EDSAF: Event-Driven Surrogate-Assisted Framework for Real-Time Optimization

Arne De Temmerman[a,b,*], **Yari Depreeuw**[a] **and Mathias Verbeke**[a,b]

[a]M-Group, KU Leuven, Spoorwegstraat 12, Bruges, Belgium
[b]Flanders Make@KU Leuven, Belgium
ORCID (Arne De Temmerman): https://orcid.org/0000-0002-4989-3810

Abstract. Surrogate-assisted optimization tackles a significant challenge in population-based optimization methods by reducing the number of evaluations required for expensive problems. However, while surrogates excel in evaluation efficiency, they compromise on runtime efficiency due to the computational requirements of using extensive data archives. These data archives result in slow feedback times during the selection, fitting, and optimization of the surrogate model. The slow feedback creates a significant downtime in the utilization of the problem as it awaits new proposals to be generated. To address this issue, we introduce the Event-Driven Surrogate-Assisted Framework (EDSAF). This framework enhances problem evaluation efficiency by parallelizing time-consuming optimization steps and leverages a trigger-based framework. It minimizes the time gap between receiving new problem evaluations and generating new proposals in response, ensuring a quick response to incoming information from the system for more effective problem utilization. EDSAF is algorithm- and problem-agnostic, making it applicable across a wide range of unconstrained, multi-objective optimization problems. In comparison with standard surrogate-assisted optimization, our results demonstrate a substantial improvement in problem utilization, resulting in reduced runtime while preserving efficient convergence on a set of multi-objective benchmarks.

1 Introduction

A near-perfect solution can be found with enough attempts and sufficient time. In most real-world optimization problems, though, a non-trivial cost exists to execute and evaluate the objective and constraint functions, also referred to as expensive optimization problems (EOP). EOPs in research and application fields such as Agriculture [18, 31], Engineering [37], Health Care [27], or Computer Science [26], are increasingly being tackled by applying heuristic search methods. These problems are characterized by their expensive solution evaluations (ESE), including running simulations (Computational Fluid Dynamics [1], Finite Element Analysis [34]), real-world parameter optimization (process setting optimization [30], material composition design [32]), or hyperparameter optimization [33]. These ESEs are black-box in nature with no gradient information available and set the unique challenges of often having multiple, incommensurable objectives with a vast feature set that is expensive to evaluate and search.

[*] Corresponding Author. Email: arne.detemmerman@kuleuven.be

Evolutionary computation (EC) methods are often a good fit for non-convex, non-linear, constrained, multi-objective, and uncertain cost functions. However, EC methods do have some limitations, mainly related to convergence efficiency. This is addressed by Surrogate Assisted Optimization (SAO) [21]. These surrogates model the objective space from previously explored solutions and use a less expensive approximation to improve the convergence of the optimization search.

The cost of an EOP can be determined in several ways. A common method is to account for the number of evaluations of the problem. For each evaluation, a prior cost estimation or post-hoc cost measurement can be determined. The total cost is then determined as the cost per evaluation multiplied by the number of evaluations. This cost can be minimized by either reducing the cost for each evaluation or reducing the number of total evaluations. This cost-per-evaluation is often not an accurate representation of real-world problems where the cost is not only defined per unit but also as an overall overhead cost. For instance, the cost of running simulations is not based on the CPU-second (i.e., the number of seconds all the cores are maximally used) but the allocated time for a compute instance for running the complete experiment. In the optimization of real-world manufacturing processes, both the number of scrap products and the total time that the system is unavailable contribute to the total cost. This can be defined as the *problem utilization*, the actual number of evaluations compared to the maximum possible number of evaluations per time unit. As the problem utilization decreases, the total cost (and, therefore, the cost per evaluation) will increase.

Surrogate-assisted algorithms trade-in problem utilization for evaluation efficiency. The process of fitting, updating, or searching an approximation using data-driven surrogate models takes considerable time. During this phase, once all evaluations are received, the evaluation step is paused while waiting for new proposal solutions. As these proposals arrive after a certain duration, the focus shifts back to the evaluation while the optimizer waits for all the evaluations to happen. A surrogate update strategy based on incoming data frequency and that distributes the computationally intensive processes would allow a decrease in the loop time between receiving new information and suggesting proposals based on this information. This decrease in loop time has several advantages:

- The proposals based on the last iteration can be evaluated sooner on the EOP.
- Dynamically selecting the size of the population for evaluation based on the iteration speed of the optimizer allows a maximized

problem utilization. This also addresses handling non-constant evaluation durations.
- The speed of the optimization step (and therefore the number of evaluations per time unit) will become less dependent on the size of the data archive.

We propose a novel Event-Driven Surrogate-Assisted Framework (EDSAF), allowing efficient surrogate-assisted optimization of population-based algorithms.

The source code of the framework is provided through the pyEDSAF package. [1]

pyEDSAF provides a framework to define asynchronous problems with customizable function evaluation calls and waitable triggers. The user can implement their own communication and parallelization to the ESE, making it highly usable for a large number of applications. This work builds upon the popular Python packages Pysamoo and Pymoo from Blank et al. [6, 8] that provides GPSAF as a surrogate provider to enable an optimization algorithm-agnostic implementation. Pymoo is a very popular multi-objective optimization toolkit with more than 3.150.000 downloads to date, providing a broad range of test problems and optimization algorithms for single- and multi-objective, constrained, and unconstrained test problems. This framework supports additional utility software, such as problem initialization with historical data, result visualization, and result metrics. Pysamoo extends on Pymoo with a number of surrogate-assisted frameworks, allowing a computationally efficient implementation of the broad package of Pymoo. pyEDSAF maintains a slim profile, extending the capabilities of these frameworks to handle real-world optimization of time-expensive problems.

EDSAF is an asynchronous implementation of the Generalized Probabilistic Surrogate-Assisted Framework (GPSAF) [7] that provides a more evaluation-centric vision. An effort is made to improve the problem utilization of EOP by limiting the downtime in the evaluation runs. This is achieved by two improvements:

- Problem evaluation is rewritten as an asynchronous implementation. It leverages a queue-based evaluation, where the problem continuously takes evaluations from the queue. The newest proposals from the optimization receive a higher priority compared to previous ones through a LIFO queue.
- The sequential surrogate tuning, fitting, and optimization are separated into a trigger-based process where each stage has its own prerequisites to start a new iteration. This allows an independent granularity where the procedures are only executed when strictly necessary.

In this paper, we first discuss related existing optimization frameworks in Python that are applicable for the optimization and evaluation of expensive problems (in Section 2). In Section 3, the deficiencies of current methodologies are analyzed through time complexity. From this analysis, we propose and discuss the novel Event-driven Surrogate-Assisted Framework (EDSAF). A comparison of EDSAF with traditional surrogate-based methods is provided in Section 4 with a conclusion in Section 5.

2 Related Work

In this section, a brief overview of implementations for multi-objective surrogate-assisted optimization will be discussed by reviewing the general direction and goal of the authors, the support for multi-objective optimization, parallel execution, surrogate-assisted implementations, and event-driven optimization for efficient problem utilization. Parallel Global Multiobjective Optimizer (PyGMO) [5] is a Python library designed for massively parallel optimization. It aims to simplify the deployment of optimization algorithms and problems in massively parallel environments by providing a unified interface. The package includes asynchronous implementations of a wide range of optimization algorithms but does not provide surrogate-assisted optimization. jMetalPy [3], a Python port of the popular jMetal Java library, is designed for multi-objective optimization using metaheuristic techniques. It provides an environment for solving multi-objective optimization problems, focusing not only on traditional metaheuristics but also on techniques supporting preference articulation and dynamic problems. Similar to PyGMO, surrogate-assisted optimization is not provided for jMetalPy. Distributed Evolutionary Algorithms in Python (DEAP) [17] is a framework for evolutionary computation that allows for rapid prototyping and testing of ideas. DEAP focuses on genetic algorithms with added capabilities for multi-objective optimization. Parallelization is supported through multiprocessing, which enables more efficient evaluations. DEAP, like pyGMO and jMetalPy, does not yet support a framework for implementing surrogate-assisted optimization out of the box. This requires custom integrations of surrogate modeling into the frameworks by the user [15, 24, 36, 40, 39].

PySOT [14] is an asynchronous parallel optimization toolbox for EOP built upon POAP, an event-driven framework for constructing and combining asynchronous optimization strategies. It is designed for the global optimization of expensive functions where concurrent function evaluations are beneficial. The package is restricted to single-objective, constrained optimization, not allowing usage for common multi-objective real-world problems where expensive evaluations are much more frequent. pySOT provides an implementation of four surrogate models, namely radial basis function (RBF), Gaussian Process Regression (GPR), Multivariate Adaptive Regression Splines (MARS), and a polynomial regression without a possibility for surrogate selection or hyperparameter tuning during the optimization runs.

These frameworks have been widely used in the field of optimization, particularly in scenarios where the objective functions are computationally expensive and the number of evaluations is limited. Most offer synchronous and asynchronous parallelism and support both continuous and integer variables. Some offer surrogate support to further improve the efficiency of the optimization. These packages work from the same principle, namely, a fixed population size is set for each generation. The optimizer waits for all these evaluations, no matter how long it takes.

Some work has been published on dynamic hyperparameters during optimization runs. For instance, Tran et al. [35] introduce a MOEA with adaptive population size, self-adaptive crossover, and self-adaptive mutation for automating the process of adjusting parameter values. The proposed method tries to keep the population size as small as possible, allowing for a large number of generations until this population size is not large enough to commensurate the difficulty of the problem. Li et al. [24] produce an adaptive surrogate-assisted single-objective DE implementation. This implementation adapts the population size based on the number of feasible solutions found, allowing an improved convergence for high-dimensional problems. In work from Elsayed et al. [13], a surrogate-assisted differential evolution was proposed called DE-DPS. The motivation behind DE-DPS is to find the most appropriate parameters for the mutation, crossover, and population size during the evolution process to

[1] The source code is publicly available through KU Leuven Gitlab repository and documentation

improve the offspring generation. The parameters with the best offspring are kept for further generations. The works of these authors optimize the population size setting based on the quality of the generated offspring but do not consider the evaluation duration or cost for EOP.

3 Methodology

The speed of surrogate-assisted optimization highly depends on the size of the historical dataset, called the archive, the used architecture for tuning, and the used surrogate model(s).

3.1 *Time complexity of optimization algorithms*

The relative time usage of each stage can be described through a time complexity analysis to compare each stage to the full procedure of the computational requirements of an event-driven surrogate-assisted optimization. The time complexity indicates the amount of time it takes to run an algorithm. This is generally expressed as a function of the size of the population N. In addition to the population size, three other parameters, problem dimensionality D, the number of epochs/generations G, and archive size A, are included to represent the influences of said parameters.

We do not take into account other influencing variables, such as model architecture. These certainly matter to the overall timing of fitting and inference runs of the surrogate model. The analysis through time complexity is executed in three steps:

1. Divide the full procedure into different phases.
2. Identify the most basic operations of each phase. These basic operations are the operations that contribute most to the total runtime.
3. Determine the number of times the basic operation is run, depending on the size of the input.

This analysis is applied to both (multi-objective) population-based optimization algorithms and their surrogate-assisted equivalents.

3.1.1 *Population-based algorithms*

Imagine an EOP with a fixed evaluation time. Population-based algorithms (PBA) can be represented as a series of iterations between the EOP and the algorithm. Both the evaluations of the EOP and the algorithm search take a finite amount of time to process. This is represented in the first timing diagram in Figure 1 by the non-zero width of each block. The length of the EOP evaluation depends on the number of proposals of the PBA and the loop time of an evaluation. Simple population-based metaheuristics require minimal computation to determine a resulting answer in the form of a set of proposals. The steps for time complexity analysis can be applied to genetic algorithms (GA). These approaches do not use historical information from previous iterations to generate proposals. Multiobjective evolutionary algorithms (MOEAs) handle a diverse set of solutions to find multiple Pareto-optimal solutions in one single simulation run. The non-dominated sorting genetic algorithm (NSGA-II) was one of the first such MOEAs. The time complexity of NSGA-II is given as:

$$O(G \cdot M \cdot N^2) \tag{1}$$

with G is the number of generations, N is the population size, and M is the number of objectives [11]. The same steps can be applied to other MOEAs, such as the Strength Pareto Evolutionary Algorithm 2 (SPEA2) from Zitzler et al. [42] and Knowles and Corne's Pareto-archived PAES [22], summarized in Table 1. The time complexity of MOEA, independent of the fitness evaluation, depends on the complexity of the problems in terms of the number of objectives M, the number of individuals in each generation N, and the total amount of generations G. The total amount of historical evaluations in the archive has no influence.

3.1.2 *Surrogate-assisted MOEA*

In contrast to non-surrogate optimization, surrogate-assisted optimization employs data-driven models to approximate the EOP. Surrogate-assisted optimization and time efficiency do not go hand in hand. The surrogate tuning and fitting steps require significantly increased computational needs and execution time relative to the optimization algorithms. A surrogate-assisted evolutionary algorithm (SAEA) can be simplified in the following steps:

1. **Archive**: A dataset of evaluations is gathered in the form of an archive of historically evaluated proposals and their corresponding objective values.
2. **Surrogate selection and hyperparameter tuning**: A range of possible surrogate models is evaluated and compared through hyperparameter tuning. This step is repeated to account for changes in model fit due to data drift caused by the optimization search itself and increasing archive size. The time complexity of this step depends on the number of hyperparameters used H, the granularity of the search space S_h, and the complexity of the underlying model $M(N)$. The total number of combinations N_c for each surrogate model to explore is:

$$N_c = \sum_{i=0}^{i=H} S_h \tag{2}$$

The surrogate selection and hyperparameter tuning for N_m possible model architectures can be noted as follows:

$$O\left(\sum_{i=0}^{i=N_m} N_{c,i} \cdot M_i(N)\right) \tag{3}$$

Different types of surrogate models can be considered. A decision tree, for instance, is built by going over each possible feature to split on, finding the best possible split for that feature, and determining the goodness of the fit. For continuous data, the input features are sorted to find the best possible split. This sorting step takes $O(N\log(N))$, which dominates the runtime. This is executed for each of the D features, resulting in:

$$O(M_{DT}(N)) = O(N\log(N) \cdot D) \tag{4}$$

The time complexity for both the fitting and inference steps of common surrogate models is listed in Table 2.

The duration of the tuning stage of the surrogate is dependent on the used surrogate model, the size of the archive, and the number

Table 1: The time complexity of a selection of commonly used genetic algorithms with G, the number of generations; M, the number of Objectives; and N, the population size.

Algorithm	Time Complexity	Citations
NSGA, SPEA	$O(G \cdot M \cdot N^3)$	[10, 11, 25]
NSGAII, SPEA2, PEAS	$O(G \cdot M \cdot N^2)$	[10, 11, 25]
Variants of NSGA-II	$O(G \cdot M \cdot N \log_{M-1} N)$	[12, 16, 20, 23]

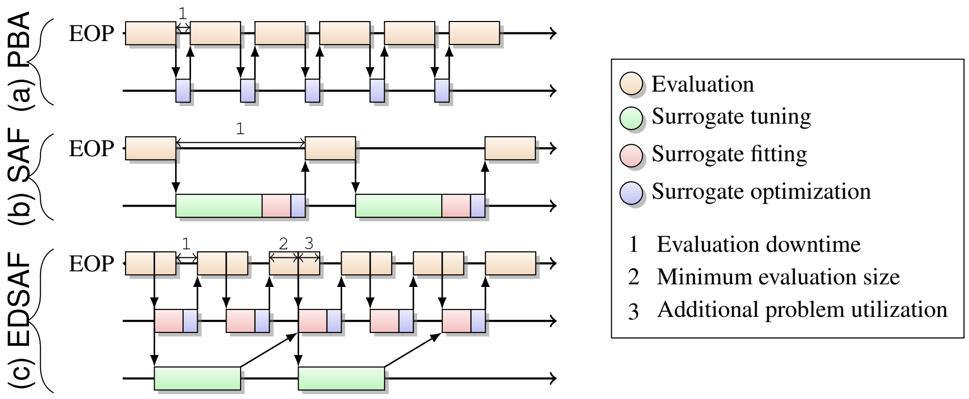


Figure 1: A representation of the timing of a population-based algorithm (PBA), a Surrogate-Assisted Framework (SAF), and the proposed Event-Driven Surrogate-Assisted Framework (EDSAF). The non-zero width of each block illustrates the time to evaluate the problem (orange), tune the surrogate (green), fit the surrogate (red), and search the surrogate or population (blue).

of hyperparameter combinations searched. A large archive of historical data would result in a significant increase in the duration of the tuning of the surrogate.

3. **Fitting of the surrogate**: The tuning of the surrogate will select an optimal hyperparameter combination to use for the current archive. This stage executes a single fitting on the archive. Similar to the tuning, the duration of the fitting stage of the surrogate is dependent on the surrogate model used and the size of the archive.
4. **Optimization of the surrogate**: Finally, after the construction of the surrogate, the optimization is applied to this model. As stated in Section 3.1.1, the optimization time-efficiency depends on G, the number of generations; N, the population size; and D, the problem dimensionality. As the surrogate replaces the EOP, we can afford a higher number of generations and population size. This results in a much faster convergence in terms of the evaluation efficiency of the EOP but is disadvantageous for time efficiency. The time complexity of this step is a combination of the inference on the surrogate and the complexity of the optimization algorithm.

Next to the used surrogate model and computational resources, the duration of the tuning, fit, and search mainly depends on the size of the archive. This means as the optimization continues, the archive will expand and in turn, increase the time between the finished evaluation and the generation of the next proposals. The analysis shows that the main aspect contributing to the time complexity of the surrogate-assisted optimization procedure are the tuning and fitting of the surrogate model. This step can have cubic growth in relation to an increase of the complete data archive, depending on the chosen model. Several approaches can lower the duration of this step:

1. The input size can be reduced by not including the full archive in the surrogate tuning and fitting. For example, the archive could be sampled based on the recency or domain representation of the data.
2. An incremental learning approach could be used for fitting the surrogate. This would allow the addition of new data to the model for each evaluation iteration without needing to fit the model to the full archive.
3. An event-driven method separates the required stages in parallel tracks. We can define a set of start prerequisites (triggers) for each stage. This allows a varying temporal granularity between the parallel stages, as some steps are allowed to take a longer duration if needed. This method allows a short response time on incoming data by not requiring all processes to finish. This is our approach for EDSAF.

Table 2: The time complexity of commonly used surrogate models for the fitting and inference stage with the size of the archive A; D, the dimensionality; K, number of trees; T, the depth of the trees.

Surrogate	Fitting	Inference	
Linear reg. (OLS)	$O(AD^2 + A^3)$	$O(D)$	[38]
GPR	$O(A^3)$	$O(A^3)$	[2]
Decision tree	$O(A\log(A) \cdot D)$	$O(\log(A))$	[28, 29]
Random forest	$O(KA\log(A) \cdot D)$	$O(K\log(A))$	[28, 29]
XGBoost	$O(KTA)$	$O(KT)$	[9]

3.2 *Event-Driven Surrogate-Assisted Framework (EDSAF)*

Figure 1 shows our proposed method EDSAF compared to the traditional structure of a Surrogate-Assisted Framework (SAF). The *fit* of the surrogate model is started when a minimum number of evaluations is reached (shown by *2*); it does not require all proposals to be evaluated. By fitting the model on a part of the proposed solution, we allow the fitting of the surrogate to commence sooner. The evaluation is still being continued, avoiding a downtime during the EOP evaluation. The evaluations (shown by *3*) that happen after the minimum of evaluations is reached are not wasted. These values still explore the problem and are used on the next fitting iteration. The optimization of the surrogate is performed sequentially after the fitting, as no benefit would be found by searching the surrogate from the previous iteration. The tuning of the surrogate is the most time-consuming stage. The back-and-forth iterations are not slowed by executing this stage in parallel to the fit and optimize processes. As the fit stage does not require the latest tuned hyperparameters before commencing, we do not wait for this stage to finish. The updated parameters are ingested to the fitting stage when available.

The algorithm can be summarized by the pseudocode shown in Algorithm 1. The EDSAF begins with an initialization (Lines 1-7); this initialization uses a chosen sampling (Line 2) to provide a set of initial proposals to the launch queue (Line 3) and waits for the complete sampling to finish (Line 4 to 6). On the first evaluations, an initial full hyperparameter tuning is performed to find the first useable surrogate model (Line 7). While the optimization run has not reached the termination criteria, set here as a maximum number of evaluations, the iteration continues (Line 9). This loop starts by fitting the surrogate

Algorithm 1 EDSAF

```
Require: Queues $Q_{l,EOP}$ and $Q_{r,EOP}$ from the EOP, Optimization
    algorithm $\Phi$, Queues $Q_{l,H}$ and $Q_{r,H}$ from the Hyperparameter
    optimization $H$, Minimum Evaluations per Iteration $SEI^{(min)}$,
    Maximum Number of Solution Evaluations $SE^{(max)}$
    ▷ Initialize an archive and proposal set
 1: $A \leftarrow \emptyset; P \leftarrow \emptyset;$
    ▷ Initial sampling of the EOP
 2: $A.X \leftarrow$ doe();
 3: $Q_{l,EOP} \leftarrow A.X$
    ▷ Wait for all evaluations to finish
 4: while $Q_{l,EOP} \neq 0$ do
 5: end while
 6: $A.F, A.G \leftarrow Q_{r,EOP}$
    ▷ Initial surrogate hyperparameter tuning
 7: $Q_{l,H} \leftarrow A.X, A.F, A.G$
 8: $H_o \leftarrow Q_{r,H}$
 9: while size($A$) $< SE^{\max}$ do
    ▷ Surrogates S for each obj. and constr.
10:     $S \leftarrow H_o$.fit($A.X, A.F, A.G$)
    ▷ Generate a set of proposals P with the chosen optimization alg.
11:     $P.X \leftarrow \Phi$.optimize($S$)
    ▷ Add the proposals on EOP launch queue $Q_l$
12:     $Q_l \leftarrow P.X$
    ▷ Wait to reach the min. required eval. from the queue $Q_e$
13:     while $size(Q_l) > size(P.X) - SEI^{(min)}$ do
14:     end while
15:     $P.F, P.G \leftarrow P.F, P.G \cup Q_e$
16:     $A \leftarrow A \cup P$
    ▷ Receive the results and start a new hyperparameter opt. if the
    previous run is finished
17:     if $size(Q_{r,H}) > 0$ then
18:         $H_o \leftarrow Q_{r,H}$
19:         $Q_{l,H} \leftarrow A.X, A.F, A.G$
20:     end if
21: end while
```

Algorithm 2 EDSAF: Evaluation

```
Require: EOP
    ▷ Initialize the evaluation queues
 1: $Q_l \leftarrow \emptyset; Q_r \leftarrow \emptyset;$
 2: for $q_l \in Q_l$ do
 3:     $q_r \leftarrow EOP$.evaluate($q_l$)
 4:     $Q_r \leftarrow Q_r \cup q_r$
 5: end for
```

model on the updated archive of all past evaluations (Line 10). The fitted surrogate is searched by the given optimization algorithm, resulting in a set of proposals (Line 11). This set of proposals is added to the launch queue to be evaluated on the EOP (Line 12). Here, a minimum number of finished evaluations SEI^{min} is required (Lines 13-14) before the algorithm receives the evaluations (Line 15) and adds the new points to the archive (Line 16). If the previous hyperparameter tuning run has finished (Line 17), the new hyperparameter set is received (Line 18), and a new run is started (Line 19). By minimizing the duration of the loop (Lines 9-21), a more frequent update of the surrogate would be possible on new data.

The evaluation of the EOP is guided through a launch and receive queue ($Q_{l,EOP}$ and $Q_{r,EOP}$) where the EOP will sequentially evaluate all the proposals given. This process is shown as a parallel process in Algorithm 2. The hyperparameter tuning of the surrogate is similarly parallel to the main process with two queues $Q_{l,H}$ and $Q_{r,H}$.

3.3 Design of experiments

To evaluate the performance, the proposed EDSAF algorithm is compared to a baseline surrogate-assisted implementation by using GPSAF on a series of unconstrained, multi-objective optimization problems. The experiments are repeated in multiple runs (n=40) on multiple independent machines with fixed semi-random seeds to take into account the non-deterministic nature of genetic-based algorithms and computational timing. The used hardware are five identical 8-thread machines (Intel(R) Core(TM) i7-2600 CPU @ 3.40GH) with 16 Gb memory. Each problem evaluation had a fixed duration of 0.2 seconds. A Gaussian Process Regression (GPR) with a Radial Basis Function (RBF), initialized with a broad hyperparameter range (normalization, regressions, kernel), is provided as a surrogate.

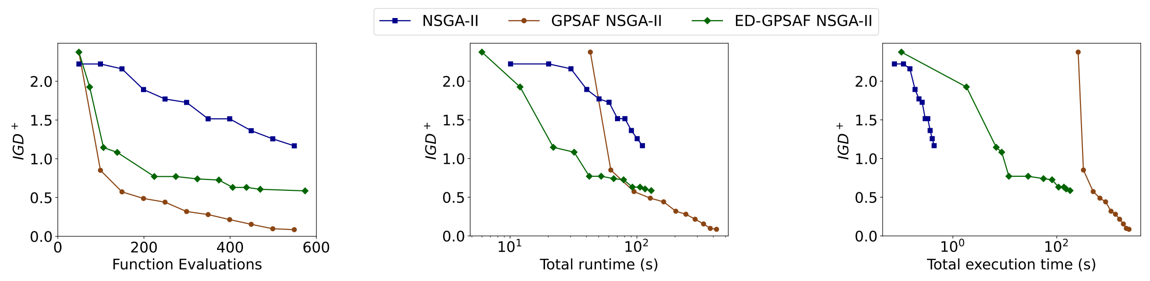
The proposed method is compared to the baselines through a rank-based comparison adapted from Blank et al. [7]. This allows us to compare the performance of the event-driven proposal across a large test suite. The rank-based comparison procedure is as follows:

1. **Statistical Domination**: After gathering data from multiple runs for each test problem and algorithm ($A \in \Omega$), we perform a pairwise comparison of performance indicators (PI) between all algorithms using the Wilcoxon Rank Sum Test ($\alpha = 0.05$). The null hypothesis H_0 is that no significant difference exists, whereas the alternative hypothesis is that the performance indicator of the first algorithm ($PI(\mathcal{A})$) is less than that of the second one ($PI(\mathcal{B})$), as follows:
$$\phi(\mathcal{A}, \mathcal{B}) = \text{RANKSUM}\left(\text{PI}(\mathcal{B}), \text{PI}(\mathcal{A}), \text{alt} =' \text{less}\,'\right) \quad (5)$$
where the function $\phi(A, B)$ returns zero if the null hypothesis is accepted or one if it is rejected.
2. **Number of Dominations**: The performance $P(\mathcal{A})$ of algorithm A is then calculated by the number of methods that are dominating it:
$$P(\mathcal{A}) = \sum_{\substack{\mathcal{B}\in\Omega \\ \mathcal{A}\neq\mathcal{B}}} \phi(\mathcal{B}, \mathcal{A}) \quad (6)$$
This results in a domination number $P(\mathcal{A})$ for each method, which is zero if no other algorithm outperforms it.
3. **Ranking**: Finally, we sort the methods by their $PI(\mathcal{A})$. This might result in a partial ordering with multiple algorithms having the same $PI(\mathcal{A})$ values. To maintain the overall sum of ranks equal, we assign their average ranks in case of ties. Averaging the ranks for ties penalizes an optimization method for being dominated by the same number of algorithms as others and keeps the rank sum for each problem the same.

3.4 Performance indicators

The proposed algorithm is evaluated on three performance indicators (PI): Convergence to the ideal solution, runtime efficiency, and computational requirements.

1. The *convergence* compares the ability of the algorithm to approach the ideal solution in function of the number of evaluations. IGD^{+}[19] is used as a PI for multi-objective problems if the optimum is known; otherwise, Hypervolume [43] is used.



(a) (b) (c)

Figure 2: Comparison of the IGD^+ of a single optimization run for the three implementations (Baseline, GPSAF, EDSAF) of the NSGA-II. The IGD^+ metric is compared against the number of evaluations (2a), the runtime of the algorithm (2b), and the execution time (2c) on ZDT1.

2. The *Runtime* signifies the total feedback time of the algorithm. This denotes the total time delta between starting the first evaluations and the final result of the optimizer. The value includes both the time needed to evaluate the problem and apply the optimization procedure to the evaluated points. The PI for the runtime is defined as the reached convergence after a fixed runtime. This PI considers both the efficiency of the optimizer and the time needed to find the results.
3. The last PI evaluated is the *total execution time* to compare the computational requirements of the algorithm. The compute usage is tracked for each iteration in terms of total CPU time (the time the CPU spends executing instructions). For multiple cores, CPU time is measured as the sum of the CPU time for all cores.

4 Results and discussion

The experimental evaluation investigates the influence of EDSAF on the PIs of existing optimization algorithms on unconstrained multi-objective problems using the six problems of the ZDT test suite [41]. A high number of evaluations is specifically required when the problem exists from multiple conflicting objectives. The following number of variables were used for each problem respectively: 30, 30, 30, 10, 80, and 10. Three popular optimization algorithms, NSGA-II [11], SPEA2 [42], and SMS-EMOA [4] are used as baseline algorithms, where we compare the convergence of the event-driven implementation of GPSAF with the standard GPSAF and the non-surrogate-based baseline.

Figure 2 compares a single optimization run from the three variants (baseline, GPSAF, and EDSAF) of NSGA-II applied to the ZDT1 problem. The first graph (2a) shows the convergence of the optimization through the IGD^+ metric. For NSGA-II and GPSAF-NSGA-II, a regular sampling can be seen with a population size of 50 evaluations for each generation. This is not the case for EDSAF, as the population size of this algorithm depends on the speed of the said algorithm. A slower optimization run will result in more proposals that can be evaluated for the next generation. In this specific run, GPSAF has a superior convergence while EDSAF trails behind. NSGA-II has a very slow convergence in relation to the number of evaluations. Figure 2b plots the convergence in relation to the runtime of the optimization run. In this instance, EDSAF does not wait for the initial sampling to be completed, resulting in a faster proposal from the first generation. This proposal is, as expected, not very good. NSGA-II waits for the first sampling to be completed, which causes a longer first iteration. Again, NSGA-II does not converge very efficiently as the runtime increases. GPSAF requires a long wait for the first iteration as it waits for all evaluations, then builds and optimizes the surrogate. Only after the surrogate optimization will a new set of proposals be created. In this run, the convergence of GPSAF does catch up to EDSAF after three generations. The execution time of each NSGA-II variant is shown in the third Figure 2c. Each variant is in a league of its own, with GPSAF requiring a considerable computation time to converge to new results.

In the first experiment, the number of evaluations is limited to 500. The convergence (Table 3a), total runtime (Table 3b), and execution time (Table 3c) after the last iteration are compared between the set of said algorithms. For the second experiment, a runtime limit of 100 seconds is set to address the efficiency of the convergence compared to the runtime. This experiment shows the true benefits of the EDSAF approach through the convergence efficiency in a limited runtime. These results are shown in Table 3d. From the results in Table 3a, we can see that the non-surrogate algorithms are outperformed in terms of convergence efficiency by the GPSAF implementations. The standard GPSAF has a superior IGD^+ metric over the event-driven version for most of the evaluated problems and baseline algorithms, except for problem ZDT4. Regarding the performance between different baselines, we see a clear ranking, with SPEA2 being the best-performing, NSGA-II in second place, and SMS-EMOA performing overall worst. For both GPSAF implementations, the baselines are comparable, but there are no real superior results for any of the three used algorithms. GPSAF-SMS-EMOA and GPSAF-NSGA-II show the best results in this experiment, with both having an average rank of 2.17. Comparing the total runtime after 500 evaluations in Table 3b, the baseline optimizers are clearly the fastest methods with no distinction in the subgroup. For the surrogate-assisted framework, the event-driven implementation is evidently faster than the standard GPSAF implementation, with the exception of EDSAF-NSGA-II for problem ZDT1. No immediate reason can be given for this discrepancy. In some discrete cases, such as ZDT4 and ZDT6, we even see no significant increase in the runtime of EDSAF compared to the baselines. For the execution time, the total CPU usage of the separate steps in the optimization is aggregated and compared. Similar to the runtime, the baseline methods are the superior method for this metric. These methods are only single-threaded processes, resulting in a very low usage of the available computing power. The event-driven methods lag behind as they spread the load from the surrogate tuning and fitting steps to additional CPU cores as parallel processes. The GPSAF methods are the worst performing, with an average ranking of 7.83, being clearly the worst group in terms of total compute usage. This is caused by tuning and fitting triggers being based on a fixed number of evaluations per iteration, resulting in a full run of all steps for each iteration.

Finally, we want to compare the convergence efficiency in a limited-time situation with the second experiment. Here, we allow a limited number optimization runtime of 100 seconds. The best IGD^+ metric within this time limit is compared between the same nine algorithm frameworks. In this specific setup, the advantage of

Table 3: Comparison of four metrics for NSGA-II, SPEA2, and SMSEMOA with their GPSAF and EDSAF variants on unconstrained multi-objective problems. The rank of the overall best-performing algorithm is shown in bold.

(a) IGD after 500 evaluations

Problem	**Baseline**			**GPSAF**			**EDSAF**		
	NSGA-II	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**
ZDT1	8.0	7.0	9.0	**2.0**	**2.0**	**2.0**	6.0	5.0	4.0
ZDT2	8.0	7.0	9.0	**2.0**	**2.0**	**2.0**	4.5	4.5	6.0
ZDT3	8.0	7.0	9.0	**2.0**	**2.0**	**2.0**	5.5	5.5	4.0
ZDT4	8.0	7.0	9.0	2.0	4.0	4.0	**1.0**	6.0	4.0
ZDT5	8.0	7.0	9.0	3.0	2.0	**1.0**	4.5	4.5	6.0
ZDT6	8.0	7.0	9.0	**2.0**	**2.0**	**2.0**	4.0	5.5	5.5
Total	8.00	7.00	9.00	**2.17**	2.33	**2.17**	4.25	5.17	4.92

(b) Runtime after 500 evaluations

Problem	**Baseline**			**GPSAF**			**EDSAF**		
	NSGA-II	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**
ZDT1	**2.0**	**2.0**	**2.0**	7.0	7.0	7.0	9.0	4.5	4.5
ZDT2	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT3	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT4	**3.5**	**3.5**	**3.5**	8.0	8.0	8.0	**3.5**	**3.5**	**3.5**
ZDT5	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT6	**3.5**	**3.5**	**3.5**	8.0	8.0	8.0	**3.5**	**3.5**	**3.5**
Total	**2.50**	**2.50**	**2.50**	7.83	7.83	7.83	5.17	4.42	4.42

(c) Execution time after 500 evaluations

Problem	**Baseline**			**GPSAF**			**EDSAF**		
	NSGA-II	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**
ZDT1	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	6.0	4.5	4.5
ZDT2	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT3	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT4	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT5	**1.5**	3.0	**1.5**	8.0	8.0	8.0	5.0	5.0	5.0
ZDT6	**2.0**	**2.0**	**2.0**	8.0	8.0	8.0	5.0	5.0	5.0
Total	**1.92**	2.17	**1.92**	8.00	8.00	8.00	5.17	4.92	4.92

(d) IGD after a runtime = 100s

Problem	**Baseline**			**GPSAF**			**EDSAF**		
	NSGA-II	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**	**NSGA-II**	**SPEA2**	**SMSEMOA**
ZDT1	8.0	7.0	9.0	5.0	5.0	5.0	**1.0**	2.5	2.5
ZDT2	8.0	7.0	9.0	5.0	5.0	5.0	2.5	**1.0**	2.5
ZDT3	4.5	4.5	6.0	8.0	8.0	8.0	**1.5**	**1.5**	3.0
ZDT4	7.0	4.0	7.0	7.0	7.0	7.0	**2.0**	**2.0**	**2.0**
ZDT5	7.5	7.5	9.0	**2.0**	**2.0**	**2.0**	5.0	5.0	5.0
ZDT6	8.5	7.0	8.5	5.0	5.0	5.0	**2.0**	**2.0**	**2.0**
Total	7.25	6.17	8.08	5.33	5.33	5.33	**2.33**	**2.33**	2.83

EDSAF is shown, with a significant improvement for all problems except ZDT5. The other methods trail behind EDSAF due to two different causes. The baseline methods lack the needed convergence to reach a good result because of the limited number of evaluations available due to the evaluation duration of the problems. This is in contrast to the GPSAF, where an excellent convergence is possible with a limited number of evaluations. Still, the optimization step requires relatively more time to reach new proposals to evaluate. This causes, in turn, a downtime of the evaluation process, where unused time is wasted while waiting for proposals. When comparing the event-driven to the standard surrogate-assisted, EDSAF maintains a close gap in terms of convergence in relation to the number of evaluations. The proposed method can still be seen as a valid implementation for ESE optimization. The main advantage of EDSAF is the clear reduction in runtime and computational needs, reducing the overall need for larger computing power even when working with larger archive datasets, while maintaining sufficient convergence. The EDSAF methods marginally lag behind the runtime performance of the baseline optimization implementations. This makes them very suitable for real-time optimization of complex, possibly multi-objective problems without needless downtime of the EOP.

5 Conclusion

In this work, we proposed an event-driven approach for surrogate-assisted optimation of expensive problems. Expensive, in this context, is defined with respect to two aspects. The traditional definition of expensive optimization problems as a cost of a single evaluation is taken into account while also accounting for the runtime efficiency of the evaluated problem. The objective is to arrive at a close-to-optimal solution with a minimal amount of evaluations while keeping the computational needs in check. This allows for a fast iteration time of the surrogate-assisted optimization, which is ideal for real-time optimization of complex, expensive problems without an increasing runtime duration due to the expanding data archive.

EDSAF has been applied to multiple, well-known population-based algorithms on complex multi-objective benchmark functions. While the proposed approach does not reach the convergence excellence of traditional, high compute-intensive surrogate-assisted methods and lags behind the runtime of a classic optimization algorithm, such as NSGA-II, it takes the best of both worlds to reach a good balance. This allows a superior convergence of the problem in relation to the runtime of the optimization, allowing better utilization of the limited evaluation availability, such as in simulation allocations or testing periods of real-world processes.

Acknowledgements

Conflict of interests The authors have no competing interests to declare that are relevant to the content of this article.

Funding This research is supported by Internal Funds KU Leuven (STG/21/057).

Availability of data and materials The resulting metrics and used parameters for the evaluated benchmarks are available on the KU Leuven Research Data Repository, 10.48804/XVLHH2.

References

[1] J. D. Anderson, J. Degroote, G. Degrez, E. Dick, R. Grundmann, and J. Vierendeels. *Computational fluid dynamics: an introduction*. Springer, 2009. ISBN 978-3-540-85055-7.

[2] M. Belyaev, E. Burnaev, and Y. Kapushev. Exact Inference for Gaussian Process Regression in case of Big Data with the Cartesian Product Structure, July 2014. URL http://arxiv.org/abs/1403.6573. arXiv:1403.6573 [math, stat].

[3] A. Benítez-Hidalgo, A. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, 51:100598, Oct. 2019. doi: 10.1016/j.swevo.2019.100598.

[4] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, Sept. 2007. ISSN 0377-2217. doi: 10.1016/j.ejor.2006.08.008. URL https://www.sciencedirect.com/science/article/pii/S0377221706005443.

[5] F. Biscani and D. Izzo. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, 5(53):2338, Sept. 2020. ISSN 2475-9066. doi: 10.21105/joss.02338. URL https://joss.theoj.org/papers/10.21105/joss.02338.

[6] J. Blank and K. Deb. pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020.

[7] J. Blank and K. Deb. GPSAF: A Generalized Probabilistic Surrogate-Assisted Framework for Constrained Single- and Multi-objective Optimization, Apr. 2022. URL http://arxiv.org/abs/2204.04054. arXiv:2204.04054 [cs, math].

[8] J. Blank and K. Deb. pysamoo: Surrogate-Assisted Multi-Objective Optimization in Python, Apr. 2022. URL http://arxiv.org/abs/2204.05855. arXiv:2204.05855 [cs].

[9] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, Aug. 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

[10] D. M. Curry and C. H. Dagli. Computational Complexity Measures for Many-objective Optimization Problems. *Procedia Computer Science*, 36:185–191, Jan. 2014. ISSN 1877-0509. doi: 10.1016/j.procs.2014.09.077. URL https://www.sciencedirect.com/science/article/pii/S187705091401326X.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr. 2002. ISSN 1941-0026. doi: 10.1109/4235.996017. URL https://ieeexplore.ieee.org/document/996017. Conference Name: IEEE Transactions on Evolutionary Computation.

[12] R. G. L. D'Souza, K. C. Sekaran, and A. Kandasamy. Improved NSGA-II Based on a Novel Ranking Scheme, Feb. 2010. URL http://arxiv.org/abs/1002.4005. arXiv:1002.4005 [cs].

[13] S. M. Elsayed, T. Ray, and R. A. Sarker. A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1062–1068, July 2014. doi: 10.1109/CEC.2014.6900351. URL https://ieeexplore.ieee.org/document/6900351. ISSN: 1941-0026.

[14] D. Eriksson, D. Bindel, and C. A. Shoemaker. pySOT and POAP: An event-driven asynchronous framework for surrogate optimization, July 2019. URL http://arxiv.org/abs/1908.00420. arXiv:1908.00420 [cs, math, stat].

[15] R. Espinosa, F. Jiménez, and J. Palma. Multi-surrogate assisted multi-objective evolutionary algorithms for feature selection in regression and classification problems with time series data. *Information Sciences*, 622:1064–1091, Apr. 2023. ISSN 0020-0255. doi: 10.1016/j.ins.2022.12.004. URL https://www.sciencedirect.com/science/article/pii/S0020025522014979.

[16] H. Fang, Q. Wang, Y.-C. Tu, and M. F. Horstemeyer. An Efficient Non-dominated Sorting Method for Evolutionary Algorithms. *Evolutionary Computation*, 16(3):355–384, Sept. 2008. ISSN 1063-6560. doi: 10.1162/evco.2008.16.3.355. URL https://doi.org/10.1162/evco.2008.16.3.355.

[17] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012. ISSN 1533-7928. URL http://jmlr.org/papers/v13/fortin12a.html.

[18] A. Hasni, R. Taibi, B. Draoui, and T. Boulard. Optimization of Greenhouse Climate Model Parameters Using Particle Swarm Optimization and Genetic Algorithms. *Energy Procedia*, 6:371–380, Jan. 2011. ISSN 1876-6102. doi: 10.1016/j.egypro.2011.05.043. URL https://www.sciencedirect.com/science/article/pii/S1876610211014548.

[19] H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima. Modified Distance Calculation in Generational Distance and Inverted Generational Distance. In A. Gaspar-Cunha, C. Henggeler Antunes, and C. C. Coello, editors, *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, pages 110–125, Cham, 2015. Springer International Publishing. ISBN 978-3-319-15892-1. doi: 10.1007/978-3-319-15892-1_8.

[20] M. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, 2003. doi: 10.1109/TEVC.2003.817234.

[21] D. R. Jones. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization*, 21(4):345–383, Dec. 2001. ISSN 1573-2916. doi: 10.1023/A:1012771025575. URL https://doi.org/10.1023/A:1012771025575.

[22] J. Knowles and D. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 98–105 Vol. 1, July 1999. doi: 10.1109/CEC.1999.781913. URL https://ieeexplore.ieee.org/document/781913.

[23] P. Kumar K., S. Sharath, G. R. D'Souza, and K. C. Sekaran. Memetic NSGA - a multi-objective genetic algorithm for classification of microarray data. In *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, pages 75–80, Dec. 2007. doi: 10.1109/ADCOM.2007.114. URL https://ieeexplore.ieee.org/document/4425954.

[24] E. Li. An adaptive surrogate assisted differential evolutionary algorithm for high dimensional constrained problems. *Applied Soft Computing*, 85:105752, 2019. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2019.105752. URL https://www.sciencedirect.com/science/article/pii/S1568494619305332.

[25] M. Liu and W. Zeng. Reducing the run-time complexity of NSGA-II for bi-objective optimization problem. In *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 2, pages 546–549, 2010. doi: 10.1109/ICICISYS.2010.5658387.

[26] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf. NSGA-Net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 419–427, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6111-8. doi: 10.1145/3321707.3321729. URL https://doi.org/10.1145/3321707.3321729. event-place: Prague, Czech Republic.

[27] S. Lucidi, M. Maurici, L. Paulon, F. Rinaldi, and M. Roma. A Simulation-Based Multiobjective Optimization Approach for Health Care Service Management. *IEEE Transactions on Automation Science and Engineering*, 13:1–12, Oct. 2016. doi: 10.1109/TASE.2016.2574950.

[28] M. J. Moshkov. Time Complexity of Decision Trees. In J. F. Peters and A. Skowron, editors, *Transactions on Rough Sets III*, pages 244–459, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31850-7.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] R. S. Rai and V. Bajpai. Optimization in Manufacturing Systems Using Evolutionary Techniques. In K. Gupta and M. K. Gupta, editors, *Optimization of Manufacturing Processes*, pages 201–229. Springer International Publishing, Cham, 2020. ISBN 978-3-030-19638-7. doi: 10.1007/978-3-030-19638-7_9. URL https://doi.org/10.1007/978-3-030-19638-7_9.

[31] P. C. Roy, A. Guber, M. Abouali, A. P. Nejadhashemi, K. Deb, and A. J. M. Smucker. Crop yield simulation optimization using precision irrigation and subsurface water retention technology. *Environmental Modelling & Software*, 119:433–444, Sept. 2019. ISSN 1364-8152. doi: 10.1016/j.envsoft.2019.07.006. URL https://www.sciencedirect.com/science/article/pii/S1364815218305644.

[32] G. Serhat and I. Basdogan. Multi-objective optimization of composite plates using lamination parameters. *Materials & Design*, 180:107904, Oct. 2019. ISSN 0264-1275. doi: 10.1016/j.matdes.2019.107904. URL https://www.sciencedirect.com/science/article/pii/S0264127519303429.

[33] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://papers.nips.cc/paper_files/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

[34] B. Szabó and I. Babuška. *Finite Element Analysis*. John Wiley & Sons, Sept. 1991. ISBN 978-0-471-50273-9.

[35] K. D. Tran. An improved Non-dominated Sorting Genetic Algorithm-II (ANSGA-II) with adaptable parameters. *International Journal of Intelligent Systems Technologies and Applications*, 7(4):347, 2009. ISSN 1740-8865, 1740-8873. doi: 10.1504/IJISTA.2009.028052. URL http://www.inderscience.com/link.php?id=28052.

[36] M. Yagoubi and H. Bederina. Surrogate-Assisted NSGA-II Algorithm for Expensive Multiobjective Optimization. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, pages 431–434, New York, NY, USA, July 2023. Association for Computing Machinery. ISBN 9798400701207. doi: 10.1145/3583133.3590746. URL https://doi.org/10.1145/3583133.3590746.

[37] H. Yin, H. Fang, G. Wen, Q. Wang, and Y. Xiao. An adaptive RBF-based multi-objective optimization method for crashworthiness design of functionally graded multi-cell tube. *Structural and Multidisciplinary Optimization*, 53(1):129–144, Jan. 2016. ISSN 1615-1488. doi: 10.1007/s00158-015-1313-1. URL https://doi.org/10.1007/s00158-015-1313-1.

[38] H. Younes, M. Alameh, A. Ibrahim, M. Rizk, and M. Valle. Efficient Algorithms for Embedded Tactile Data Processing. In *Electronic Skin*. River Publishers, 2020. ISBN 978-1-00-333806-2. Num Pages: 26.

[39] L. Zhao, Y. Hu, B. Wang, X. Jiang, C. Liu, and C. Zheng. A surrogate-assisted evolutionary algorithm based on multi-population clustering and prediction for solving computationally expensive dynamic optimization problems. *Expert Systems with Applications*, 223:119815, Aug. 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.119815. URL https://www.sciencedirect.com/science/article/pii/S0957417423003160.

[40] M. Zhao, K. Zhang, G. Chen, X. Zhao, C. Yao, H. Sun, Z. Huang, and J. Yao. A surrogate-assisted multi-objective evolutionary algorithm with dimension-reduction for production optimization. *Journal of Petroleum Science and Engineering*, 192:107192, Sept. 2020. ISSN 0920-4105. doi: 10.1016/j.petrol.2020.107192. URL https://www.sciencedirect.com/science/article/pii/S0920410520302783.

[41] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, June 2000. ISSN 1063-6560, 1530-9304. doi: 10.1162/106365600568202. URL https://direct.mit.edu/evco/article/8/2/173-195/868.

[42] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Report, ETH Zurich, Computer Engineering and Networks Laboratory, Zurich, May 2001. Publication Title: TIK Report Volume: 103.

[43] E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: on the design of pareto-compliant indicators via weighted integration. In *Proceedings of the 4th international conference on Evolutionary multi-criterion optimization*, EMO'07, pages 862–876, Berlin, Heidelberg, Mar. 2007. Springer-Verlag. ISBN 978-3-540-70927-5.