{\sum_{k=1}^{r} \mathbf{Y}_l^{(k,j)}}{r}\right]_{n\times m}$, $j = \{1, \ldots, n+i-1\}$, $l = \{1, \ldots, m\}$

3: $\quad \overline{\mathbf{V}}_{n\times m} \leftarrow \left[\frac{\frac{1}{r-1}\sum_{k=1}^{r}\left[Y_l^{(k,j)} - \overline{y}_l^{(j)}\right]^2}{r}\right]_{n\times m}$, $j \in \{1, \ldots, n+i-1\}$, $l = \{1, \ldots, m\}$

4: $\quad sGP \leftarrow$ STOCHASTIC_GP_FIT($\mathbf{X}$, $\overline{\mathbf{Y}}_1$, $\overline{\mathbf{V}}_1$) ▷ Metamodel training for objective break strength

5: $\quad dGP1 \leftarrow$ ORDINARY_GP_FIT($\mathbf{X}$, $\overline{\mathbf{Y}}_2$) ▷ Metamodel training for objective production cost

6: $\quad dGP2 \leftarrow$ ORDINARY_GP_FIT($\mathbf{X}$, $\overline{\mathbf{C}}$) ▷ Metamodel training of the constraint

7: $\quad \mathbf{X}_{PF} \leftarrow \left\{\mathbf{x}^* \mid \mathbf{y}^* \; is \; feasible \; \wedge \; rank(\mathbf{y}^*) = 1, \; \mathbf{y}^* \in \overline{\mathbf{Y}}\right\}$

8: $\quad \widehat{Y}_{PF} \leftarrow [sGP(\mathbf{X}_{PF}) \;\; dGP1(\mathbf{X}_{PF})]_{n\times 2}$

9: $\quad \mathbf{x}^* \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}}$ cEHVI-SK($sGP, dGP1, \; dGP2, \; \widehat{Y}_{PF}, \; \mathbf{x}$) ▷ Acquisition function maximization

10: $\quad \mathbf{Y}^*_{2\times r}, \; C^* \leftarrow$ SIMULATE($\mathbf{x}^*$, r) ▷ Expensive evaluation of $\mathbf{x}^*$

11: $\quad \mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{x}^*$

12: $\quad \mathbf{Y} \leftarrow \mathbf{Y} \cup \mathbf{Y}^*$

13: $\quad \overline{\mathbf{C}} \leftarrow \overline{\mathbf{C}} \cup C^*$

14: **end for**

return the set of non-dominated feasible solutions

5.2.2 cMEI-SK acquisition function

When a scalarization function is used, then only one metamodel is trained on the scalarized objective at each BO iteration. In this work, we use the augmented Chebychev scalarization function (Equation 2.18), which is popular in general multi-objective optimization problems due to the theoretical guarantees it provides [107, 82].

We then fit a (stochastic) GPR metamodel on $Z_{\boldsymbol{\lambda}}(\mathbf{x})$, explicitly accounting for the intrinsic noise on the scalarized objective (see Equation 2.11). The metamodel information is then used in an acquisition function to guide the search for novel configurations (Line 10 in Algorithm 5.1). Here we use a modification of the EI presented in Chapter 2 and used in Chapter 3, referred to as MEI [126, 132], to compute the EI also accounting for the output heterogeneous noise (Line 10). For any arbitrary configuration $\mathbf{x}$, MEI is given by

$$\text{MEI-SK}(\mathbf{x}) = [\widehat{y}_s(\mathbf{x}_{\min}) - \widehat{y}_s(\mathbf{x})]\,\Phi\left(\frac{\widehat{y}_s(\mathbf{x}_{\min}) - \widehat{y}_s(\mathbf{x})}{\widehat{s}_o(\mathbf{x})}\right) + \widehat{s}_o\phi\left(\frac{\widehat{y}_s(\mathbf{x}_{\min}) - \widehat{y}_s(\mathbf{x})}{\widehat{s}_o(\mathbf{x})}\right) \tag{5.4}$$

where $\widehat{y}_s(\mathbf{x}_{\min})$ is the SK prediction for the scalarized function ($Z_{\boldsymbol{\lambda}}$) at $\mathbf{x}_{\min}$ (i.e., the point having the lowest sample mean for the scalarized objective among all feasible points already sampled), and $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function, respectively [126, 132]. Then, the corresponding constrained MEI (denoted cMEI-SK) is given by

$$\text{cMEI-SK}(\mathbf{x}) = \widehat{y}_c(\mathbf{x}) \times \text{MEI}(\mathbf{x}) \tag{5.5}$$

Note that at each BO iteration, a new weight vector $\boldsymbol{\lambda}$ is selected from a set of weights distributed uniformly, allowing the algorithm to sample points across the entire Pareto front [82].

5.2.3 cEHVI-SK acquisition function

When scalarization is not used, an independent (stochastic) GPR metamodel is trained for each of the m objectives. The Expected Hypervolume Improvement (EHVI) is a popular acquisition function in *unconstrained* and *deterministic* settings [100, 158]. As noticed in Chapter 2, the hypervolume is the size of the space dominated by a Pareto front $\mathbf{P}$ given a reference point [160]. Therefore, the hypervolume improvement of an objective vector $\mathbf{y} \in \mathbb{R}^m$ is defined as the increment of the hypervolume indicator after $\mathbf{y}$ is added to the current approximation of the Pareto front (see Equation 2.22).

For noisy problems, we adjusted the EHVI presented in Equation (5.6) to use the stochastic GPR prediction instead:

$$\text{EHVI-SK}(\mathbf{x}) = \int_{\mathbf{y}(\mathbf{x})\in A} I(\mathbf{y}(\mathbf{x}), \mathbf{P}) \prod_{i=1}^{m} \frac{1}{\widehat{s}_{s_i}(\mathbf{x})}\phi\left(\frac{y_i(\mathbf{x}) - \widehat{y}_{s_i}(\mathbf{x})}{\widehat{s}_{s_i}(\mathbf{x})}\right) dy_i(\mathbf{x}) \tag{5.6}$$

where A stands for the non-dominated area and $\phi(\cdot)$ is the standard normal density distribution function. The terms $\widehat{y}_{s_i}(\mathbf{x})$ and $\widehat{s}_{s_i}(\mathbf{x})$ represent the objective and uncertainty estimators of the stochastic GP model respectively. Finally, for a novel input configuration $\mathbf{x}$, the corresponding constrained EHVI (denoted cEHVI-SK) is given by (Line 9 of Algorithm 5.2)

$$\text{cEHVI-SK}(\mathbf{x}) = \widehat{y}_c(\mathbf{x}) \times \text{EHVI-SK}(\mathbf{x}) \tag{5.7}$$

5.3 Design of experiments

To test the proposed optimization approach, we benchmark its performance against five state-of-the-art constrained EMOAs: C-NSGA-II [22], C-MOEA/D [73], C-TAEA [92], C-MOPSO [54], an adaptation of C-NSGA-II to use the OK metamodel prediction to generate new populations (OK-C-NSGA-II) [124], and a modification to the surrogate-assisted evolutionary algorithm K-RVEA (referred as C-K-RVEA in this paper)[31]. The latter uses GP surrogates to approximate the objective functions, and it was modified to handle infeasible configurations with a penalization factor given by PoF (see Equation 5.1). To evaluate the impact of stochastic GPR models, we also implemented our proposed approaches using ordinary GPR models; these are denoted with an 'OK' in the algorithm name. Furthermore, the optimization of the acquisition function in Bayesian optimization tends to be non-trivial, as the function is often non-linear, non-convex, and multimodal [153]. Here we use a *Particle Swarm Optimization* (PSO) algorithm to find the infill point that maximizes the proposed acquisition functions (i.e., the fitness function of this inner optimization). Our choice is motivated by the good performance and low computational time observed in other studies with high-dimensional search space [156]. With PSO, the position of the particle represents the values of each variable to optimize.

Since the physical experiments are very expensive, a Matlab process simulator was provided by the Joining & Materials Lab[2]. This simulator predicts the lap shear strength of the sample (MPa), failure mode (adhesive, substrate, or cohesive failure), sample production cost (in euros), and visual quality outcome (OK or not OK) based on the process parameters discussed above, in a matter of seconds. However, this simulator is *not* meant to be a digital twin of the true process, but rather a tool for the *relative* comparison of the performance of the five algorithms, under different conditions, at almost zero cost. Table 5.1 shows the range of each process parameter considered in the optimization problem, and Table 5.2 summarizes the parameters of the optimization algorithms.

Table 5.1: Range of the process settings (input variables) considered in the optimization.

ID	Variable	Min	Max
v1	Pre-processing	Yes or No	
v2	Power setting (W)	300	500
v3	Torch speed (mm/s)	5	250
v4	Distance between the torch and the sample (cm)	0.2	2
v5	Number of passes	1	50
v6	Time between plasma treatment and glue application (min)	1	120

The simulator allows the analyst to experiment with different levels of noise. In real life, multiple factors cause noise to occur. One of these is the so-called *contact angle*[3]: this

[2] `https://www.flandersmake.be`

[3] Other noise factors could not be controlled in the simulator, so they are not further discussed.

typical measure reflects the extent to which the adhesive can maintain good contact with the material. This is important to achieve a strong adhesive bond. The contact angle depends on the type of material but also on impurities or contaminants such as wax, oil, plasticizers, etc. present on the material surface. Even though all samples in our setting are made of the same material, variations in the degree of these contaminants occur across samples, implying variations in contact angle. These result in noisy measurements of the final break strength of the bonded joints. A realistic value for the *standard deviation of the contact angle* is $\gamma = 30\%$ of the mean, which is what we use in our experiments.

Table 5.2: Summary of the parameters of the optimization algorithms

Setting	**BO algorithms**	**Evolutionary algorithms**
Size of initial design/population	LHS: $N = 20$	
Crossover probability	-	0.5 (C-MOEA/D); 0.9 o.w
Mutation probability	-	0.1
Reference directions	-	19 (C-TAEA); 3 (C-MOEA/D)
Inertia weight	-	0.4 (C-MOPSO)
C1 factor	-	2 (C-MOPSO)
C2 factor	-	2 (C-MOPSO)
Max velocity (%)	-	5 (C-MOPSO)
Reference vectors	-	151 (C-K-RVEA)
Replications	$r = 5$	
Iterations	40	2
Acquisition function	cMEI-SK/cEHVI-SK	-
Acquisition function optimization	PSO*	-
Kernel (for all GPR models)	Gaussian	-

* PSO configuration: swarm size = 50, max iterations=1800, max stall iterations = 10, tolerance = $1e^{-6}$

Given the experimental setting in Table 5.2, each algorithm evaluates exactly 60 process configurations in an expensive way, with 5 replications per configuration (i.e., 300 expensive evaluations in total). The BO algorithms start with an initial design of 20 process configurations (note that this is smaller than the usual choice of $k = 11d - 1$); exactly one infill point is added in the following 40 iterations. The EMOAs, by contrast, use an initial population of 20 process configurations (the same initial set used by BO algorithms) and in each of the 2 successive generations, a novel population is generated. As common in the literature, the fitness of the configuration outcomes is evaluated based on their sample means over the 5 replications (note that, by doing so, these algorithms implicitly ignore the fact that this sample mean is in itself uncertain). Our approach takes into account both the sample mean and the sample variance though, as explained in Section 5.2. While a total budget of 300 evaluations may seem high, it allows us to also study the progress the algorithms would have obtained at lower budgets, as illustrated below.

Furthermore, Zhan and Xing [158] present a different formulation for constrained EI by using the definition of EI and the Probability of Feasibility (PoF) [141] to model the

constraints. However, we observed in our experiments that the formulation given in Equation (5.5) got, on average, superior results compared to the results obtained with the formulation suggested in [158]. Appendix C.1 details these results.

We evaluate the quality of the resulting fronts using the hypervolume (HV) and IGD+ indicators [7, 94, 71], applied to the sample means. We also include performance analysis of the proposed methods using the *empirical attainment function* [102]. As the front obtained by the algorithms may depend on the initial design, we performed 50 macro-replications, each one starting with a different initial design.

5.4 Results

To gain some insight into the objective space to be explored by the algorithms under idealized conditions (i.e., if the contact angle could be perfectly controlled), Figure 5.3 shows the mean responses of the simulator on a set of 60 000 process configurations (with $\gamma = 0\%$). These configurations were determined through Halton sampling, and each configuration was replicated five times. Interestingly, the feasible solutions seem to be clustered in areas with high break strength. Moreover, the use of pre-processing seems to merely lead to a cost increase, while the resulting gains in break strength are scarce and only minor. The Pareto optimal (feasible) points estimated by means of these Halton results, under these idealized conditions ($\gamma = 0\%$), serve as the benchmark Pareto front to judge the quality of the competing optimization algorithms.

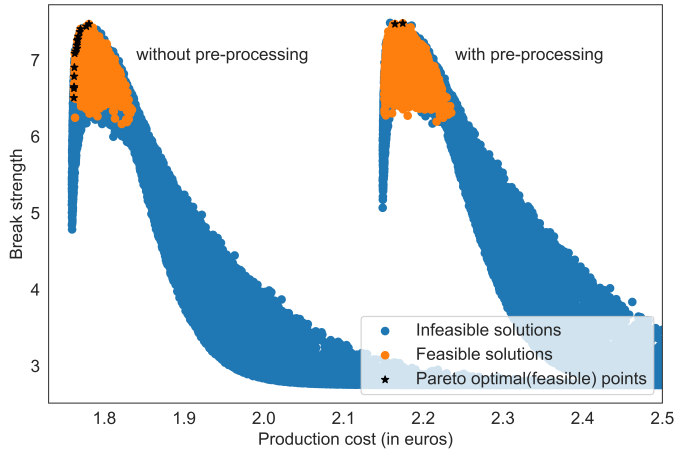


Figure 5.3: Sample mean of break strength versus production cost, estimated by the simulator for 60 000 random process configurations ($\gamma = 0\%$).

Figure 5.4 shows an exploratory analysis of the Pareto fronts obtained by each BO algorithm in the different macro-replications. The analysis is performed using the concept of the empirical attainment function (EAF, [102]). For each point, the EAF gives an estimate of the probability that this point is dominated (attained) by the Pareto front put forward by the given algorithm. Connecting points with the same given EAF value yields an *attainment surface* that separates the objective space into two regions: those objective vectors that are

attained by the resulting Pareto fronts with (at least) that probability, and those that are not. The attainment surfaces allow us to summarize the location of the objective vectors obtained by a stochastic algorithm. The median attainment surface, for instance, consists of the objective vectors that are attained by half of the runs (representing a probability of 50%). Similarly, the worst-case results of an algorithm are reflected in the worst attainment surface, whereas the best results are given by the best attainment surface. The shaded areas in the right-hand side of Figure 5.4 show the objective areas where the proposed cMEI-SK approach reaches better attainment surfaces minimizing the objective **cost**. On the other hand, using cEHVI-SK means a better attainment surface minimizing the objective **break strength**.

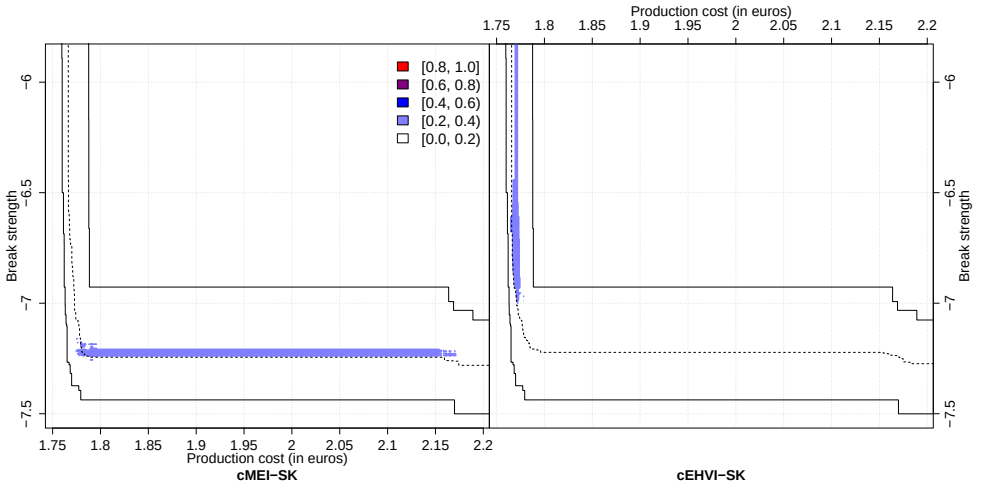
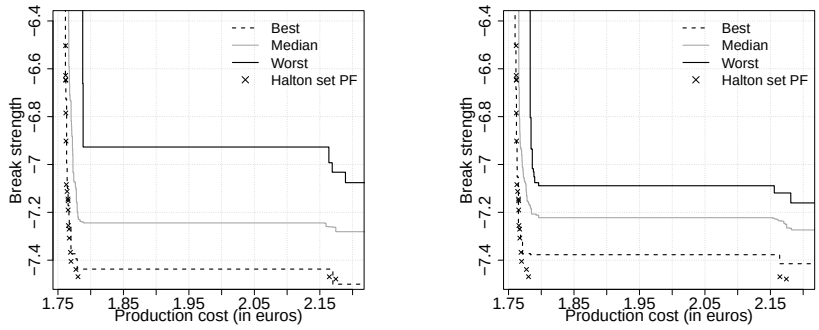


Figure 5.4: Visualization of the differences between the EAFs using different acquisition functions in the BO methods. The Y-axis is inverted to allow for the minimization of both objectives. Each plot shows the median surface (dashed line), along with the best and worst surfaces (full lines). The areas where a better attainment surface is obtained are indicated by a shaded area (the colors indicate the improvement in the probabilities).

Figure 5.5 shows the final *best* Pareto front obtained by the proposed approaches over 50 macro-replications (along with the *median* and *worst* fronts), for $\gamma = 30\%$. The best Pareto front obtained by cEHVI-SK is very close to the ideal Pareto front estimated by means of the Halton set exploration. It also leads to a *faster* increase in HV, in terms of the number of expensive evaluations performed, than all other algorithms. This is evident from Figure 5.6, which shows the evolution of the *average* hypervolume and IGD+ obtained (across macro-replications) during the optimization process (again, for $\gamma = 30\%$). This superior performance is also evident from Table 5.3, which shows the results for the average HV, along with those of the average IGD+ (the latter uses the ideal front obtained in the Halton experiment of Figure 5.3 as the true front). These results highlight that BO methods are able to obtain better quality results for the Pareto front than evolutionary algorithms (including surrogate-assisted ones), particularly at very limited budgets. Appendix C.2 shows that statistical differences (Wilcoxon test, $\alpha = 5\%$), both for HV and IGD+, were mainly

focused amongst C-MOPSO, C-MOEA/D, and C-TAEA; and metamodel-based optimization algorithms (including C-K-RVEA).



(a) $cEHVI - SK$ **(b)** $cMEI - SK$

Figure 5.5: Best, median, and worst Pareto front obtained by the BO methods. The Y-axis is inverted to allow minimization of both objectives

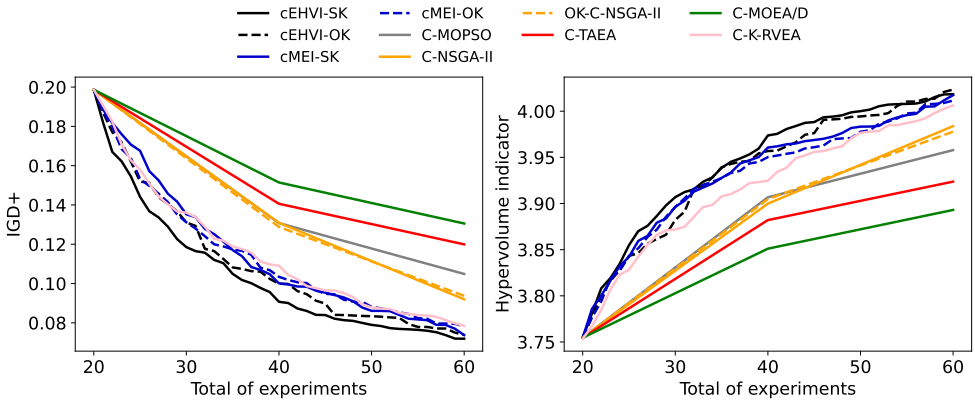


Figure 5.6: Evolution of the IGD+ metric (Left) and hypervolume indicator (Right) of the Pareto-optimal solutions throughout the search, for $\gamma = 30\%$ (average of 50 macro-replications). The Pareto front obtained from the Halton set is considered to compute the IGD+ metric and the reference point with production cost=3, break strength=4 is used to compute the hypervolume indicator

Overall, ignoring the input-dependent noise associated with the objective *break strength* (thus using the mean value of the replications) negatively influences the performance of optimization algorithms. As shown in Figure 5.6, the curves of the *deterministic* BO methods (i.e., the proposed algorithms using OK metamodels) are inferior to the one when input dependency is taken into account. Yet, they remain superior to the evolutionary algorithms, which reinforces the advantages of the exploration/exploitation performed by the optimization of the acquisition function and thus obtaining the best possible trade-off between expected performance and model error. Additionally, we note that surrogate-assisted

evolutionary algorithms (such as C-K-RVEA) can indeed benefit from the approximation of the objectives and obtain better configurations than standard EMOAs. However, this performance remains inferior to BO approaches when data efficiency is needed on top of effective black-box optimization.

Finally, by looking closer at the solutions (in input space) suggested by cEHVI-SK (the one with the highest HV value), we found that ±63% of the solutions skip pre-processing, meaning that the production costs are reduced. Thus in Figure 5.7 we show the distribution of the remaining input variables. Overall, the algorithm suggests that the power should be between 480 W and 500 W, the speed should move at a speed between 127.5 mm/s and 152 mm/s, the distance between the torch and the sample should be in the range of 0.2cm and 0.38 cm, between 11 and 16 passes should be performed, and a time difference between 1 and 13 minutes should be considered before the glue application.

Table 5.3: Average IGD+ and HV of the fronts obtained over 50 macro-replications, for $\gamma = 30\%$

	IGD+	**HV**
cEHVI-SK	0.0719	4.0184
cEHVI-OK	0.0739	4.0235
cMEI-SK	0.0737	4.0174
cMEI-OK	0.0782	4.0116
C-K-RVEA	0.0785	4.0062
C-NSGA-II	0.092	3.9839
OK-C-NSGA-II	0.0937	3.979
C-MOPSO	0.1049	3.9579
C-TAEA	0.1199	3.9237
C-MOEA/D	0.1306	3.893

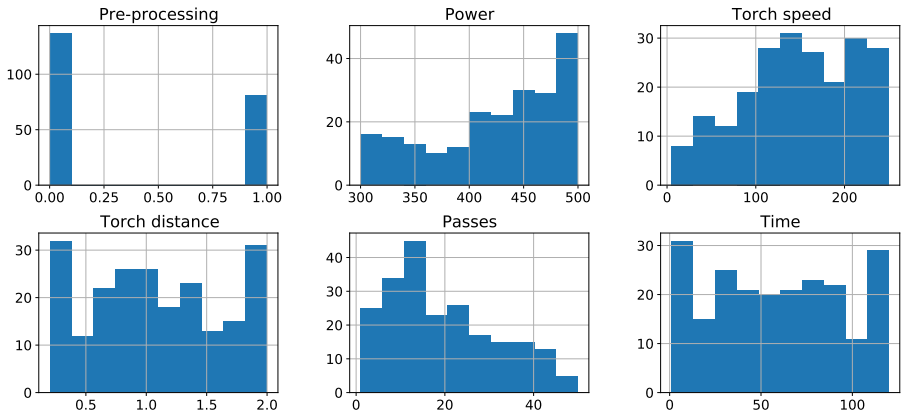


Figure 5.7: Distribution of the Pareto-optimal input values obtained by MO-GP_cEI (EHVI), across 50 macro-replications.

5.5 Concluding note

This chapter presented two constrained Bayesian optimization algorithms to solve a bi-objective problem related to the adhesive bonding process of materials (maximizing break strength while minimizing production costs). As the real experiments carried out physically in a lab are costly, the budget for evaluations is very limited. The proposed Bayesian approach is shown to clearly outperform state-of-the-art evolutionary algorithms, which are commonly used in engineering design when solving general multi-objective, constrained problems. The difference lies in the way the experimental design is guided throughout the search: the Bayesian approach selects infill points based on an (explainable) acquisition function, which is related to the expected merit of the new infill point for optimization. The BO model ensures that the search focuses on infill points that have a high probability of being feasible. Moreover, the GP model used to approximate the objective(s) accounts for the output (heterogenous) noise, whereas the evolutionary algorithms rely simply on the (uncertain) sample means as performance approximations. Moreover, the search in EMOAs (as it is generally with metaheuristic approaches) is guided by hard-to-tune evolutionary operators. The success of evolutionary processes is largely dependent on the availability of a sufficient experimentation budget, which is not always the case in practice.

Our research highlights the superiority of BO methods, particularly using an EHVI-based infill criterion, over traditional EMOAs. We are convinced that the use of Bayesian approaches holds great promise in solving noisy and expensive engineering problems, in terms of both search efficiency (i.e., finding solutions within a limited budget) and search effectiveness (i.e., yielding high-quality solutions).

— CHAPTER 6 —

WINTER IS COMING

Online learning of windmill time series using Long Short-term Cognitive Networks

Forecasting windmill time series is often the basis of other processes such as anomaly detection, health monitoring, or maintenance scheduling. In our opinion, the amount of data generated by windmill farms make online learning the most viable strategy to follow. Such settings require retraining the model each time new data is available. However, updating the model with new information is often very expensive when using traditional Recurrent Neural Networks (RNNs). This chapter presents the application of Long Short-term Cognitive Networks (LSTCNs) to forecast windmill time series in online settings.[1] Section 6.2 analyzes the main components of the proposed LSTCN-based power forecasting model. LSTCN consists of chained Short-term Cognitive Network blocks, each processing a temporal data chunk. The learning algorithm of these blocks is based on a fast, deterministic learning rule that makes LSTCNs suitable for online learning tasks. The numerical simulations presented in Section 6.3 using a case study involving four windmills showed that our approach reported the lowest forecasting errors with respect to a simple RNN, a Long Short-term Memory, a Gated Recurrent Unit, and a Hidden Markov Model. What is perhaps more important is that the LSTCN approach is significantly faster than these state-of-the-art models.

6.1 Forecasting models with recurrent neural networks

Neural networks are a family of biology-inspired computational models that have found applications in many fields. An example of engineering applications of neural models is the support of wind turbine operation and maintenance. In this area, neural models dedicated to the analysis of temporal data have proven to be quite useful. This is motivated by the fact that typical data describing the operation of a wind turbine are collected by sensors forming a supervisory control and data acquisition (SCADA) system [38, 151]. Such data come in the form of long sequences of numerical values [144], thus making Recurrent Neural Networks (RNNs) the right choice for processing such data. However, traditional neural networks (including RNN) face additional challenges in time series forecasting; such as expert knowledge requirements and handcrafted features, no full modeling of long-term dependencies hidden in time-domain signals, lack of interpretability, long training times, and/or gradient vanishing

[1] The content of this chapter has been included in the publication *"Online learning of windmill time series using Long Short-term Cognitive Networks". In: Expert Systems with Applications* [113].

problems. This section briefly revises the literature on the applications of RNNs on data analysis in the area of wind turbine operation and maintenance support.

RNNs differ from other neural networks in the way the input data is propagated. In standard neural networks, the input data is processed in a feed-forward manner, meaning the signal is transmitted unidirectionally. In RNN models, the signal goes through neurons that can have backward connections from further layers to earlier layers [27]. Depending on a particular neural model architecture, we can restrict the layers with feedback connections to only selected ones. The overall idea is to allow the network to "revisit" nodes, which mimics the natural phenomenon of memory [86]. RNNs turned out to be useful for accurate time series prediction tasks [142], including wind turbine time series prediction [34].

Currently, the most popular variant of RNN in the field of wind turbine data processing is the Long Short-Term Memory (LSTM) model [64, 109]. In this model, the inner operations are defined by neural gates called *cell*, *input gate*, *output gate*, and *forget gate*. The cell acts as the memory, while the other components determine the way the signal propagates through the neural architecture [159]. The introduction of these specialized units helped prevent (to some extent) the gradient problems associated with training RNN models [138].

Existing neural network approaches to wind turbine data forecasting do not pay enough attention to the issue of model complexity and efficiency. In most studies, authors reduce the available set of input variables rather than optimizing the neural architecture used. For example, Bo et. al. [49] used the LSTM model with hand-picked three SCADA input variables, while Riganti-Fulginei, Sun, and Sun [131] used eleven SCADA variables. Qian et. al. [123] also used LSTM to predict wind turbine data. In their study, the initial set of input variables consisted of 121 series, but this was later reduced to only three variables and then to two variables using the Mahalanobis distance method. The issue of pre-processing and feature selection was also raised by Wang et. al. [150], suggesting Principal Component Analysis to reduce the dimensionality of the data.

LSTM has been found to work well even when the time series variables are of incompatible types; by applying data-fusion strategies and capturing long-term dependencies through the recursive behavior and gate mechanism of this network. It is worth citing the study of Lei, Liu, and Jiang [91], who used LSTM to predict two qualitatively different types of time series simultaneously: (i) vibration measurements that have a high sampling rate and (ii) slow varying measurements (e.g., bearing temperature). It should be noted that existing studies bring additional techniques that enhance the capabilities of the standard LSTM model. For example, Cao et. al. [25] propose segmenting the data and using segment-related features instead of raw signals. Ling et. al. [154] also do not use raw signals. Instead, they use Convolutional Neural Networks (CNNs) to extract the dynamic features of the data, which is then fed to LSTM. A similar approach, combining CNN with LSTM, was presented in [155]. Another interesting technique was introduced by Chen et. al. [28], who combined LSTM with an auto-encoder (AE) neural network so that their model can detect and reject anomalies while achieving better results for non-anomalous data.

While most of the recently published studies employ LSTM to predict multivariate wind turbine time series, there are also several approaches focusing on other RNN variants. For example, there are several papers on the use of Elman neural networks in forecasting multivariate wind turbine data [97, 98, 88]. Likewise, we should mention the work of López et. al. [101], which involved Echo State Network and LSTM. Finally, it is worth mentioning the work of Kong et. al. [87], in which the task of processing data from wind turbines is imple-

mented using CNNs and Gated Recurrent Unit (GRU) [29]. The latter neural architecture is a variant of RNN, which can be seen as a simplification of the LSTM architecture.

There are other models equipped with reasoning mechanisms similar to the one used by neural networks. In particular, the concept of "neuron" can also be found in Hidden Markov Models (HMMs) [127]. Such neurons are implemented as *states*, and the set of states essentially plays a role analogous to that of hidden neurons in a standard neural network. HMMs have also found applications in wind power forecasting. The studies of Bhaumik et. al. [17] and Qu et. al. [125] should be mentioned in this context. Both research teams highlight decent predictions and robustness to noise in the data.

Recently, Nápoles et. al. [117] introduced a recurrent neural system termed *Long Short-term Cognitive Network* (LSTCN) that seems suitable for online learning settings where data might be volatile. Moreover, the cognitive component of such a recurrent neural network allows for interpretability and it is given by two facts. Firstly, neural concepts and weights have a well-defined meaning for the problem domain being modeled. This means that the resulting model can easily be interpreted with little effort. For example, in [117] the authors discussed a measure to compute the relevance of each variable in multivariate time series without the need for any post-hoc method. Additionally, the Short-term Cognitive Network (STCN) composing an LSTCN enables inserting domain knowledge into the network by modifying a prior knowledge matrix, which is not altered during the learning process [118].

Despite the advantages of the LSTCN model when it comes to its forecasting capabilities, intrinsic interpretability, and short training time, it has not yet been applied to a real-world problem, as far as we know. In addition, we have little knowledge of the performance of this brand-new model in online learning settings operating with volatile data that might be available for a short time. Such a lack of knowledge and the challenges related to the wind prediction described above have motivated us to study the LSTCNs' performance on a real-world problem concerning the power forecasting of four windmills.

6.2 Long Short-term Cognitive Network

This section elaborates on the task of forecasting power generation in windmills using the LSTCN model. By doing that, we propose an LSTCN-based pipeline to tackle the related online learning problem where each data chunk is processed only once. In this pipeline, every time a new data chunk is available, a pre-processing step is applied to transform it into the structure needed to train the network. Then, a new Short-term Cognitive Network (STCN) block [118] will use the knowledge transferred from the previous block to learn from the incoming data. This means the model can be retrained without compromising what the network learned from previous data chunks. Given the transfer of knowledge implemented from previous blocks to the new ones, future inference steps are performed using only the last STCN block. The LSTCN-based pipeline is detailed below. First, we describe the data transformation needed to apply this model and the architecture and neural reasoning of LSTCN. Later, we discuss how the parameter learning is performed to forecast multivariate time series.

6.2.1 Data preparation for online learning simulations

Let $x \in \mathbb{R}$ be a variable observed over a discrete time scale within a period $t \in \{1, 2, \ldots, T\}$ where $T \in \mathbb{N}$ is the number of observations. Hence, a univariate time series can be defined as a sequence of observations $\{x^{(t)}\}_{t=1}^{T} = \{x^{(1)}, x^{(2)}, \ldots, x^{(T)}\}$. Similarly, we can define a multivariate time series as a sequence $\{\mathbf{X}^{(t)}\}_{t=1}^{T} = \{\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(T)}\}$ of vectors of d variables, such that $\mathbf{X}^{(t)} = [x_1^{(t)}, x_2^{(t)}, \ldots, x_d^{(t)}]$. A model F is used to forecast the next $L < T$ steps ahead. In this paper, we assume that the model F is built as a sequence of neural blocks with local learning capabilities, each able to capture the trends in the current time patch (i.e., a chunk of the time series) being processed. The following sub-sections will detail the network architecture and the parameter learning algorithm.

Let us assume that $\mathbf{X} \in \mathbb{R}^{d \times T}$ is a dataset comprising a multivariate time series (Figure 6.1a). Firstly, we need to transform $\mathbf{X}$ into a set of Q tuples with the form $(\mathbf{X}^{(t-R)}, \mathbf{X}^{(t+L)}), t - R > 0, t + L \leq T$ where R represents how many past steps we will use to forecast the following L steps ahead (see Figure 6.1b). This step is common in time series forecasting algorithms since prepares the data so that the network can learn from the past to forecast the future. We assume that $R = L$ for the sake of simplicity. Secondly, each component in the tuple is flattened such that we obtain a $Q \times (d(R + L))$ matrix. Although this is an accommodation implemented to facilitate the learning process, this does not imply a loss of information, and similar accommodations are used together with LSTM networks as well. Finally, we create the set $P = \{P^{(1)}, \ldots, P^{(k)}, \ldots, P^{(K)}\}$ from the set of flattened tuples such that $P^{(k)} = (P_1^{(k)}, P_2^{(k)})$ is the k-th time patch involving two data pieces $P_1^{(k)}, P_2^{(k)} \in \mathbb{R}^{C \times N}$, where $N = dR$ and C denotes the number of instances in that time patch.

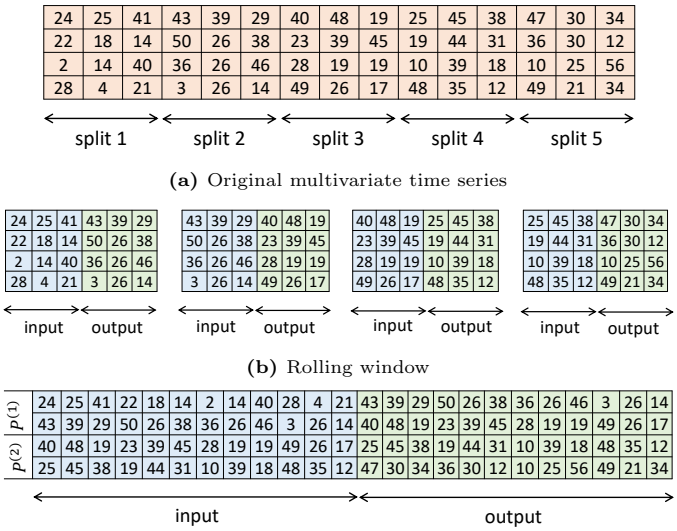
Figure 6.1 shows an example of such a pre-processing method. First, the times series is split into chunks of equal length as defined by the L and R parameters. Second, we use the resulting chunks to create a set of input-output pairs. Finally, we flatten these pairs to obtain the tuples with the inputs to the network and the corresponding expected outputs.

It should be highlighted that the forecasting model will have access to a time patch in each iteration, as it usually happens in an online scenario. If the neural model is fed with several time steps, then it will be able to forecast multiple-step ahead of all variables describing the time series.

6.2.2 Network architecture and neural reasoning

In the online learning setting, we consider a time series (regardless of the number of observed variables) as a sequence of time patches of a certain length. Such a sequence refers to the set $P = \{P^{(1)}, \ldots, P^{(k)}, \ldots, P^{(K)}\}$ obtained with the data preparation steps discussed in the previous subsection. Hence, the proposed network architecture consists of an LSTCN model able to process the sequence of time patches.

An LSTCN model can be defined as a collection of STCN blocks, each processing a specific time patch and transferring knowledge to the following STCN block in the form of weight matrices. Figure 6.2 shows the recurrent pipeline of an LSTCN involving three STCN blocks to model a multivariate time series decomposed into three time patches. It should be highlighted that learning happens inside each STCN block to prevent the information flow from vanishing as the network processes more time patches. Moreover, weights estimated in



(a) Original multivariate time series

(b) Rolling window

(c) Flattening

Figure 6.1: Data pre-processing using $R = L = 3$. (a) The original multivariate time series $X \in \mathbb{R}^{d \times T}$, with rows as variables and columns as timestamps. (b) Selection of sub-sequences of the time series according to parameters R and L. (c) Each sub-sequence is flattened to obtain the temporal instances. In this example, the flattened dataset is divided into two temporal parches.

the current STCN block are transferred to the following STCN block to perform the next reasoning process (see Figure 6.3). These weights will no longer be modified in subsequent learning processes, which allows for preserving the knowledge we have learned up to the current time patch. In our opinion, that makes our approach suitable for online learning settings.

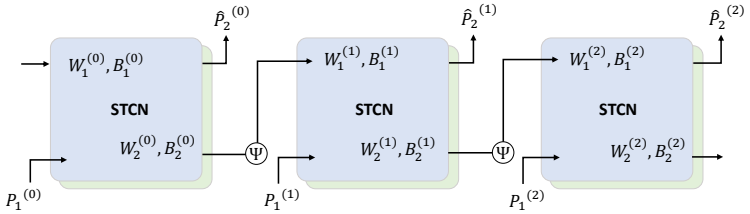


Figure 6.2: LSTCN architecture of three STCN blocks. The weights learned in the current block (last one) are transferred to the following STCN block as a prior knowledge matrix.

The reasoning within an STCN block involves two gates: the *input gate* and the *output gate*. The input gate operates the prior knowledge matrix $W_1^{(k)} \in \mathbb{R}^{N \times N}$ with the input data $P_1^{(k)} \in \mathbb{R}^{C \times N}$ and the prior bias matrix $B_1^{(k)} \in \mathbb{R}^{1 \times N}$ denoting the bias weights. Both matrices $W_1^{(k)}$ and $B_1^{(k)}$ are transferred from the previous block and remain locked during the learning phase to be performed in that STCN block. Note that matrices $W_1^{(k)}$ and $B_1^{(k)}$ resemble somehow the existing memory cell in LSTM networks to "remember" what has happened up to the time step t, in our case, time patch k. The result of the input gate

is a temporal state $H^{(k)} \in \mathbb{R}^{C \times N}$ that represents the outcome that the block would have produced given $P_1^{(k)}$ if the block would not have been adjusted to the block's expected output $P_2^{(k)}$. Such an adaptation is done in the output gate where the temporal state is operated with the matrices $W_2^{(k)} \in \mathbb{R}^{N \times N}$ and $B_2^{(k)} \in \mathbb{R}^{1 \times N}$, which contain learnable weights. Figure 6.3 depicts the reasoning process within the k-th block.

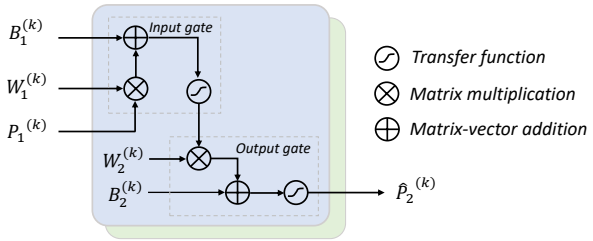


Figure 6.3: Reasoning within an STCN block. Firstly, the current time patch is mixed with the prior knowledge matrices $W_1^{(k)}$ and $B_1^{(k)}$. This operation produces a temporal state matrix $H^{(k)}$. Secondly, we operate the $H^{(k)}$ matrix with the matrices $W_2^{(k)}$ and $B_2^{(k)}$. The result of such an operation will be an approximation of the expected output $P_2^{(k)}$

Equations (6.1) and (6.2) show the short-term reasoning process of this model in the k-th iteration,

$$\hat{P_2}^{(k)} = f\left(H^{(k)} W_2^{(k)} \oplus B_2^{(k)}\right) \tag{6.1}$$

and

$$H^{(k)} = f\left(P_1^{(k)} W_1^{(k)} \oplus B_1^{(k)}\right) \tag{6.2}$$

where $f(x) = \frac{1}{1+e^{-x}}$, whereas $\hat{P_2}^{(k)}$ is an approximation of the expected block's output. In these equations, the $\oplus$ operator performs a matrix-vector addition by operating each row of a given matrix with a vector, provided that both the matrix and the vector have the same number of columns. Notice that we assumed that values to be forecast are in the $[0, 1]$ interval.

As mentioned, the LSTCN model consists of a sequential collection of STCN blocks. In this neural system, the knowledge from one block is passed to the next one using an aggregation procedure (see Figure 6.2). This aggregation operates on the knowledge learned in the previous block (that is to say, the $W_2^{(k-1)}$ matrix). In this paper, we use the following non-linear operator in all our simulations:

$$W_1^{(k)} = \Psi(W_2^{(k-1)}),\ k - 1 \geq 0 \tag{6.3}$$

and

$$B_1^{(k)} = \Psi(B_2^{(k-1)}),\ k - 1 \geq 0 \tag{6.4}$$

such that $\Psi(x) = tanh(x)$. However, we can design operators combining the knowledge in both $W_1^{(k-1)}$ and $W_2^{(k-1)}$.

There is an important detail to be discussed. Once we have processed the available sequence (i.e., performed K short-term reasoning steps with their corresponding learning

processes), the whole LSTCN model will narrow down to the last STCN block. Therefore, that block will be used to forecast new data chunks as they arrive and a new learning process will follow, as needed in online learning settings.

6.2.3 Parameter learning

Training the LSTCN in Figure 6.2 means training each STCN block with its corresponding time patch. The learning process within a block is partially independent of other blocks as it only uses the prior weights matrices that are transferred from the previous block. As mentioned, these prior knowledge matrices are used to compute the temporal state and are not modified during the block's learning process.

The learning task within an STCN block can be summarized as follows. Given a temporal state $H^{(k)}$ resulting from the input gate and the block's expected output $P_2^{(k)}$, we need to compute the matrices $W_2^{(k)} \in \mathbb{R}^{N \times N}$ and $B_2^{(k)} \in \mathbb{R}^{1 \times N}$.

Mathematically speaking, the learning is performed by solving a system of linear equations that adapt the temporal state to the expected output. Equation (6.5) displays the deterministic learning rule solving this regression problem,

$$\begin{bmatrix} W_2^{(k)} \\ B_2^{(k)} \end{bmatrix} = \left[\left(\Phi^{(k)} \right)^\top \Phi^{(k)} + \lambda \Omega^{(k)} \right]^{-1} \left(\Phi^{(k)} \right)^\top f^- \left(P_2^{(k)} \right) \tag{6.5}$$

where $\Phi^{(k)} = (H^{(k)}|A)$ such that $A_{C \times 1}$ is a column vector filled with ones, $\Omega^{(k)}$ denotes the diagonal matrix of $(\Phi^{(k)})^\top \Phi^{(k)}$, while $\lambda \geq 0$ denotes the ridge regularization penalty. This learning rule assumes that the neuron's activation values inner layer are standardized. When the final weights are returned, they are adjusted back into their original scale.

It shall be noted that we need to specify $W_1^{(0)}$ and $B_1^{(0)}$ in the first STCN block. We can use a transfer learning approach from a previous learning process or it can be provided by domain experts. Since this information is not available, we fit a single STCN block without an intermediate state (i.e., $H^{(0)} = P_1^{(0)}$) on a smoothed representation of the whole (available) time series. The smoothed time series is obtained using the moving average method for a given window size w. This resembles somehow Exponential Smoothing-based forecasting methods [55] but without having to specify any additional parameter to control the influence of the observations at prior time steps in the forecasting. Instead, the learning rule of the STCN block can learn such influence. As referred by the Nápoles *et. al.*, the training performed on each STCN block is similar to the one performed in Extreme Learning Machine (ELM) [67], which is a special case of a two-layer multilayer perceptron. However, LSTCN uses prior knowledge to initialize matrices $W_1^{(0)}$ and $B_1^{(0)}$ instead of the random initialization used in ELM. Additionally, the hidden layer used in ELM can be of an arbitrary width while an STCN block uses the number of steps ahead to be predicted and the number of features in the multivariate time series.

Figure 6.4 portrays the workflow of the iterative learning process of an LSTCN model. An incoming chunk of data triggers a new training process using the knowledge learned in previous iterations and stored on the last STCN block. After the data pre-processing explained in Figure 6.1 is applied to the new data and a new STCN block is trained, the prior knowledge matrices are updated using an aggregation operator and stored to perform reasoning.

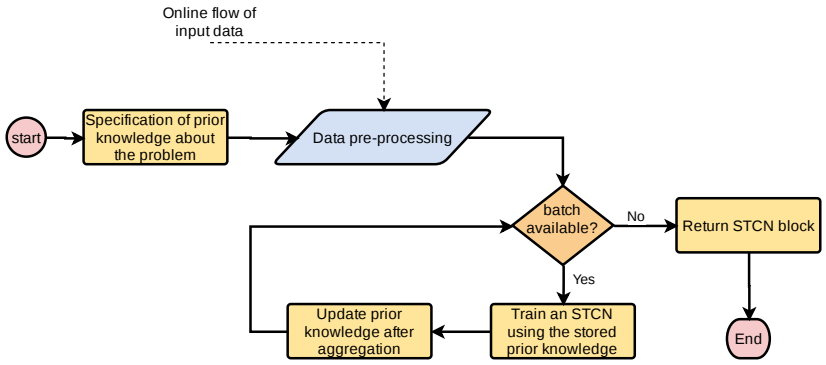


Figure 6.4: The LSTCN model can be seen as a sequential collection of STCN blocks that perform iterative learning. When a new chunk of data is available, a new STCN block is trained and the prior knowledge is updated using an aggregation procedure.

6.3 Numerical simulations

This section explores the performance (forecasting error and training time) of the proposed LSTCN-based online forecasting model for windmill time series.

6.3.1 Description of windmill datasets

To conduct our experiments, we adopted four public datasets from the ENGIE web page[2]. Each dataset corresponds to a windmill where measurements were recorded every 10 minutes from 2013 to 2017. The time series of each windmill contains 264,671 timestamps. Eight variables concerning the windmill and environmental conditions were selected: *generated power, rotor temperature, rotor bearing temperature, gearbox inlet temperature, generator stator temperature, wind speed, outdoor temperature,* and *nacelle temperature.*

As for the pre-processing steps, we removed duplicated timestamps, imputed missing timestamps and values, and applied a min-max normalization. Moreover, the data preparation procedure described in Figure 6.1 was applied to each dataset. Table 6.1 displays a descriptive summary of all datasets after normalization where the minimum, median, and maximum of the absolute Pearson's correlation values among the variables are denoted as *min*, *med*, and *max*, respectively.

Table 6.1: Descriptive statistics for the windmill datasets

Dataset	*min*	*med*	*max*
1	0.0708	0.2799	0.9456
2	0.0888	0.3032	0.8848
3	0.0687	0.3014	0.9497
4	0.0835	0.3148	0.9441

We split each dataset using a hold-out approach (80% for training and 20% for testing purposes). As for the performance metric, we use the mean absolute error (MAE) in all

[2] `https://opendata-renewables.engie.com/explore/index`

simulations reported in this section. In addition, we report the training and test times of each forecasting model. The training time (in seconds) of each algorithm was computed by adding the time needed to train the algorithm in each time patch. Finally, we arbitrarily fix the size of time patches to 1024 (batch size for other RNN).

6.3.2 Baseline models

We contrast the LSTCNs' performance against five different models used to handle online learning settings. Four recurrent models were considered in the comparison following a network-based approach where the output is to be fed back to the input. The other baseline model was Vector Exponential Smoothing (VES) [139], which builds directly upon the Exponential Smoothing for univariate time series but considers now the relation amongst each variable.

The RNN, LSTM, and GRU networks were implemented using Keras v2.4.3, while HMM was implemented using the *hmmlearn* library[3]. The training of these models was adapted to online learning scenarios. In practice, this means that RNN, GRU, and LSTM were retrained on each time patch using the prior knowledge structures learned in previous learning steps. The transition probability matrix is passed from one patch to another and updated based on the new information in the HMM-based model. As for the fifth forecasting model, the implementation provided for VES on the package *Legion*[4] was used. In this case, the model does not use any information from previous batches and is trained only with the (temporally) available data.

In the LSTCN model, we used $L = \{6, 48, 72\}$ such that $R = L$ (hereinafter we will only refer to L) and $w = 10$. Notice that given the sampling interval of the data, six steps represent one hour while 72 steps represent half a day. We did not perform hyperparameter tuning since the online learning setting demands fast re-training of these recurrent models when a new data chunk arrives. It would not be feasible to fine-tune the hyperparameters in each iteration since such a process is computationally demanding. Instead, we retained the default hyperparameters reported on the corresponding Keras layers. We used four hidden states and Gaussian emissions to generate the predictions in the HMM-based model. These parameter values were arbitrarily selected without further experimentation.

Regardless of our choice, we recognize that HPO is as essential in online learning as in traditional ML problems. However, there is very little consensus when it comes to how to perform hyperparameter tuning in the online setting [10]. As an alternative to keeping the default hyperparameter values, one can (1) perform a sensitivity analysis on some datasets [90] and pick the hyperparameters that work well to use in the online setting, (2) periodically perform an offline HPO with logged data [2] and adjust the online learner with the new hyperparameter configuration if the performance improved with the new HP configuration, (3) perform an initial training of a candidate set of hyperparameter configurations and select the model with lower errors to make predictions [20], (4) transfer information from previous HPO steps to adjust the hyperparameters of the model when new data is available [157], or (5) collect meta-data that describe prior learning tasks and previously learned models (hyperparameter settings, pipeline compositions and/or network architectures, the resulting

[3] `https://github.com/hmmlearn/hmmlearn`
[4] `https://cran.r-project.org/package=legion`

model evaluations, the learned model parameters, as well as measurable properties of the task itself) [70].

The last two approaches have the advantage of somehow considering past optimization steps to guide the model optimization with new data. For instance, the optimization algorithm can start the optimization with previous optimal hyperparameter configurations as *warm-up* or use the last metamodel (in BO-based optimization) to estimate the performance of the ML algorithm with new data. Since this metamodel was built only on the previous dataset, the online optimization algorithm will progressively adjust the response surface to the new dataset by incorporating the recent performance evaluations [11]. At the same time, the *outdated* performance of hyperparameter configurations, evaluated on the former dataset(s), can still be considered to some extent when updating the new metamodel, or they can be progressively forgotten (the ML algorithm has better performance with other (possibly new) hyperparameters configurations).

Alternatively, characterizations (meta-features) of the dataset at hand [147] can be used to define a dataset similarity measure (based on, for instance, the Euclidean distance between the metafeature vectors of two datasets), so that we can transfer information from the most similar dataset to the new one. Moreover, together with prior evaluations (on previous datasets), we can train a meta-learner (in an ensemble fashion) to predict the performance of hyperparameter configurations (already observed or new ones) on a new dataset.

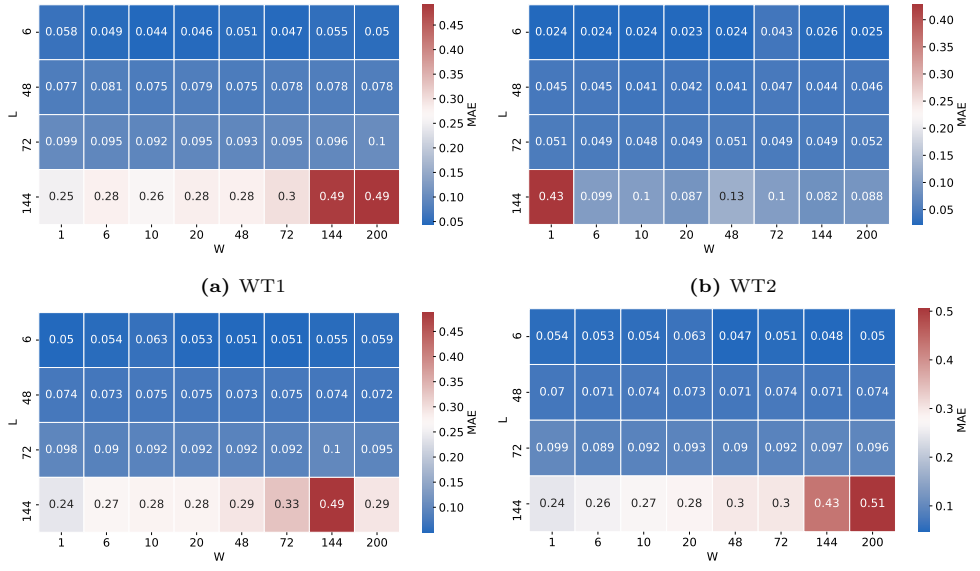
In theory, any of the model-based HPO methods presented in Chapter 3 and Chapter 4 (or in the existing literature) can be used in online learning scenarios, using transfer learning or meta-learning approaches. Yet, the computational effort for HPO can be avoided altogether by opting for parameterless forecasting models, such as LSTCN.

6.3.3 Results and discussion

Figure 6.5 shows an analysis of the influence of w and L on the model's behavior. The parameters were varied in the discrete set $w = \{1, 6, 10, 20, 48, 72, 144\}$ and $L = \{6, 48, 72, 144\}$, and the MAE computed on the test set was used for comparative purposes. The results did not show a large difference when changing w while keeping L fixed. However, the reduction in model performance was more evident when L increases, which is usual in time series forecasting models.

As mentioned, the knowledge used by the first STCN is extracted from a smoothed representation of the time series data we have. Nevertheless, we can start with a zero-filled matrix if such knowledge is not available. Figure 6.6 shows the MAE of the predictions in the training set of the four windmills in both settings. Starting from scratch (no knowledge about the data), the LSTCN's predictions have a large MAE in the first time patch. As new data is received, the network updates its knowledge and reduces the prediction error. In this simulation, we used five time patches such that each STCN block is fitted on the newly received data. The LSTCN model using general knowledge of the time series (assumed as a warm-up) generates small errors from the first time patch.

Figure 6.7 shows the distribution of weights in the W_1 and W_2 matrices obtained in the first five STCN blocks trained whether or not the initial prior knowledge is used (for the first windmill). In other words, we visualize the distributions of prior knowledge weights and the weights learned in each STCN block. If the LSTCN is trained using initial prior knowledge (W_1 in STCN of the first row, the first column of Figure 6.7), the network is able to adapt



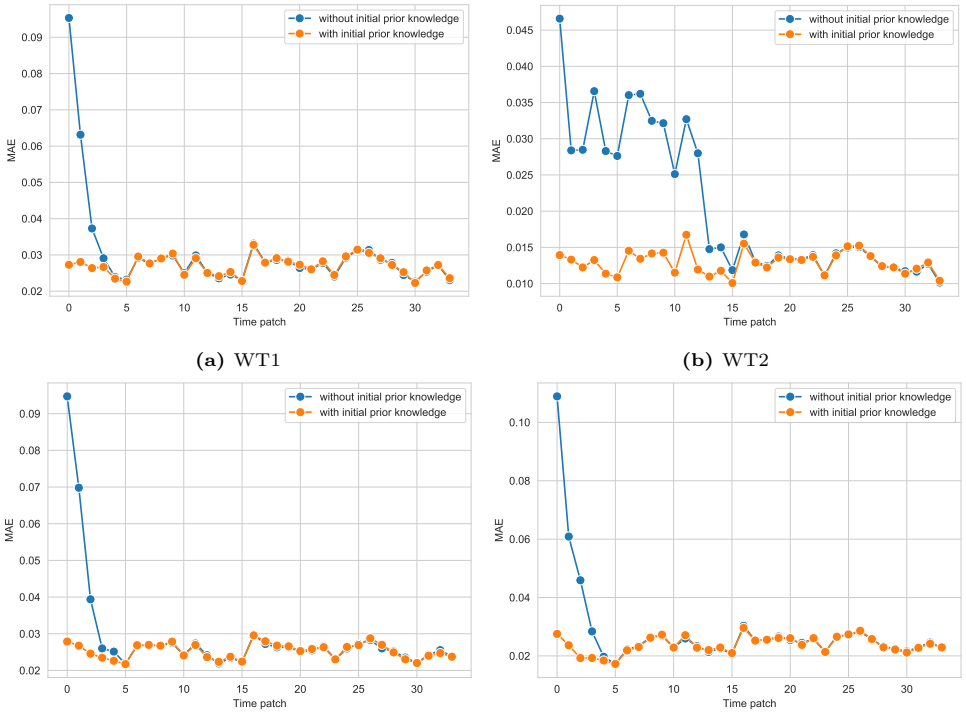
(a) WT1 **(b)** WT2

(c) WT3 **(d)** WT4

Figure 6.5: MAE values obtained by the LSTCN-based model when changing the w and L parameters. As expected, expanding the prediction horizon (that is to say, increasing the number of steps ahead to be predicted) leads to performance degradation of predictions. However, the model does not seem to be especially sensitive to the w parameter, except for larger L values where the error increases as the w gets larger.

to the new data made available. Therefore, the MAE of the first block should be lower compared to when no initial prior knowledge is used, since in this case the learning process has started from previous experience. On the other hand, the first STCN block will learn only from the data available at that moment if no prior knowledge is considered (first row, second column of Figure 6.7). As a consequence, the MAE of this first block is expected to be larger and only after some additional STCN blocks have been processed, the error should behave similarly to the ones obtained when the initial prior knowledge is considered (Figure 6.6a). Appendix D supports this finding by showing how the weights of the first fourth blocks changes in both scenarios.

Tables 6.2, 6.3 and 6.4 show the results for $L = 6, L = 48$ and $L = 72$, respectively. More explicitly, we report the average training and test errors, and the average training and test times (in seconds). The LSTCN model obtained the lowest MAE value when $L = 48$ and $L = 72$ (the lowest average test error for each windmill is highlighted in boldface). As we can see, VES had the lowest MAE value when $L = 6$, followed by LSTCN in second place. This is not surprising since statistical forecasting models such as VES get good forecasting results when the horizon is short. Although the forecasting error generally increases with the forecast horizon in the case of LSTCN, it is still relatively stable compared with the steep increase observed in VES when the horizon goes up to 48 and 72. Overall, these results allow us to conclude that our approach is able to produce better forecasting results when compared with well-established forecasting algorithms, and LSTCN performance is not compromised



(a) WT1 **(b)** WT2

(c) WT3 **(d)** WT4

Figure 6.6: MAE values obtained by the LSTCN-based model on the four windmill datasets with and without using initial prior knowledge. It can be noticed that the model needs to process more time patches to reduce the error when the model is initialized with a random weight matrix. If this knowledge is not available, the network will still produce good results provided it performs enough iterations.

too heavily when forecasting over a long horizon. It should be noted, however, that such a conclusion is attached to the fact that no hyper-parameter tuning was performed in our simulations.

Another clear advantage of LSTCN over these state-of-the-art algorithms is the reduced training and test times. Re-training the model quickly when a new piece of data arrives while retaining the knowledge we have learned so far is a key challenge in online learning settings. Recurrent neural models such as RNN, LSTM, and GRU use a backpropagation-based learning algorithm to compute the weights regulating the network behavior. The algorithm needs to iterate multiple times over the data with limited vectorization possibilities.

Overall, there is a trade-off between accuracy and training time when it comes to batch size in backpropagation-based learning. The smaller the batch size in the backpropagation learning, the more accurate the predictions are expected to be. However, smaller batch sizes make the training process slower. Another issue with gradient-based optimization methods is that they usually operate in a stochastic fashion, thus making them quite sensitive to the initial conditions. Notice that HMM also requires several iterations to build the probability transition matrix.

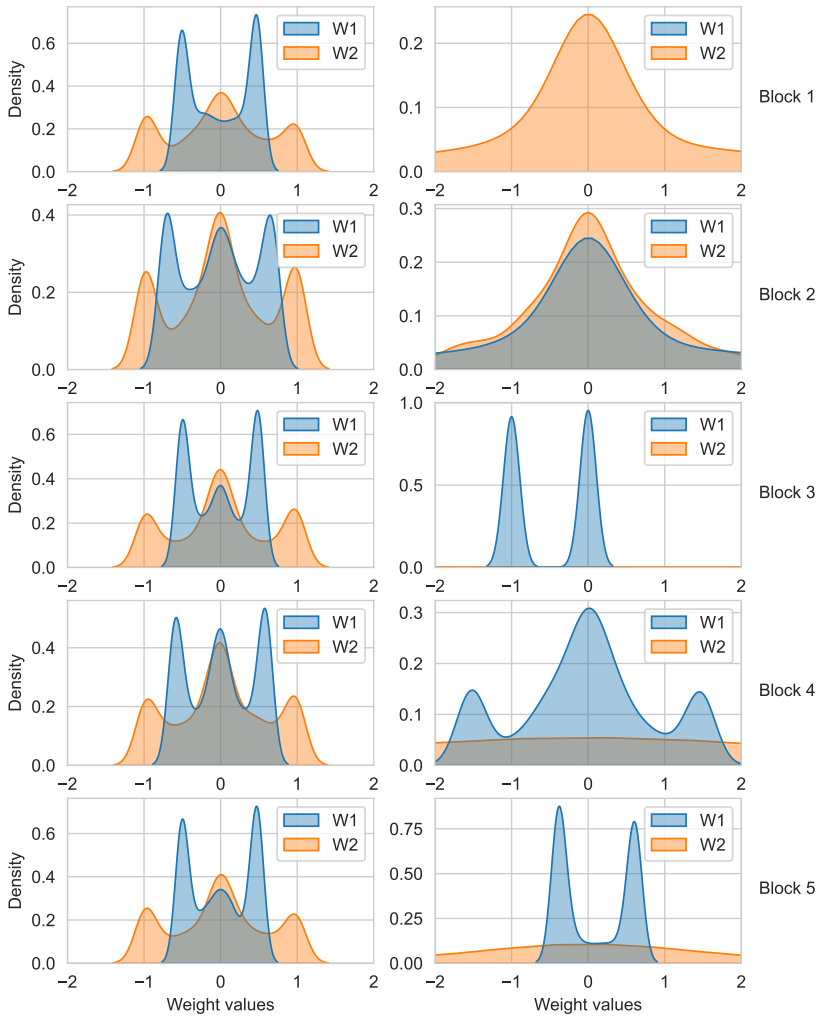
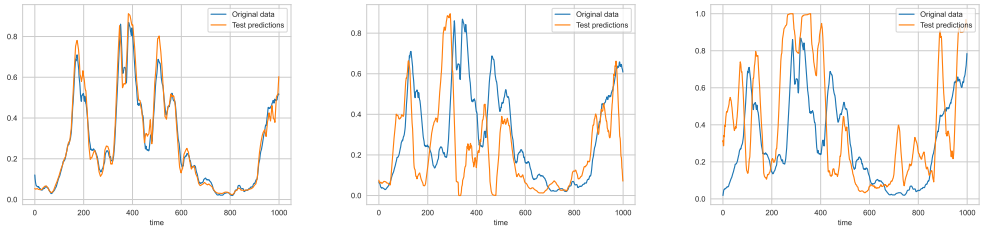


Figure 6.7: Distribution of weights for the first five STCN blocks trained in WT1 (Left) with initial prior knowledge and (Right) without initial prior knowledge. The first row corresponds to the first STCN block.

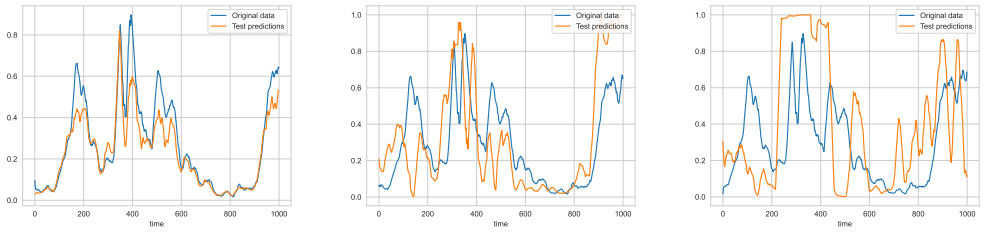


(a) $L = 6$ **(b)** $L = 48$ **(c)** $L = 72$

Figure 6.8: Moving average power predictions ($w = 24$) for the first windmill with (a) $L = 6$, (b) $L = 48$ and (c) $L = 72$.

Table 6.2: Results for the windmill case study for $L = 6$ (1 hour). The test time of VES is considered as part of the training time (library implementation)

	Model	Training error	Training time	Test error	Test time
WT1	LSTCN	0.0672	0.0091	0.0715	0.0255
	RNN	0.1087	0.6477	0.1321	1.3503
	LSTM	0.1036	1.7681	0.1258	2.6994
	GRU	0.1193	1.7162	0.1396	2.4513
	VES	0.0178	0.3702	**0.0460**	-
	HMM	0.0567	241.17	0.1267	91.08
WT2	LSTCN	0.0592	0.0091	0.0647	0.0261
	RNN	0.1086	0.6516	0.1209	1.3048
	LSTM	0.1035	1.7075	0.1147	2.5682
	GRU	0.1243	1.7718	0.1384	2.3741
	VES	0.0080	0.3913	**0.019**	-
	HMM	0.0381	333.61	0.0941	107.54
WT3	LSTCN	0.0474	0.0103	0.0564	0.0370
	RNN	0.1221	0.6418	0.1532	1.3171
	LSTM	0.1107	1.6932	0.1455	2.6672
	GRU	0.1219	1.6821	0.1499	2.3471
	VES	0.0177	0.3648	**0.0440**	-
	HMM	0.0683	318.11	0.1768	118.31
WT4	LSTCN	0.0602	0.0112	0.0666	0.0398
	RNN	0.1069	0.6609	0.1398	1.3507
	LSTM	0.0979	1.1241	0.1275	1.4140
	GRU	0.1162	1.6661	0.1405	2.3841
	VES	0.0172	0.3584	**0.0426**	-
	HMM	0.0534	367.05	0.1445	164.15



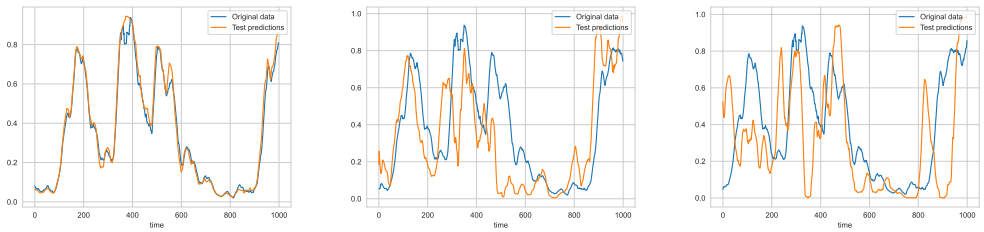
(a) $L = 6$ **(b)** $L = 48$ **(c)** $L = 72$

Figure 6.9: Moving average power predictions ($w = 24$) for the second windmill with (a) $L = 6$, (b) $L = 48$ and (c) $L = 72$.

To alleviate the problem caused by larger prediction horizons, we could increase the batch size (the number of instances in a time patch) in our model (or decrease the batch size of other recurrent models used for comparison purposes). In that way, we will have more data in each training process, which will likely lead to models with improved predictive power. Alternatively, we could adopt an incremental learning approach to reuse data concerning previous time patches as defined by a given window parameter. However, we should be aware that many online learning problems operate on volatile data that is just available for a short period.

Table 6.3: Results for the windmill case study for $L = 48$ (8 hours). The test time of VES is considered as part of the training time (library implementation)

	Model	Training error	Training time	Test error	Test time
WT1	LSTCN	0.0701	0.1958	**0.0721**	0.0408
	RNN	0.5963	17.7336	0.5966	4.5753
	LSTM	0.1248	229.1311	0.1290	30.7941
	GRU	0.1641	155.0201	0.1689	13.7719
	VES	0.0142	0.454681	0.0873	-
	HMM	0.0794	1432.65	0.0978	431.02
WT2	LSTCN	0.0594	0.7503	**0.0600**	0.0753
	RNN	0.5340	27.3444	0.5357	5.7755
	LSTM	0.1424	225.2245	0.1444	31.2621
	GRU	0.2317	137.3365	0.2324	15.9500
	VES	0.0076	0.4783	0.0612	-
	HMM	0.0388	1586.91	0.0821	441.19
WT3	LSTCN	0.0474	0.2333	**0.0482**	0.0470
	RNN	0.4991	20.1501	0.5048	5.8768
	LSTM	0.1332	236.3217	0.1401	31.8122
	GRU	0.1887	136.6582	0.1933	16.3957
	VES	0.0141	0.4444	0.09261822	-
	HMM	0.0761	1462.68	0.1582	413.93
WT4	LSTCN	0.0600	0.2463	**0.0623**	0.0490
	RNN	0.3407	37.1072	0.3425	5.8727
	LSTM	0.1254	214.9554	0.1318	19.9095
	GRU	0.1634	138.3381	0.1688	16.1307
	VES	0.0135	0.4437	0.0862	-
	HMM	0.0799	1971.42	0.1239	531.94



(a) $L = 6$ **(b)** $L = 48$ **(c)** $L = 72$

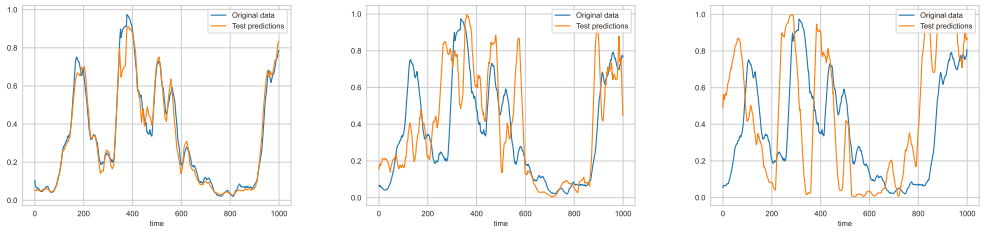
Figure 6.10: Moving average power predictions ($w = 24$) for the third windmill with (a) $L = 6$, (b) $L = 48$ and (c) $L = 72$.

6.4 Concluding note

In this chapter, we investigated the performance of Long Short-term Cognitive Networks to forecast windmill time series in online setting scenarios. This brand-new recurrent model system consists of a sequence of Short-term Cognitive Network blocks. Each of these blocks is trained with the available data at that moment in time such that the learned knowledge is propagated to the next blocks. Therefore, the network is able to adjust its knowledge to new information, which makes this model suitable for online settings since we retain the

Table 6.4: Results for the windmill case study for $L = 72$ (12 hours). The test time of VES is considered as part of the training time (library implementation)

	Model	Training error	Training time	Test error	Test time
WT1	LSTCN	0.0706	0.3770	**0.0726**	0.0430
	RNN	0.5187	45.8973	0.5219	9.5791
	LSTM	0.1376	714.2750	0.1418	53.8956
	GRU	0.1891	892.9639	0.1927	31.4892
	VES	0.0143	0.5042	0.1095	-
	HMM	0.0861	2967.08	0.1236	549.49
WT2	LSTCN	0.0597	1.0251	**0.0604**	0.0605
	RNN	0.6490	60.2035	0.6553	8.9644
	LSTM	0.1320	726.8744	0.1345	53.2367
	GRU	0.2304	806.9376	0.2316	38.6462
	VES	0.0078	0.5484	0.0722	-
	HMM	0.0488	2642.24	0.0718	622.14
WT3	LSTCN	0.0476	0.5155	**0.0494**	0.0545
	RNN	0.5524	47.0321	0.5618	9.4718
	LSTM	0.1441	768.9152	0.1518	54.4804
	GRU	0.1846	787.8666	0.1909	36.3055
	VES	0.0143	0.5106	0.1084	-
	HMM	0.0811	2916.17	0.1281	631.11
WT4	LSTCN	0.0605	0.5040	**0.0618**	0.0535
	RNN	0.5313	45.3868	0.5378	10.0499
	LSTM	0.1358	690.0783	0.1404	37.7908
	GRU	0.1813	809.8271	0.1895	31.4764
	VES	0.0136	0.4908	0.1014	-
	HMM	0.0887	2365.61	0.1112	589.81



(a) $L = 6$ **(b)** $L = 48$ **(c)** $L = 72$

Figure 6.11: Moving average power predictions ($w = 24$) for the fourth windmill with (a) $L = 6$, (b) $L = 48$ and (c) $L = 72$.

knowledge learned from previous learning processes.

The experiments conducted using four windmill datasets reported that our approach outperforms other state-of-the-art recurrent neural networks in terms of MAE. In addition, the proposed LSTCN-based model is significantly faster than these recurrent models when it comes to both training and test times. Such a feature is of paramount relevance when designing forecasting models operating in online learning modes. Regrettably, the overall performance of all forecasting models deteriorated when increasing the number of steps ahead to be predicted. While this result is not surprising, further efforts are needed to build

forecasting models with better scalability properties as defined by the prediction horizon.

It is worth mentioning that the proposed architecture for online time series forecasting is not restricted to windmill data. Instead, the architecture can be applied to any univariate or multivariate time series provided that the proper pre-processing steps are conducted.

— CHAPTER 7 —

THE BEGINNING OF THE END

Concluding remarks

When we apply Machine Learning and optimization algorithms to solve real-world problems, we often encounter three challenges: time constraints, data sparsity, and uncertainty. Time constraints refer to the limited amount of time we have to train our Machine learning models, evaluate our optimization objectives, or complete the entire optimization process. Data sparsity is another challenge that can arise due to various factors, such as limited resources or the high cost of collecting data. This can make it difficult to obtain sufficient data to train our models or guide the optimization of our objectives. In such situations, it is important to have efficient algorithms to maximize the utility of the available data. Finally, the presence of uncertainty in the problem at hand can further complicate the use of Machine Learning and optimization algorithms. To overcome these challenges, researchers and practitioners need to develop innovative techniques and methods to handle these issues effectively. This chapter outlines the key results of this thesis and future research directions to be accomplished in the near future.

7.1 Conclusions

In this research, we have presented novel methods in the context of hyperparameter and process bonding optimization, and time series forecasting. Bayesian optimization is a widely used technique in scenarios where the evaluation of objectives is costly, which is why it is the foundation of the proposed optimization methods. In the context of multi-objective optimization, Bayesian optimization can be extended to handle multiple objectives by constructing a model that can capture the trade-offs between them.

The first optimization algorithm presented in this thesis (GP_MOTPE) combines the predictor information (both predictor and predictor variance) obtained from a Gaussian Process Regression (GPR) model with heterogenous noise, and the sampling strategy performed by Multi-objective Tree Parzen Estimators (MOTPE). In this way, the algorithm should select new points that are likely to be non-dominated, and that are expected to cause the maximum improvement in the scalarized objective function. The experiments reported that this approach performed relatively well for analytical test functions, especially compared to ParEGO (using a GPR for noisy observations). In the HPO experiments, GP_MOTPE showed the best average rank w.r.t. the hypervolume computed on the validation set and showed promising reliability properties (small changes in hypervolume when the ML algorithm is evaluated on the test set). The observation that it outperforms the pure GPR-based

algorithm (which used a metaheuristic to maximize the infill criterion) is useful in its own right, as the optimization of infill criteria is known to be challenging.

Although the superiority of GP_MOTPE was demonstrated in terms of the increase in hypervolume, the inclusion of GPR to handle the uncertainty of the hyperparameter evaluation adds additional computational complexity to the HPO procedure and makes it difficult to handle mixed search space. Since TPE-based algorithms do not suffer from this issue, a more elegant alternative should directly handle noisy objective functions. As a result, we introduced a TPE modification for single-objective optimization to account for the performance variability of hyperparameter evaluations. The proposed modification uses the probability of being "bad" and "good" of the observed points to estimate the probability distribution of each hyperparameter in the input space. Therefore, the splitting procedure used by the noiseless TPE is no longer needed. The aforementioned probability of being "good" and "bad" uses the performance distributions observed in a cross-validation protocol and reflect the influence of each point on the density function estimate used to suggest a new input configuration. Experiments have shown that this modification is effective in terms of classification errors. Likewise, the result also showed that using small values for γ report better hyperparameter configurations; and although the size of the candidate set to sample from $l(x)$ was not an obvious choice, values smaller than 1000 candidates were preferred.

In addition to expensive and uncertain objectives, the optimization problem may also have several constraints. Two constrained BO algorithms were also introduced in this thesis to solve a bi-objective problem related to the adhesive bonding process of materials (maximizing break strength while minimizing production costs). The proposed approaches are shown to clearly outperform state-of-the-art evolutionary algorithms, which are commonly used in engineering design when solving general multi-objective, constrained problems. The difference lies in how the experimental design is guided throughout the search: the Bayesian approach selects infill points based on an acquisition function, which is related to the expected merit of the new infill point for optimization. The BO model ensures that the search focuses on infill points that have a high probability of being feasible. Moreover, the GPR model used to approximate the objective(s) accounts for the output (heterogenous) noise, whereas the evolutionary algorithms rely simply on the (uncertain) sample means as performance approximations. The success of evolutionary processes is largely dependent on the availability of a sufficient experimentation budget, which is not always the case in practice.

Online learning requires algorithms with short training times. While the available data may be enough to train the algorithm, it is not always accessible. As a result, the Machine Learning algorithm must have the ability to assimilate new information entering the system by adapting the acquired knowledge from previous training steps. We have shown that Long Short-term Cognitive Networks are suitable for such scenarios. In this sense, the experiments conducted using four windmill datasets reported that our approach outperforms other state-of-the-art recurrent neural networks in terms of forecasting errors. In addition, the proposed LSTCN-based model is significantly faster than these recurrent models when it comes to both training and test times. Such a feature is of paramount relevance when designing forecasting models operating in online learning modes.

Before concluding, it is worth mentioning that all the algorithms proposed in this thesis are not restricted to the use cases analyzed. The proposed architecture of the LSTCN can be applied to any univariate or multivariate time series provided that the proper pre-processing steps are conducted. Similarly, the optimization algorithms presented for single and multi-

objective optimization can be easily generalized to any other optimization problem where the objectives are expensive to evaluate and are affected by noise. In this case, an appropriate experimental design must be established to account for uncertainty and metamodel selection.

7.2 Recommendations for future research

As the field of multi-objective HPO is gaining speed, it presents diverse opportunities for further studies. Recent research has shown potential benefits in studying cheaply available (yet lower fidelity) information, obtained for instance by evaluating only a fraction of the training data or a small number of iterations. Low fidelity methods such as bandit-based approaches [93] have, to the best of our knowledge, not yet been applied in multi-objective HPO. Also, early stopping criteria [35] could be considered to ensure more intelligent use of the available computational budget. This has already been applied in single-objective optimization [84, 122], by considering the algorithm's learning curve: the training procedure for a given hyperparameter configuration is then stopped when adding further resources (training instance, iterations, training time, etc) is predicted to be futile. However, these techniques should also be adapted to accommodate the performance variability of the hyperparameter evaluation if they are included in the algorithms presented in this research. Nevertheless, the synergy of multi-fidelity and BO techniques is worth studying in further studies.

Evolutionary algorithms (EA) are undoubtedly widely used to solve optimization problems. This is quite striking, as such approaches require the (noiseless) evaluation of many points, and in many cases, such evaluation is expensive to perform. As we have shown in this research, BO-based techniques are better suitable for optimization problems where a limited evaluation budget exists and the objectives are affected by noise. However, further research on metamodel-assisted EA appears promising here. One would expect that such algorithms combine the best of two worlds, providing low computational cost (as the metamodel provides inexpensive function evaluations) along with a heuristic sampling of new points.

A typical characteristic of hyperparameter optimization is the mixed input space, with a combination of discrete, categorical, numerical, and potentially even conditional variables. Given that such mixed input spaces are non-obvious for algorithms using GPR (such as GP_MOTPE), we studied an adaptation of the original TPE algorithm (which naturally accounts for such complex search spaces) to handle noisy objectives. However, further research for GP_MOTPE could focus on the inclusion of specific techniques to (1) adapt the kernels of the GPR-based algorithms such that they can account for such mixed input spaces or (2) adjust the input space such that traditional kernels can be used. On the other hand, the proposed noisy TPE algorithm may also benefit from a multivariate kernel density estimation to better handle interaction effects in the input space [47].

As for the proposed forecasting workflow in an online learning setting, it was observed that the overall performance of all forecasting models deteriorated when increasing the number of steps ahead to be predicted. While this result is not surprising, further efforts are needed to build forecasting models with better scalability properties defined by the prediction horizon. We assumed that all variables studied in this research influence each other, which is not always accurate in practice. Therefore, it would be worth investigating the performance of LSTCN in problems where knowledge about this relationship is available.

This can be particularly interesting if simpler models are required and where, in addition to the temporal relationship, a spatial relationship is inherent in the problem (e.g., for mobility data forecasting).

APPENDICES

— APPENDIX A —

Additional materials from Chapter 3

A.1 Comparison of Hypervolume values computed in validation and test set

Table A.1: Comparison of the optimization algorithms according to the difference between the hypervolume computed using the HP evaluation in the validation set and then evaluated with the test set. The order relationship was determined by analyzing the difference between the hypervolume computed using the validation set and the hypervolume computed using the test set. The lower the rank the better.

Dataset ID	ML algorithm	Absolute differences in HV			Ranks		
		ParEGO	MOTPE	GP_MOTPE	ParEGO	MOTPE	GP_MOTPE
997	MLP	0.0415	0.0403	**0.0273**	3	2	1
841	MLP	0.2402	0.1055	**0.1022**	3	2	1
53	MLP	**0.5755**	0.7563	0.6928	1	3	2
814	MLP	0.0160	**0.0055**	0.0257	2	1	3
770	MLP	0.5268	**0.2053**	0.3606	3	1	2
778	MLP	0.6022	0.6949	**0.2575**	2	3	1
41945	MLP	**0.2951**	0.3501	0.3138	1	3	2
980	MLP	0.0476	**0.0162**	0.0309	3	1	2
871	MLP	0.6007	0.2550	**0.2209**	3	2	1
41146	MLP	0.3833	**0.2318**	0.4306	2	1	3
847	MLP	**0.1700**	0.1856	0.2338	1	2	3
803	MLP	0.1318	0.1421	**0.1170**	2	3	1
997	DT	0.0756	**0.0420**	0.0457	3	1	2
841	DT	0.0584	0.0729	**0.0338**	2	3	1
53	DT	0.2116	**0.1116**	0.1621	3	1	2
814	DT	0.0426	**0.0233**	0.0352	3	1	2
770	DT	**0.1173**	0.0167	0.0168	3	1	2
778	DT	0.2637	0.2640	**0.2610**	2	3	1
41945	DT	0.0993	**0.0181**	0.0340	3	1	2
980	DT	0.0670	0.0371	**0.0244**	3	2	1
871	DT	0.0995	**0.0243**	0.0389	3	1	2
41146	DT	0.0226	0.0189	**0.0187**	3	2	1
847	DT	**0.0084**	0.0206	0.0096	1	3	2
803	DT	0.0054	0.0054	**0.0039**	2	3	1

Continued on next page

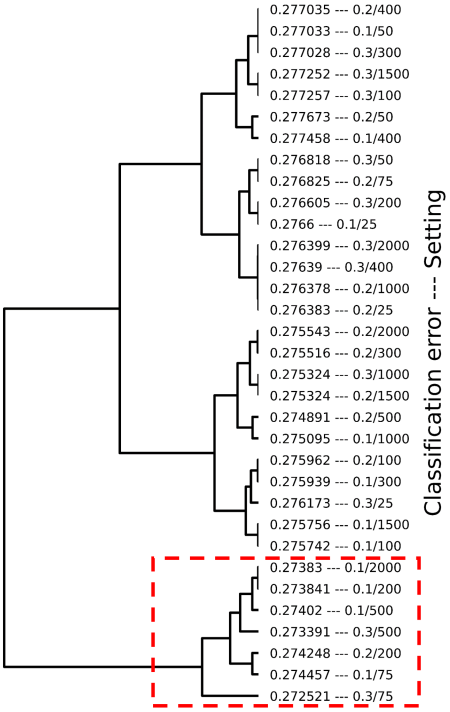
Dataset ID	ML algorithm	Absolute differences in HV			Ranks		
		ParEGO	MOTPE	GP_MOTPE	ParEGO	MOTPE	GP_MOTPE
997	SVM	0.0020	**0.0000**	**0.0000**	3	1	1
841	SVM	0.0219	0.0222	**0.0217**	2	3	1
53	SVM	0.1064	0.1029	**0.0986**	3	2	1
814	SVM	0.0075	**0.0071**	0.0085	2	1	3
770	SVM	0.0629	0.1463	**0.0593**	2	3	1
778	SVM	0.1181	**0.0592**	0.1181	2	1	2
41945	SVM	0.0700	**0.0351**	0.0859	2	1	3
980	SVM	**0.0155**	**0.0155**	**0.0155**	1	1	1
871	SVM	0.0196	**0.0107**	0.0337	2	1	3
41146	SVM	0.0243	0.0312	**0.0206**	2	3	1
847	SVM	**0.0187**	0.0283	0.0199	1	3	2
803	SVM	0.0103	**0.0094**	0.0120	2	1	3
				Mean rank	2.25	1.86	1.75

— APPENDIX B —

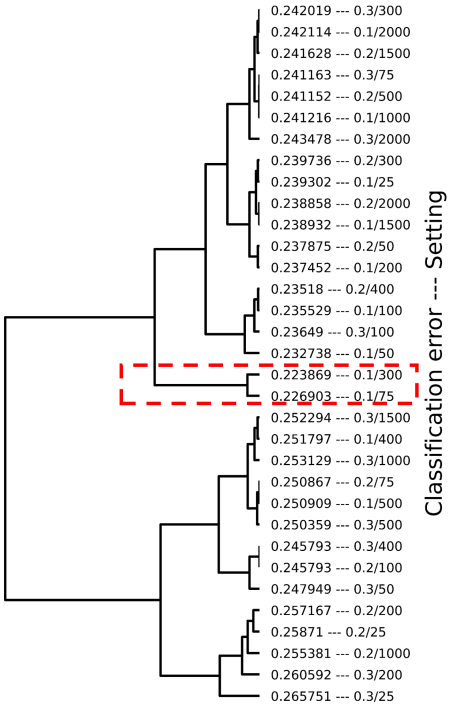
Additional materials from Chapter 4

B.1 Agglomerative clustering of the settings considered in the sensitivity analysis performed in Chapter 4

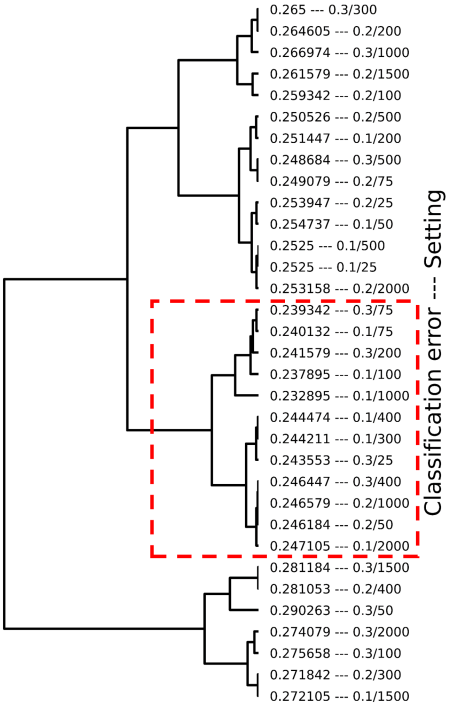
The hierarchical/agglomerative clustering was generated using the clustering module *scipy.cluster.hierarchy.linkage* of the library sckit-learn. The clustering is made based on the Euclidian distance between the mean classification error of the settings at the end of 10 macro-replications. We obtained 3 clusters; Cluster 1 with the settings with the lowest classification error (the best), Cluster 3 with setting with the largest classification error (the worst), and Cluster 2 with "medium" performances (the settings were not the worst but they were not the best either).



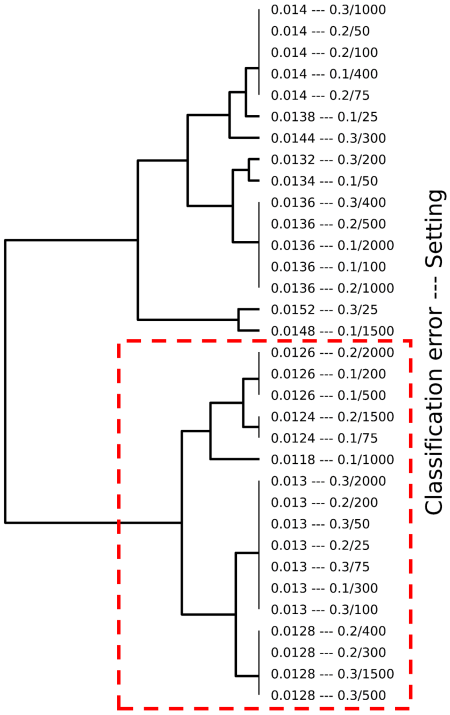
(a) Dataset 41945



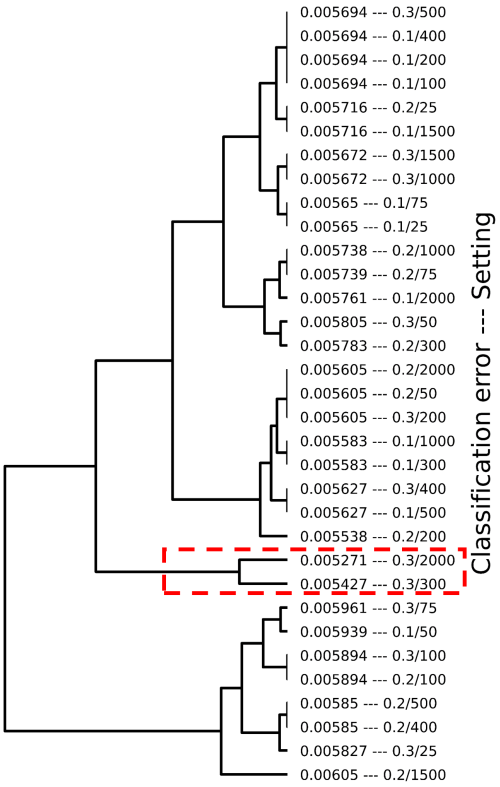
(b) Dataset 53



(c) Dataset 841



(d) Dataset 997



(e) Dataset 980

Figure B.1: Clustering of the settings considered in the sensitivity analysis of the TPE parameters with noisy observations. Dash red box highlights the settings included in Cluster 1.

— APPENDIX C —

Additional materials from Chapter 5

C.1 Constrained Expected Improvement (CEI)

The Constrained Expected Improvement (CEI, [158]) uses one OK metamodel to approximate the expensive objective and one metamodel to approximate each expensive constraint independently. Then, for a constrained optimization problem with c constraints

$$\begin{aligned} \min \quad & y(\mathbf{x}), \mathbf{x} \in \mathbb{R} \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, c \end{aligned} \tag{C.1}$$

the objective value of point $\mathbf{x}$ can be treated as a Gaussian random variable $\mathcal{N}\left(\widehat{y}(\mathbf{x}), \widehat{s}(x)\right)$ and the i-th constraint value of $\mathbf{x}$ can be also treated as a Gaussian random variable $\mathcal{N}\left(\widehat{g}_i(\mathbf{x}), \widehat{e}_i(x)\right),\ i = 1, 2, \ldots, c$. For this, $\widehat{y}$ and $\widehat{s}$ are the GP prediction and standard error of the objective function respectively, and $\widehat{g}_i$ and $\widehat{e}_i$ are the GP prediction and standard error of the i-th constraint function respectively.

Then, from Equation 5.1 we can transform the constraint and

$$g(\mathbf{x}) = 0.5 - Pf(\mathbf{x}) \leq 0 = Pf(\mathbf{x}) \geq 0.5 = \frac{Pf(\mathbf{x}) - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})} \geq \frac{0.5 - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})} \tag{C.2}$$

$$\begin{aligned} PoF(\mathbf{x}) &= Prob\left(\frac{Pf(\mathbf{x}) - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})} \geq \frac{0.5 - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})}\right) \\ &= 1 - Prob\left(\frac{Pf(\mathbf{x}) - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})} \leq \frac{0.5 - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})}\right) \\ &\equiv 1 - \Phi\left(\frac{0.5 - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})}\right) \end{aligned} \tag{C.3}$$

In case the GP standing for the objective and constraint function are mutually independent, the CEI can be obtained by combining the EI (to be consistent with our work we used MEI instead and the predictors defined in Equation 2.11 and Equation 2.13) and PoF as

$$\begin{aligned} \text{CMEI-SK}(\mathbf{x}) &= MEI(\mathbf{x}) \times PoF(\mathbf{x}) \\ &= \left[\left(\widehat{f}_{SK}(\mathbf{x}_{min}) - \widehat{f}_{SK}(\mathbf{x})\right)\Phi\left(\mathcal{D}\right) + \widehat{s}_{OK}(\mathbf{x})\phi\left(\mathcal{D}\right)\right] \\ &\quad \times \left[1 - \Phi\left(\frac{0.5 - \widehat{g}(\mathbf{x})}{\widehat{e}(\mathbf{x})}\right)\right] \end{aligned} \tag{C.4}$$

and

$$\mathcal{D} = \frac{\widehat{f}_{SK}(\mathbf{x}_{min}) - \widehat{f}_{SK}(\mathbf{x})}{\widehat{s}_{OK}(\mathbf{x})} \tag{C.5}$$

Note that this equation is different from our proposed Equation 5.5 in the derivation of the PoF. Figure C.1 shows that our proposed cMEI-SK got on average better Pareto fronts (higher hypervolume values) than that of CEI. This suggests that considering the uncertainty predicted by the GP may lead the optimization to points that do not cause an increase in the hypervolume. This was more evident when the improvement was measured with MEI (and fitting only one GP to the scalarized objectives).

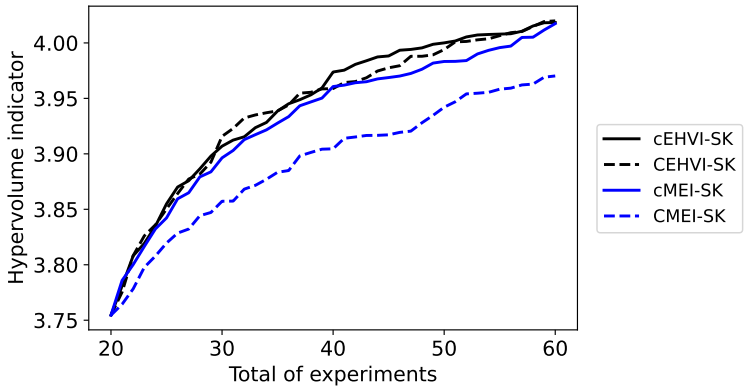
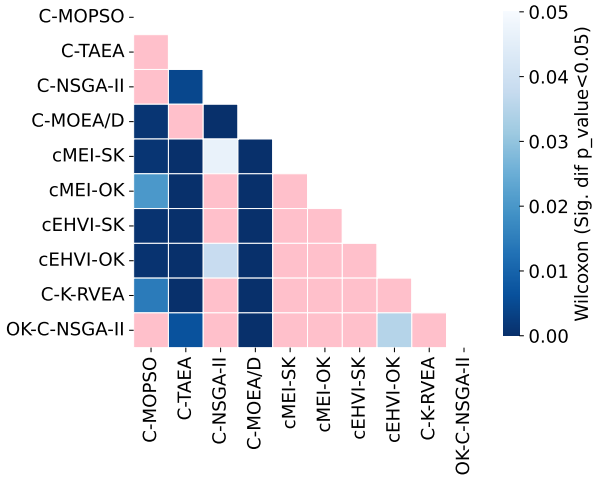
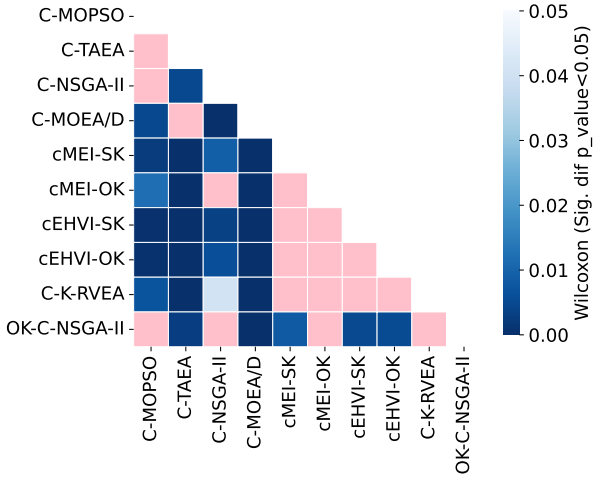


Figure C.1: Evolution of the mean hypervolume throughout the optimization (of 50 macro-replications). The reference point [production cost=3, break strength=4] is used to compute this metric

C.2 Wilcoxon test results



(a) HV



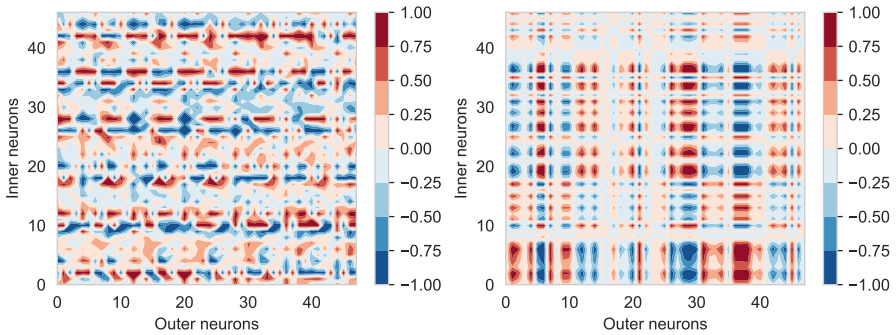
(b) IGD+

Figure C.2: Wilcoxon test results for significant differences between algorithms, using (a) Hypervolume and (b) IGD+ indicators. Pink color indicate no significant differences ($p\_value \geq 5\%$)

— APPENDIX D —

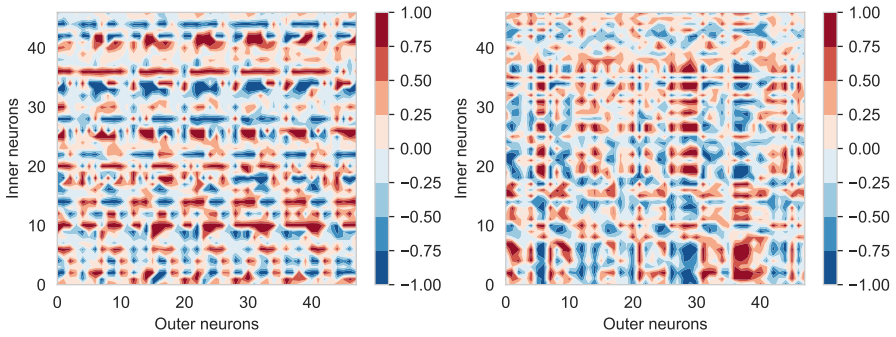
Additional materials from Chapter 6

Figure D.1 shows the overall behavior of weights connecting the inner neurons with the outer ones in the first three STCN blocks when initial prior knowledge is used (first column) and when the network starts training from scratch (second column). In this simulation, we apply the *tanh* function to the average of the W_1 and W_2 matrices for the sake of simplicity, thus resulting in an average layer. Here, inner and outer neurons refer to the leftmost and rightmost neurons, respectively. Observe that weights in the networks, when initial prior knowledge is not considered (second columns), are slowly changing towards a similar pattern to the one observed when initial prior knowledge is used (first column).

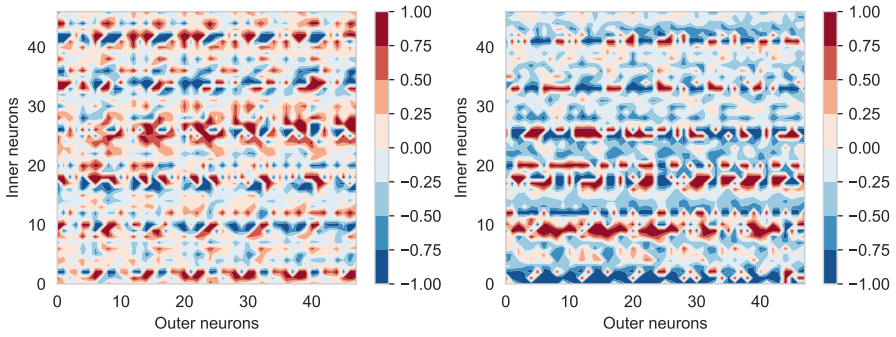


(a) STCN block 1

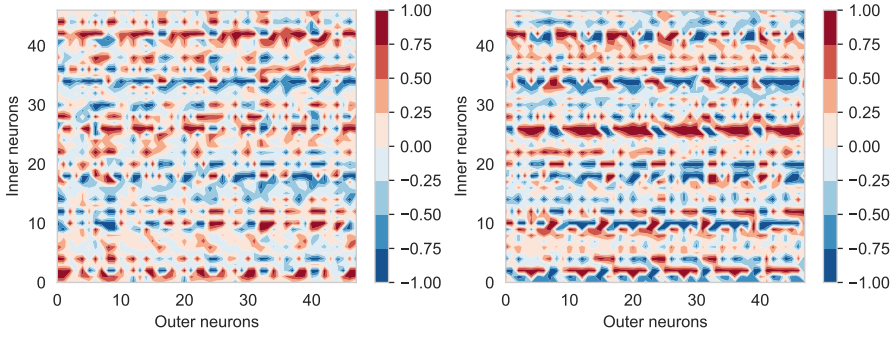
Figure D.1: Behavior of weights connecting the inner neurons with the outer ones in the first STCN block after applying the *tanh* function to the average of the W_1 and W_2. The first column and second column correspond to the network using initial prior knowledge and starting learning from scratch respectively.



(b) STCN block 2



(c) STCN block 3



(d) STCN block 4

Figure D.1: Behavior of weights connecting the inner neurons with the outer ones in the second, third, and fourth STCN block after applying the *tanh* function to the average of the W_1 and W_2. The first column and second column correspond to the network using initial prior knowledge and starting learning from scratch respectively. (cont.)

Bibliography

[1] Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U.R., et al.: A review of uncertainty quantification in deep learning: Techniques, applications and challenges. Information Fusion 76, 243–297 (2021)

[2] Aharon, M., Kagian, A., Somekh, O.: Adaptive online hyper-parameters tuning for ad event-prediction models. In: Proceedings of the 26th International Conference on World Wide Web Companion. pp. 672–679 (2017)

[3] ogly Aliev, R.A., Aliev, R.R.: Soft computing and its applications. World Scientific (2001)

[4] Andreopoulos, A., Tsotsos, J.K.: 50 years of object recognition: Directions forward. Computer vision and image understanding 117(8), 827–891 (2013), https://doi.org/10.1016/j.cviu.2013.04.005

[5] Ankenman, B., Nelson, B.L., Staum, J.: Stochastic kriging for simulation metamodeling. In: 2008 Winter Simulation Conference. pp. 362–370. IEEE (2008), https://doi.org/10.1109/WSC.2008.4736089

[6] Ankenman, B., Nelson, B.L., Staum, J.: Stochastic kriging for simulation metamodeling. Operations Research 58(2), 371–382 (2010), https://doi.org/10.1109/WSC.2008.4736089

[7] Auger, A., Bader, J., Brockhoff, D., Zitzler, E.: Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. Theoretical Computer Science 425, 75–103 (2012), https://doi.org/10.1016/j.tcs.2011.03.012

[8] Ayhan, M.S., Berens, P.: Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks. In: Medical Imaging with Deep Learning (2018)

[9] Bader, J., Zitzler, E.: Hype: An algorithm for fast hypervolume-based many-objective optimization. Evolutionary computation 19(1), 45–76 (2011)

[10] Barbaro, B.: Tuning hyperparameters for online learning. Ph.D. thesis, Case Western Reserve University (2018)

[11] Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: International conference on machine learning. pp. 199–207. PMLR (2013)

[12] Bates, S., Hastie, T., Tibshirani, R.: Cross-validation: what does it estimate and how well does it do it? Journal of the American Statistical Association (just-accepted), 1–22 (2023)

[13] Benavoli, A., Corani, G., Mangili, F.: Should we really use post-hoc tests based on mean-ranks? The Journal of Machine Learning Research 17(1), 152–161 (2016)

[14] Bengio, Y., Grandvalet, Y.: Bias in estimating the variance of k-fold cross-validation. Statistical modeling and analysis for complex data problems pp. 75–95 (2005)

[15] Berenji, H.R.: Treatment of uncertainty in artificial intelligence (1988)

[16] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: 25th annual conference on neural information processing systems (NIPS 2011). vol. 24. Neural Information Processing Systems Foundation (2011)

[17] Bhaumik, D., Crommelin, D., Kapodistria, S., Zwart, B.: Hidden markov models for wind farm power output. IEEE Transactions on Sustainable Energy 10(2), 533–539 (2019)

[18] Binois, M., Gramacy, R.B.: hetgp: Heteroskedastic gaussian process modeling and sequential design in r (2021)

[19] Bischl, B., Mersmann, O., Trautmann, H., Weihs, C.: Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation 20(2), 249–275 (2012), https://doi.org/10.1162/EVCO_a_00069

[20] Blum, A.: On-line algorithms in machine learning. Online algorithms: the state of the art pp. 306–325 (2005)

[21] Breiman, L.: Heuristics of instability and stabilization in model selection. The annals of statistics 24(6), 2350–2383 (1996)

[22] Brownlee, A.E., Wright, J.A.: Constrained, mixed-integer and multi-objective optimisation of building designs by nsga-ii with fitness approximation. Applied Soft Computing 33, 114–126 (2015), https://doi.org/10.1016/j.asoc.2015.04.010

[23] Cabrera-Hernández, L., Hernández, A.M., Gómez, M.M., Meneses, A.: Diversity-based selection of learning algorithms: a bagging approach. Investigación Operacional 42(4), 495–510 (2021)

[24] Cai, X., Hu, Z., Zhao, P., Zhang, W., Chen, J.: A hybrid recommendation system with many-objective evolutionary algorithm. Expert Systems with Applications 159, 113648 (2020), https://doi.org/10.1016/j.eswa.2020.113648

[25] Cao, L., Zhang, J., Wang, J., Qian, Z.: Intelligent fault diagnosis of wind turbine gearbox based on long short-term memory networks. In: 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE). pp. 890–895 (2019)

[26] Celikyilmaz, A., Turksen, I.B.: Modeling uncertainty with fuzzy logic. Studies in fuzziness and soft computing 240, 149–215 (2009)

[27] Che, Z., Purushotham, S., Cho, K., Sontag, D., Liu, Y.: Recurrent neural networks for multivariate time series with missing values. Scientific Reports 8(1), 6085 (2018)

[28] Chen, H., Liu, H., Chu, X., Liu, Q., Xue, D.: Anomaly detection and critical SCADA parameters identification for wind turbines based on LSTM-AE neural network. Renewable Energy 172, 829–840 (2021)

[29] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734 (2014)

[30] Chugh, T.: Scalarizing functions in bayesian multiobjective optimization. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2020)

[31] Chugh, T., Jin, Y., Miettinen, K., Hakanen, J., Sindhya, K.: A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization. IEEE Transactions on Evolutionary Computation 22(1), 129–142 (2016)

[32] Chugh, T., Sindhya, K., Hakanen, J., Miettinen, K.: A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. Soft Computing pp. 1–30 (2017), https://doi.org/10.1007/s00500-017-2965-0

[33] Couckuyt, I., Deschrijver, D., Dhaene, T.: Fast calculation of multiobjective probability of improvement and expected improvement criteria for pareto optimization. Journal of Global Optimization 60(3), 575–594 (2014), https://doi.org/10.1007/s10898-013-0118-2

[34] Cui, Y., Bangalore, P., Bertling Tjernberg, L.: A fault detection framework using recurrent neural networks for condition monitoring of wind turbines. Wind Energy 24(11), 1249–1262 (2021)

[35] Dai, Z., Yu, H., Low, B.K.H., Jaillet, P.: Bayesian optimization meets bayesian optimal stopping. In: International Conference on Machine Learning. pp. 1496–1506. PMLR (2019), \url{http://proceedings.mlr.press/v97/dai19a.html}

[36] Daulton, S., Balandat, M., Bakshy, E.: Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. Advances in Neural Information Processing Systems 33, 9851–9864 (2020)

[37] Daulton, S., Balandat, M., Bakshy, E.: Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. Advances in Neural Information Processing Systems 34, 2187–2200 (2021)

[38] Du, M., Yi, J., Mazidi, P., Cheng, L., Guo, J.: A parameter selection method for wind turbine health management through SCADA data. Energies 10(2) (2017)

[39] Dumont, V., Garner, C., Trivedi, A., Jones, C., Ganapati, V., Mueller, J., Perciano, T., Kiran, M., Day, M.: Hyppo: A surrogate-based multi-level parallelism tool for hyperparameter optimization. In: 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC). pp. 81–93. IEEE (2021)

[40] Ekbal, A., Saha, S.: Joint model for feature selection and parameter optimization coupled with classifier ensemble in chemical mention recognition. Knowledge-Based Systems 85, 37–51 (2015), https://doi.org/10.1016/j.knosys.2015.04.015

[41] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. The Journal of Machine Learning Research 20(1), 1997–2017 (2019)

[42] Emmerich, M.T., Deutz, A.H.: A tutorial on multiobjective optimization: fundamentals and evolutionary methods. Natural computing 17(3), 585–609 (2018)

[43] Emmerich, M.T., Deutz, A.H., Klinkenberg, J.W.: Hypervolume-based expected improvement: Monotonicity properties and exact computation. In: 2011 IEEE Congress of Evolutionary Computation (CEC). pp. 2147–2154. IEEE (2011), https://doi.org/10.1109/CEC.2011.5949880

[44] Emmerich, M.T., Giannakoglou, K.C., Naujoks, B.: Single-and multiobjective evolutionary optimization assisted by gaussian random field metamodels. IEEE Transactions on Evolutionary Computation 10(4), 421–439 (2006)

[45] Ertel, W.: Introduction to artificial intelligence. Springer (2018)

[46] Falkner, S., Klein, A., Hutter, F.: Combining hyperband and bayesian optimization. In: NIPS 2017 Bayesian Optimization Workshop (Dec 2017) (2017)

[47] Falkner, S., Klein, A., Hutter, F.: Bohb: Robust and efficient hyperparameter optimization at scale. In: International Conference on Machine Learning. pp. 1437–1446. PMLR (2018), \url{http://proceedings.mlr.press/v80/falkner18a.html}

[48] Feliot, P., Bect, J., Vazquez, E.: A bayesian approach to constrained single-and multi-objective optimization. Journal of Global Optimization 67(1-2), 97–133 (2017), https://doi.org/10.1007/s10898-016-0427-3

[49] Feng, B., Zhang, D., Si, Y., Tian, X., Qian, P.: A condition monitoring method of wind turbines based on long short-term memory neural network. In: 2019 25th International Conference on Automation and Computing (ICAC). pp. 1–4 (2019)

[50] Feurer, M., Hutter, F.: Hyperparameter optimization. In: Automated machine learning: methods, systems, challenges, pp. 3–33. Springer, Cham (2019)

[51] Forrester, A., Sobester, A., Keane, A.: Engineering Design via Surrogate Modelling (A Practical Guide). John Wiley and Sons, West Sussex, UK, 1st edn. (2008)

[52] Forrester, A.I., Keane, A.J., Bressloff, N.W.: Design and analysis of" noisy" computer experiments. AIAA journal 44(10), 2331–2339 (2006), https://doi.org/10.2514/1.20068

[53] Frazier, P.I.: Bayesian optimization. In: Recent advances in optimization and modeling of contemporary problems, pp. 255–278. Informs (2018)

[54] Gao, Y.l., Qu, M.: Constrained multi-objective particle swarm optimization algorithm. In: International Conference on Intelligent Computing. pp. 47–55. Springer (2012)

[55] Gardner Jr, E.S.: Exponential smoothing: The state of the art. Journal of forecasting 4(1), 1–28 (1985)

[56] Garrido, E.C., Hernández, D.: Predictive entropy search for multi-objective bayesian optimization with constraints. Neurocomputing 361, 50–68 (2019), https://doi.org/10.1016/j.neucom.2019.06.025

[57] Gawlikowski, J., Tassi, C.R.N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., et al.: A survey of uncertainty in deep neural networks. arXiv preprint arXiv:2107.03342 (2021)

[58] Gelbart, M.A., Adams, R.P., Hoffman, M.W., Ghahramani, Z., et al.: A general framework for constrained bayesian optimization using information-based search. Journal of Machine Learning Research 17(160), 1–53 (2016)

[59] Gonzalez, S.R., Jalali, H., Van Nieuwenhuyse, I.: A multiobjective stochastic simulation optimization algorithm. European Journal of Operational Research 284(1), 212–226 (2020), https://doi.org/10.1016/j.ejor.2019.12.014

[60] Gramacy, R.B.: Surrogates: Gaussian process modeling, design, and optimization for the applied sciences. Chapman and Hall/CRC (2020)

[61] Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International conference on machine learning. pp. 1321–1330. PMLR (2017)

[62] Hernández-Lobato, J.M., Gelbart, M., Hoffman, M., Adams, R., Ghahramani, Z.: Predictive entropy search for bayesian optimization with unknown constraints. In: International conference on machine learning. pp. 1699–1707. PMLR (2015)

[63] Hertel, L., Baldi, P., Gillen, D.L.: Reproducible hyperparameter optimization. Journal of Computational and Graphical Statistics 31(1), 84–99 (2022)

[64] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)

[65] Horn, D., Bischl, B.: Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2016), https://doi.org/10.1109/SSCI.2016.7850221

[66] Horn, D., Dagge, M., Sun, X., Bischl, B.: First investigations on noisy model-based multi-objective optimization. In: International Conference on Evolutionary Multi-Criterion Optimization. pp. 298–313. Springer (2017), https://doi.org/10.1007/978-3-319-54157-0_21

[67] Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing 70(1-3), 489–501 (2006)

[68] Huband, S., Hingston, P., Barone, L., While, L.: A review of multiobjective test problems and a scalable test problem toolkit. IEEE Transactions on Evolutionary Computation 10(5), 477–506 (2006)

[69] Hunter, S.R., Applegate, E.A., Arora, V., Chong, B., Cooper, K., Rincón-Guevara, O., Vivas-Valencia, C.: An introduction to multiobjective simulation optimization. ACM Trans. Model. Comput. Simul. 29(1), 7:1–7:36 (Jan 2019), http://doi.acm.org/10.1145/3299872

[70] Hutter, F., Kotthoff, L., Vanschoren, J.: Automated machine learning: methods, systems, challenges. Springer Nature (2019)

[71] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y.: Modified distance calculation in generational distance and inverted generational distance. In: International conference on evolutionary multi-criterion optimization. pp. 110–125. Springer (2015), https://doi.org/10.1007/978-3-319-15892-1_8

[72] Ishibuchi, H., Sakane, Y., Tsukamoto, N., Nojima, Y.: Simultaneous use of different scalarizing functions in moea/d. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 519–526 (2010)

[73] Jain, H., Deb, K.: An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. IEEE Transactions on evolutionary computation 18(4), 602–622 (2013)

[74] Jalali, H., Van Nieuwenhuyse, I., Picheny, V.: Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise. European Journal of Operational Research 261(1), 279–301 (2017)

[75] Jalali, H., Van Nieuwenhuyse, I., Picheny, V.: Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise. European Journal of Operational Research 261(1), 279–301 (2017), https://doi.org/10.1016/j.ejor.2017.01.035

[76] Jeong, S., Obayashi, S.: Efficient global optimization (ego) for multi-objective problem and data mining. In: 2005 IEEE congress on evolutionary computation. vol. 3, pp. 2138–2145. IEEE (2005)

[77] Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., Wang, Y., Dong, Q., Shen, H., Wang, Y.: Artificial intelligence in healthcare: past, present and future. Stroke and vascular neurology 2(4), 230–243 (2017), https://doi.org/10.1136/svn-2017-000101

[78] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global optimization 13(4), 455–492 (1998), https://doi.org/10.1023/A:1008306431147

[79] Jordens, J., Van Doninck, B., Satrio, N.R., Morales-Hernández, A., Couckuyt, I., Van Nieuwenhuyse, I., Witters, M.: Optimization of plasma-assisted surface treatment for adhesive bonding via artificial intelligence. In: 2nd International Conference on Industrial Applications of Adhesives 2022: Selected Contributions of IAA 2022. pp. 47–64. Springer (2022)

[80] Kleijnen, J.P.: Design and analysis of simulation experiments. Springer (2018)

[81] Klir, G., Wierman, M.: Uncertainty-based information: elements of generalized information theory, vol. 15. Springer Science & Business Media (1999)

[82] Knowles, J.: Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Transactions on Evolutionary Computation 10(1), 50–66 (2006), https://doi.org/10.1109/TEVC.2005.851274

[83] Koch, P., Wagner, T., Emmerich, M.T., Bäck, T., Konen, W.: Efficient multi-criteria optimization on noisy machine learning problems. Applied Soft Computing 29, 357–370 (2015), https://doi.org/10.1016/j.asoc.2015.01.005

[84] Kohavi, R., John, G.H.: Automatic parameter selection by minimizing estimated error. In: Machine Learning Proceedings 1995, pp. 304–312. Elsevier (1995), https://doi.org/10.1016/B978-1-55860-377-6.50045-1

[85] Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai. vol. 14, pp. 1137–1145. Montreal, Canada (1995)

[86] Kong, W., Dong, Z.Y., Jia, Y., Hill, D.J., Xu, Y., Zhang, Y.: Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Transactions on Smart Grid 10(1), 841–851 (2019)

[87] Kong, Z., Tang, B., Deng, L., Liu, W., Han, Y.: Condition monitoring of wind turbines based on spatio-temporal fusion of SCADA data by convolutional neural networks and gated recurrent units. Renewable Energy 146, 760–768 (2020)

[88] Kramti, S.E., Ben Ali, J., Saidi, L., Sayadi, M., Bechhoefer, E.: Direct wind turbine drivetrain prognosis approach using elman neural network. In: 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT). pp. 859–864 (2018)

[89] Lázaro-Gredilla, M., Titsias, M.K.: Variational heteroscedastic gaussian process regression. In: ICML. pp. 841–848 (2011)

[90] Le, T., Nguyen, T., Nguyen, V., Phung, D.: Dual space gradient descent for online learning. Advances in Neural Information Processing Systems 29 (2016)

[91] Lei, J., Liu, C., Jiang, D.: Fault diagnosis of wind turbine based on long short-term memory networks. Renewable Energy 133, 422–432 (2019)

[92] Li, K., Chen, R., Fu, G., Yao, X.: Two-archive evolutionary algorithm for constrained multiobjective optimization. IEEE Transactions on Evolutionary Computation 23(2), 303–315 (2018)

[93] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research 18(1), 6765–6816 (2017), \url{https://jmlr.org/papers/v18/16-558.html}

[94] Li, M., Yao, X.: Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Computing Surveys 52(2), 26 (2019)

[95] Li, M., Li, G., Azarm, S.: A kriging metamodel assisted multi-objective genetic algorithm for design optimization. Journal of Mechanical Design 130(3) (2008), https://doi.org/10.1115/1.2829879

[96] Li, M., Yao, X.: Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Computing Surveys (CSUR) 52(2), 1–38 (2019), https://doi.org/10.1145/3300148

[97] Lin, C.H.: Recurrent modified elman neural network control of PM synchronous generator system using wind turbine emulator of pm synchronous servo motor drive. International Journal of Electrical Power & Energy Systems 52, 143–160 (2013)

[98] Lin, C.H.: Wind turbine driving a PM synchronous generator using novel recurrent Chebyshev neural network control with the ideal learning rate. Energies 9(6) (2016)

[99] Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Marben, J., Müller, P., Hutter, F.: Boah: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters. arXiv:1908.06756 [cs.LG] (2019)

[100] Loka, N., Couckuyt, I., Garbuglia, F., Spina, D., Van Nieuwenhuyse, I., Dhaene, T.: Bi-objective bayesian optimization of engineering problems with cheap and expensive cost functions. Engineering with Computers pp. 1–11 (2022)

[101] López, E., Valle, C., Allende-Cid, H., Allende, H.: Comparison of recurrent neural networks for wind power forecasting. In: Pattern Recognition. pp. 25–34 (2020)

[102] López-Ibánez, M., Paquete, L., Stützle, T.: Exploratory analysis of stochastic local search algorithms in biobjective optimization. In: Experimental methods for the analysis of optimization algorithms, pp. 209–222. Springer (2010), https://doi.org/10.1007/978-3-642-02538-9_9

[103] Losing, V., Hammer, B., Wersing, H.: Incremental on-line learning: A review and comparison of state of the art algorithms. Neurocomputing 275, 1261–1274 (2018)

[104] Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1), 18 (2016), https://doi.org/10.1007/s13721-016-0125-6

[105] Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26(6), 369–395 (2004), https://doi.org/10.1007/s00158-003-0368-6

[106] Meckesheimer, M., Barton, R.R., Simpson, T., Limayem, F., Yannou, B.: Metamodeling of combined discrete/continuous responses. AIAA journal 39(10), 1950–1959 (2001)

[107] Miettinen, K.: Nonlinear multiobjective optimization, vol. 12. Springer Science & Business Media (1999)

[108] Miettinen, K., Mäkelä, M.M.: On scalarizing functions in multiobjective optimization. OR spectrum 24(2), 193–213 (2002), https://doi.org/10.1007/s00291-001-0092-9

[109] Mishra, S., Bordin, C., Taharaguchi, K., Palu, I.: Comparison of deep learning models for multivariate prediction of time series wind power generation and temperature. Energy Reports 6, 273–286 (2020)

[110] Mishra, V.K., Rajagopalan, A.: A novel extreme value theory based approach to hyperparameter optimization. Procedia Computer Science 218, 2411–2419 (2023)

[111] Mitchell, M.: An introduction to genetic algorithms. MIT press (1998)

[112] Mitchell, T.M., et al.: Machine learning. Burr Ridge, IL: McGraw Hill 45(37), 870–877 (1997)

[113] Morales-Hernández, A., Nápoles, G., Jastrzebska, A., Salgueiro, Y., Vanhoof, K.: Online learning of windmill time series using long short-term cognitive networks. Expert Systems with Applications 205, 117721 (2022)

[114] Morales-Hernández, A., Van Nieuwenhuyse, I., Nápoles, G.: Multi-objective hyperparameter optimization with performance uncertainty. In: Optimization and Learning: 5th International Conference, OLA 2022, Syracuse, Sicilia, Italy, July 18–20, 2022, Proceedings. pp. 37–46. Springer (2022)

[115] Morales-Hernández, A., Van Nieuwenhuyse, I., Rojas Gonzalez, S.: A survey on multi-objective hyperparameter optimization algorithms for machine learning. Artificial Intelligence Review pp. 1–51 (2022)

[116] Muñoz-González, L., Lázaro-Gredilla, M., Figueiras-Vidal, A.R.: Heteroscedastic gaussian process regression using expectation propagation. In: 2011 IEEE International Workshop on Machine Learning for Signal Processing. pp. 1–6. IEEE (2011)

[117] Nápoles, G., Grau, I., Jastrzebska, A., Salgueiro, Y.: Long short-term cognitive networks. Neural Computing and Applications 34(19), 16959–16971 (2022)

[118] Nápoles, G., Vanhoenshoven, F., Vanhoof, K.: Short-term cognitive networks, flexible reasoning and nonsynaptic learning. Neural Networks 115, 72–81 (2019)

[119] Netrapalli, P.: Stochastic gradient descent and its variants in machine learning. Journal of the Indian Institute of Science 99(2), 201–213 (2019)

[120] Ozaki, Y., Tanigaki, Y., Watanabe, S., Onishi, M.: Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference. pp. 533–541 (2020), https://doi.org/10.1145/3377930.3389817

[121] Phillips, P.J., Flynn, P.J., Scruggs, T., Bowyer, K.W., Chang, J., Hoffman, K., Marques, J., Min, J., Worek, W.: Overview of the face recognition grand challenge. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). vol. 1, pp. 947–954. IEEE (2005), https://doi.org/10.1109/CVPR.2005.268

[122] Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 23–32 (1999), https://doi.org/10.1145/312129.312188

[123] Qian, P., Tian, X., Kanfoud, J., Lee, J.L.Y., Gan, T.H.: A novel condition monitoring method of wind turbines based on long short-term memory neural network. Energies 12(18) (2019)

[124] Qin, S., Sun, C., Jin, Y., Zhang, G.: Bayesian approaches to surrogate-assisted evolutionary multi-objective optimization: A comparative study. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 2074–2080 (2019)

[125] Qu, K., Si, G., Sun, X., Lian, W., Huang, Y., Li, P.: Time series simulation for multiple wind farms based on hmms and regular vine copulas. Journal of Renewable and Sustainable Energy 13(2), 023311 (2021)

[126] Quan, N., Yin, J., Ng, S.H., Lee, L.H.: Simulation optimization via kriging: a sequential search using expected improvement with computing budget constraints. Iie Transactions 45(7), 763–780 (2013)

[127] Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)

[128] Rajagopal, A., Joshi, G.P., Ramachandran, A., Subhalakshmi, R., Khari, M., Jha, S., Shankar, K., You, J.: A deep learning model based on multi-objective particle swarm optimization for scene classification in unmanned aerial vehicles. IEEE Access 8, 135383–135393 (2020), https://doi.org/10.1109/ACCESS.2020.3011502

[129] Ren, P., Xiao, Y., Chang, X., Huang, P.Y., Li, Z., Chen, X., Wang, X.: A comprehensive survey of neural architecture search: Challenges and solutions. ACM Computing Surveys (CSUR) 54(4), 1–34 (2021)

[130] Richter, J., Kotthaus, H., Bischl, B., Marwedel, P., Rahnenführer, J., Lang, M.: Faster model-based optimization through resource-aware scheduling strategies. In: International Conference on Learning and Intelligent Optimization. pp. 267–273. Springer (2016), https://doi.org/10.1007/978-3-319-50349-3_22

[131] Riganti-Fulginei, F., Sun, Z., Sun, H.: Health status assessment for wind turbine with recurrent neural networks. Mathematical Problems in Engineering 2018, 6972481 (2018)

[132] Rojas Gonzalez, S., Jalali, H., Nieuwenhuyse, I.V.: A multiobjective stochastic simulation optimization algorithm. European Journal of Operational Research 284(1), 212–226 (2020), https://doi.org/10.1016/j.ejor.2019.12.014

[133] Rojas Gonzalez, S., Jalali, H., Van Nieuwenhuyse, I.: A multiobjective stochastic simulation optimization algorithm. European Journal of Operational Research 284(1), 212–226 (2020), https://doi.org/10.1016/j.ejor.2019.12.014

[134] Rojas-Gonzalez, S., van Nieuwenhuyse, I.: A survey on kriging-based infill algorithms for multiobjective simulation optimization. Computers and Operations Research 116, 104869 (2020), https://doi.org/10.1016/j.cor.2019.104869

[135] Rojas-Gonzalez, S., Van Nieuwenhuyse, I.: A survey on kriging-based infill algorithms for multiobjective simulation optimization. Computers & Operations Research 116, 104869 (2020), https://doi.org/10.1016/j.cor.2019.104869

[136] Schaffer, C.: Selecting a classification method by cross-validation. Machine learning 13, 135–143 (1993)

[137] Scott, D.W.: Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons (2015)

[138] Sherstinsky, A.: Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena 404, 132306 (2020)

[139] de Silva, A.: Vector Exponential Smoothing, pp. 287–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-71918-2_17

[140] Silverman, B.W.: Density estimation for statistics and data analysis. Routledge (1998)

[141] Sobester, A., Forrester, A., Keane, A.: Engineering design via surrogate modelling: a practical guide. John Wiley & Sons (2008)

[142] Strobelt, H., Gehrmann, S., Pfister, H., Rush, A.M.: Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. IEEE Transactions on Visualization and Computer Graphics 24(1), 667–676 (2018)

[143] Talbi, E.G.: Automated design of deep neural networks: A survey and unified taxonomy. ACM Computing Surveys (CSUR) 54(2), 1–37 (2021), https://doi.org/10.1145/3439730

[144] Tautz-Weinert, J., Watson, S.J.: Using scada data for wind turbine condition monitoring–a review. IET Renewable Power Generation 11(4), 382–394 (2017)

[145] Tharwat, A.: Classification assessment methods. Applied Computing and Informatics (2020), https://doi.org/10.1016/j.aci.2018.08.003

[146] de Toro, F., Ros, E., Mota, S., Ortega, J.: Multi-objective optimization evolutionary algorithms applied to paroxysmal atrial fibrillation diagnosis based on the k-nearest neighbours classifier. In: Ibero-American Conference on Artificial Intelligence. pp. 313–318. Springer (2002), https://doi.org/10.1007/3-540-36131-6_32

[147] Vanschoren, J.: Meta-learning. In: Automated machine learning: methods, systems, challenges, pp. 35–61. Springer, Cham (2019)

[148] Viana, F.A.: Things you wanted to know about the latin hypercube design and were afraid to ask. In: 10th World Congress on Structural and Multidisciplinary Optimization. vol. 19. sn (2013)

[149] Villmann, T., Kaden, M., Lange, M., Stürmer, P., Hermann, W.: Precision-recall-optimization in learning vector quantization classifiers for improved medical classification systems. In: 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 71–77. IEEE (2014)

[150] Wang, Y., Xie, D., Wang, X., Zhang, Y.: Prediction of wind turbine-grid interaction based on a principal component analysis-long short term memory model. Energies 11(11) (2018)

[151] Weerakody, P.B., Wong, K.W., Wang, G., Ela, W.: A review of irregular time series data handling with gated recurrent neural networks. Neurocomputing 441, 161–178 (2021)

[152] Williams, C.K., Rasmussen, C.E.: Gaussian processes for machine learning, vol. 2. MIT press Cambridge, MA (2006)

[153] Wilson, J., Hutter, F., Deisenroth, M.: Maximizing acquisition functions for bayesian optimization. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper/2018/file/498f2c21688f6451d9f5fd09d53edda7-Paper.pdf

[154] Xiang, L., Wang, P., Yang, X., Hu, A., Su, H.: Fault detection of wind turbine based on SCADA data analysis using CNN and LSTM with attention mechanism. Measurement 175, 109094 (2021)

[155] Xue, X., Xie, Y., Zhao, J., Qiang, B., Mi, L., Tang, C., Li, L.: Attention mechanism-based CNN-LSTM model for wind turbine fault prediction using SSN ontology annotation. Wireless Communications and Mobile Computing 2021, 6627588 (2021)

[156] Yarat, S., Senan, S., Orman, Z.: A Comparative Study on PSO with Other Metaheuristic Methods, pp. 49–72. Springer International Publishing, Cham (2021)

[157] Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: Artificial intelligence and statistics. pp. 1077–1085. PMLR (2014)

[158] Zhan, D., Xing, H.: Expected improvement for expensive optimization: a review. Journal of Global Optimization 78(3), 507–544 (2020), https://doi.org/10.1007/s10898-020-00923-x

[159] Zhang, X.M., Han, Q.L., Ge, X., Ding, D.: An overview of recent developments in lyapunov-krasovskii functionals and stability criteria for recurrent neural networks with time-varying delays. Neurocomputing 313, 392–401 (2018)

[160] Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms—a comparative case study. In: Parallel Problem Solving from Nature—PPSN V: 5th International Conference Amsterdam, The Netherlands September 27–30, 1998 Proceedings 5. pp. 292–301. Springer (1998)

Publications

- **Morales-Hernández, A.**, Van Nieuwenhuyse, I., Rojas, S., Jordens, J., Witters, M., Van Doninck, B. (2023). Expensive multi-objective optimization of adhesive bonding process in constrained settings. In: *Optimization and Learning. OLA 2023. Communications in Computer and Information Science*, (*accepted for publication*).

- **Morales-Hernández, A.**, Rojas, S., Van Nieuwenhuyse, I., Couckuyt, I., Jordens, J., Witters, M., Van Doninck, B. (2023). Bayesian multi-objective optimization of process design parameters in constrained settings with noise: an engineering design application. (*submitted*).

- **Morales-Hernández, A.**, Van Nieuwenhuyse, I., Rojas, S. (2022). A survey on multi-objective hyperparameter optimization for Machine Learning. *Artificial Intelligence review*, `https://doi.org/10.1007/s10462-022-10359-2`

- **Morales-Hernández, A.**, Nápoles G., Jastrzebska A., Salgueiro Y., Vanhoof K. (2022) Online learning of windmill time series using Long Short-term Cognitive Networks. *Expert Systems with Applications*. `https://doi.org/10.1016/j.eswa.2022.117721`

- Jastrzebska A, **Morales-Hernández A.**, Nápoles G., Salgueiro Y, Vanhoof K. (2022) Measuring wind turbine health using fuzzy-concept-based drifting models. *Renewable Energy*. 190, 730–740, `https://doi.org/10.1016/j.renene.2022.03.116`.

- Jordens, J., Van Doninck, B., Satrio, N. R., **Morales-Hernández A.**, Couckuyt, I., Van Nieuwenhuyse, I., Witters, M. (2023). Optimization of Plasma-Assisted Surface Treatment for Adhesive Bonding via Artificial Intelligence. In *2nd International Conference on Industrial Applications of Adhesives* pp. 47-64. `https://doi.org/10.1007/978-3-031-11150-1_4`

- **Morales-Hernández, A.**, Van Nieuwenhuyse, I., Nápoles, G. (2022). Multi-objective Hyperparameter Optimization with Performance Uncertainty. In: *Optimization and Learning. OLA 2022. Communications in Computer and Information Science*, vol 1684. `https://doi.org/10.1007/978-3-031-22039-5_4`