Optimization of Time-Consuming Objective Functions
Derivative-Free Approaches and their Application in Architecture

[Extended Abstract]

Catarina Belém
catarina.belem@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal
May 2019

Abstract

The building sector presents one of the largest economic and environmental footprints. Building performance optimization can minimize this impact by combining (1) algorithmic approaches, to generate multiple building design variants, (2) simulation tools, to evaluate building's performance regarding distinct aspects, and (3) optimization algorithms, to seek more efficient building designs. Unfortunately, despite the existence of different optimization algorithms, their application to architectural optimization problems is not well-studied and this often drives architects towards the simplest available algorithm. However, this rarely is the most efficient option for addressing a specific problem, in particular, due to the time-intensive simulations required to evaluate building designs. As a result, poor algorithm selection might lead to unacceptable optimization times and less efficient designs.

This dissertation addresses optimization algorithms specifically tailored for handling simulation-based optimization problems. In particular, we develop and assess an optimization framework in the context of three architectural case studies involving single- and multi-objective optimization of simulation-based building performance. Obtained results reveal that solutions' quality and the time spent in optimization are strongly dependent on the algorithm's choice. Thus, different algorithms should be tested in order to determine the one that better fits an optimization problem. Finally, this dissertation presents the benefits of optimization to reduce the impact of the building sector and motivates its introduction as an indispensable phase of the architectural design process.

Keywords: Derivative-Free Optimization; Single-Objective Optimization; Multi-Objective Optimization; Algorithmic Optimization; Algorithmic Design

1. Introduction

Since the creation of the digital computer, architecture adopted computer science tools and techniques to change its own practices. This lead to the dissemination of digital modelling tools, which simplified the design of highly complex buildings. However, these days, it is not enough to have a well-designed building, it is also necessary to ensure a good performance at various level (e.g., thermal comfort, lighting, structural), among others. This motivated the appearance of Performance-Based Design (PBD) - an approach which seeks for more efficient design solutions by considering the design's performance, possibly measured by computational tools.

These tools, known as simulation-based analysis tools, take a specialized model of a design (designated analytical model) and estimate its performance regarding the intended criteria. Using these tools, architects can validate whether their design satisfies the efficiency requirements and, ultimately, optimize the design by iteratively remodeling it in order to obtain variations, assessing their performance, and selecting the better ones, a process known as Building Performance Optimization (BPO).

However, this process has several problems: (1) the manual generation of the multiple design variants required by optimization processes comprises a tiresome and difficult task; (2) hand-made analytical models might be a more faithful representation of the original model but they require a considerable amount of effort to create; (3) despite the existence of tools that attempt to convert a 3D model into its corresponding analytical models, this conversion is frequently fragile and can cause loss of information or errors; (4) ideally, the analysis' results would afterwards be used to guide changes in the original design, but this requires additional time and effort to implement, as also does redoing the analysis to confirm the improvements. This explains why performance analysis is typically
postponed to the later stages of the design process, only to verify if the performance meets the standard requirements. Unfortunately, by following such process, it becomes difficult to optimize a design, which, nowadays, has become important for ensuring the design of efficient and sustainable buildings.

In order to fully support optimization processes, several changes should be implemented in the design process. First, we need to be able to quickly change a design and, almost simultaneously, produce its corresponding analytical model. Secondly, we need to automate the performance analysis and to use it as the function to optimize.

1.1. Algorithmic Design

To simplify and speed up the implementation of changes to a design, one can use parametric models. These models have parameters representing degrees of freedom in the design, which the architect is willing to manipulate in the search for a better performing design. A particularly good approach to produce parametric models is through Algorithmic Design (AD) [20]. Here, instead of directly creating a 3D model in a digital modeling tool, the architect creates a program that can be executed with different values of its parameters to produce the corresponding 3D models. This considerably improves the implementation of design changes, allowing a broader exploration of the design space.

Even though AD enables the automatic generation of multiple design solutions, the implementation of automatic optimization processes still requires their automatic evaluation.

1.2. Algorithmic Analysis

Similarly to AD, in Algorithmic Analysis (AA) [1], one uses an algorithm to automatically produce analytical models, instead of creating them by hand or relying on fragile conversion tools. The novelty is that this algorithm is the same that is used for AD, but configured differently, so as to match the requirements of the analysis tools being used. For example, for lighting analysis tools, a building might be represented by its surfaces, while for structural analysis tools, it might be represented by a graph of nodes and edges. In either case, the same algorithm is capable of generating different models for different types of analysis.

Moreover, AA is also concerned with the automation of the whole analysis process. In practice, this means that not only is the generation of the analytical model automated, but so is the setup of the analysis tool and the collection of its results.

1.3. Algorithmic Optimization

With the ability to quickly update a design, to generate the corresponding analytical model, and, finally, to automatically evaluate the design in an analytical tool, it becomes possible to implement automated optimization processes, which we name Algorithmic Optimization (AO).

AO treats the results produced by an analysis tool as the function to optimize (called objective function).

1.4. Research Goals

Optimization is rarely sought in architecture, mainly due to (1) the difficulties in defining the optimization problem, (2) the need for expertise to select and fine-tune the optimization algorithms, (3) the complexity of the existing optimization tools, and (4) the long computation time of the optimization process [2]. Moreover, the application of algorithms to architectural optimization problems is still insufficiently explored and often leads architects towards the simplest available algorithm, which rarely is the most adequate option, often resulting in unacceptable optimization times and less efficient designs.

Therefore, founded on the No Free Lunch Theorems (NFLTs), which state that no algorithm can consistently perform better than all others on all problems, the main goal of this dissertation is to identify the most efficient algorithms for different BPO problems that involve computationally heavy objective functions. Moreover, this dissertation aims to encapsulate these algorithms in an optimization framework that simplifies their application and, thus, promotes design optimization in architecture. Finally, we evaluate the proposed framework in different real architectural problems, including the optimization of lighting and structural aspects of buildings.

2. Optimization

An optimization process consists in two main parts: the model of the problem to be optimized and the optimization algorithm. The creation of the model involves the definition of variables, objective functions, and, optionally, constraints. In architectural design optimization, the variables are the parameters whose value will be modified, objective functions represent the performance criteria that we aim to optimize, and constraints are the conditions on the parameters that must be satisfied so that a solution is valid for the defined problem.

Depending on the number of objectives, optimization problems can be classified as Single-Objective Optimization (SOO) or Multi-Objective Optimization (MOO). The former focuses on the optimization of one objective, whilst the latter focuses on more complex problems that involve multiple objectives. Despite the increased complex-
ity, MOO encloses benefits, such as the visualization of the trade-offs among the different objectives and the selection of the solution that better fits the users’ needs. This can be achieved through Pareto-based optimization, an approach that finds the solutions for which it is impossible to improve an objective value without deteriorating others. These solutions are called nondominated and represent the Pareto front.

2.1. Optimization Algorithms
Optimization algorithms can be classified differently according to their properties. Considering the extent of the search, algorithms can be distinguished into local or global. Local optimization algorithms strive to find a solution for which the objective function yields a better value than for all the other solutions in its vicinity, i.e., a local optimum. These algorithms are usually highly sensitive to the starting point of the search and they tend to focus on smaller regions of the solution space. Conversely, global optimization algorithms strive to find globally optimal solutions, i.e., the best of all the local optima. To this end, these algorithms explore larger regions of the solution space, requiring a larger number of evaluations. However, this can be problematic for problems involving expensive objective functions. In these cases, one can apply a global algorithm to identify a promising region and then a local algorithm to more rapidly converge towards the optima within that region.

A second classification differentiates deterministic and stochastic algorithms. Given the same starting point and configuration, deterministic algorithms systematically apply the same sequence of steps and, as a consequence, they always return the same result. In contrast, stochastic algorithms include some form of randomness and, therefore, often yield different results for the same starting point and algorithm’s configuration.

The final distinction is between derivative-based and derivative-free algorithms, depending on their use of information about the objective functions’ derivatives (e.g., the direction and magnitude of its greatest increase).

In BPO, the objective function tends to be simulation-based and, thus, does not have an analytical form. In these cases, it is not possible to apply derivative-based optimization algorithms. While, ideally, one could apply finite-difference algorithms to approximate the derivatives, this would require several extra simulations, which becomes impractical for problems involving expensive objective functions. Instead, one must resort to derivative-free optimization algorithms, which treat the objective functions as black-boxes.

The following subsections describe three classes of derivative-free algorithms according to the classification proposed in the context of architectural design [25], which first subdivides the algorithms according to the search strategy’s determinism, namely, metaheuristics and iterative algorithms, and then partitions the latter into direct-search and model-based algorithms, based on the function explored to guide the search.

2.2. Direct-search Algorithms
Direct-search algorithms iteratively evaluate a sequence of candidate solutions, proposed by a deterministic strategy, and select the best solution obtained up to that moment [4]. They are regarded as valuable tools to address complex optimization problems, not only because most of them were proved to rely on solid mathematical principles, but also due to their good performance at initial stages of the search process [18].

The main limitations of these algorithms are their performance deterioration with an increasing number of input variables and their slow asymptotic convergence rates as they get closer to the optimal solution [12]. Despite the existence of algorithms and benchmarks comparing SOO direct-search algorithms [26, 21], only recently have these started to appear in the context of MOO [5].

Undoubtedly, one of the most relevant direct-search algorithms is the Nelder-Mead Simplex (NMS) [15], a local SOO direct-search algorithm. The NMS algorithm envelops a region of the design space using a simplex, which is a generalization of a triangle to arbitrary dimensions, which is then successively modified using geometric operations that replace the simplex’s worst vertex values.

Another interesting simplex-based algorithm is SUBPLEX [19], which attempts to overcome the NMS difficulties when addressing higher-dimensional problems by decomposing the problem in low-dimensional subspaces. The most promising ones are then explored by NMS.

Differently from the previous local algorithms, Dividing RECTangles (DIRECT) [11] is a global SOO algorithm, which recursively subdivides the design space into smaller multidimensional hyper-rectangles, each represented by a solution in their center. For each solution, the objective function is evaluated, thus yielding an estimate of the quality of each rectangle. Based on these values, DIRECT focus the search on more promising regions of the design space, further subdividing those. DIRECT-L [8] is a modification of DIRECT that minimizes the overall number of function evaluations, making it more efficient for functions with few local optima and a single global optimum.

Direct MultiSearch (DMS) [5] is a global multi-objective direct-search algorithmic framework
which combines the main ideas of directional direct-search algorithms with Pareto dominance concepts. In simple terms, DMS maintains a list of feasible nondominated solutions and their associated step sizes, from which it iteratively evaluates a few solutions along a predefined set of directions located at a distance determined by the solution's step size. The list is updated when better solutions are found. Otherwise, the step size is reduced. The process then repeats.

Overall, direct-search algorithms are not as popular as other classes of derivative-free algorithms. Nevertheless, their convergence proofs and the recent developments in the field of MOO make this class very appealing for BPO.

2.3. Metaheuristics Algorithms

Metaheuristics are algorithms that rely on randomization, and biological or physical analogies to locate good solutions in complex design spaces [9]. Their non-deterministic and inexact nature confer them the ability to handle complex and irregular objective functions.

Metaheuristics can be good optimization algorithms when provided with sufficient amounts of time to do the necessary objective function evaluations [4]. Unfortunately, this is not possible in architecture, where each evaluation can be a time-consuming simulation. However, because of their stochastic nature, limiting the number of evaluations has repercussions on the convergence guarantees [10].

Depending on the metaphors adopted by each algorithm, metaheuristics can be classified in different subclasses, including, Evolutionary Algorithms (EAs), which explore natural selection and evolution concepts (e.g., reproduction, recombination, crossover, and mutation), Swarm algorithms, which explore collective intelligence through the cooperation of homogeneous agents in the environment (e.g., birds), and Physical Algorithms, which take inspiration from physical processes (e.g., the annealing process in metallurgy) [9].

The Genetic Algorithm (GA) is an EA that explores biological evolution to search for better solutions. It randomly generates an initial set of individuals representing the candidate solutions, called population, which is then evolved for a specified number of iterations (or generations). The evolution process is comprised of four main phases: (1) adaptability, where individuals of the population are assigned a suitability or fitness value; (2) mating selection, where pairs of individuals are selected for reproduction based on a probabilistic function; (3) crossover, where the variables' values of the selected individuals are recombined to produce new individuals; and (4) mutation, where new individuals are subjected to random copying errors with a certain probability. While in earlier generations, populations are usually more diverse, in final generations, their individuals are often similar to the fittest individuals, thus emulating the natural selection mechanism [9].

Along the same line, Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a GA specially tailored for MOO problems, which incorporates the ideas of population genetics and evolution, Pareto dominance, and a crowding measure to attain an approximation of the Pareto front that is both accurate and diverse [6]. In simple terms, NSGA-II maintains an archive and a population, which are iteratively evolved until a stopping criteria is met. Every iteration, the algorithm combines the archive and the population and sorts their individuals according to their nondomination ranks, a measure of the number of Pareto fronts that have to be removed until an individual becomes nondominated. Individuals belonging to the same rank are also sorted according to a density measure, based on the average distance of their two closest individuals. Afterwards, the archive is replaced with the best 50% individuals and the next offspring produced based on the recombination and mutation of the individuals in the archive.

Although less explored, Particle-Swarm Optimization (PSO) algorithms have also been shown promising results in some optimization problems, surpassing widely reputed EAs. In their simpler versions, PSO algorithms are global algorithms inspired by biological systems, such as the collective behavior of flocking birds, which interact and learn from one another to solve problems. In PSO, the intelligence is decentralized, self-organized, and distributed throughout the participating particles (or swarm), which maintain information about their velocity, current and individual best positions, and also the global best position known to the swarm. At each time step, the position and velocity of each particle are updated according to the position of the best swarm or close neighbor [9].

Overall, metaheuristics are stochastic algorithms, mostly without convergence guarantees, usually requiring several hundreds or even thousands of evaluations in order to obtain good solutions. For problems involving expensive objective functions, like most architectural optimization problems, and especially for MOO problems, these algorithms are usually not a good choice. Unfortunately, there is a predominance of these algorithms (e.g., GA and NSGA-II) among BPO practitioners [26], with very few optimization tools supporting other classes of algorithms. Nevertheless, they can be very efficient solvers when properly configured. However, the optimal set of parame-
ters is problem-specific and the same configuration applied to another problem might yield a bad performance.

2.4. Model-based Algorithms
Model-based algorithms replace the original objective functions with approximations. These approximations, called the surrogates, are generated from a limited set of known objective function values and can then be efficiently explored to determine the promising candidate solutions to evaluate next. These evaluations are then used to improve the surrogates and this process is repeated until a stopping condition is satisfied [13].

Undoubtedly, the best feature of model-based algorithms is the reduction in total optimization time. This is particularly relevant in the context of BPO, which involves time-consuming objective functions. Notwithstanding the existence of different studies involving Machine Learning (ML) techniques for the creation of surrogate models [13], including Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF), only a few have actually been applied in the context of architecture. This scenario is even more self-evident when we shift from the single- to the multi-objective optimization context.

Two relevant local model-based algorithms are Constrained Optimization BY Linear Approximation (COBYLA) and Bound Optimization BY Quadratic Approximation (BOBYQA). The former uses a simplex to iteratively generate linear approximations of the objective function, whereas the latter generates quadratic approximations instead [16, 17].

RBF-based algorithms create a global approximation of the objective function as the weighted sum of radial basis functions, whose weights are estimated based on the interpolation of previous evaluation results and are iteratively updated to improve the approximation.

When compared with metaheuristics, global model-based algorithms exhibit very good performance at initial stages of an optimization process, allowing for considerable speedups in the overall optimization.

3. Algorithmic Optimization in Architecture
The architectural design paradigm has recently adopted AD and AA to quickly (1) update a design, (2) generate the corresponding analytical model, and (3) automatically evaluate the design in an analytical tool. These mechanisms laid down the foundations for automated optimization processes, which we named AO. This prompted the development of a few optimization architectural tools (e.g., Galapagos, Opossum) implemented on top of visual AD tools (e.g., Grasshopper).

Despite promoting the practice of optimization within architecture, most of them are mainly focused on SOO, thus forcing architects to adopt simplified strategies when addressing MOO problems. Additionally, the majority of these tools only provide metaheuristics algorithms, which rarely are the best choice for most problems involving expensive objective functions [26], often incurring in large computational times. Finally, because each tool provides a narrow set of algorithms, every time architects decide to test multiple algorithms, they must spend additional time and efforts to adapt their program and configure other optimization tools.

To overcome these limitations, we propose an optimization framework containing (1) an easy-to-use and intuitive interface to define and solve optimization problems, (2) a wide variety of sampling algorithms and derivative-free optimization algorithms, which currently include 4 sampling algorithms, 5 direct-search, 17 metaheuristics, and 10 model-based algorithms, (3) a set of performance indicators to measure the quality of different algorithms, and (4) a set of visual and post-processing mechanisms to aid in the interpretation of the optimization results.

To support the AO approach, the optimization framework is combined with an AD tool to facilitate the resolution of BPO problems. To this end, architects are required to: (1) create the AD model reflecting their design intents; (2) select the performance aspects to optimize and, thus, the analysis tools to be used (e.g., thermal, structural, costs), and, finally, (3) select the parameters of the optimization process (e.g., algorithm and algorithm’s parameters).

One important aspect that results from this combination is that architects are able to opt for the most suitable optimization algorithm for a specific problem. However, since the best choice is not always known, the optimization framework also provides automated testing mechanisms to enable the sequential execution of multiple optimization algorithms for a specified amount of evaluations. This feature promotes more informed decisions towards an adequate selection of optimization algorithms [24].

The interactive visualization of the evaluated design solutions is of great importance for architects, as it allows them to explore and corroborate the optimization results during the optimization process. To this end, the framework provides visualization mechanisms that promote a better comprehension of the optimization process, allowing early error detection and the potential reduction in the overall optimization time by providing the architect with enough information to stop the optimization process sooner.
4. Case Studies

To evaluate both the proposed framework and some of the available algorithms, we applied them on a series of computationally complex BPO problems that considered the daylight, structural, and cost aspects of buildings. The following sections present a simplified analysis of two case studies.

All tests were run on a dual Intel Xeon CPU E5-2670 @ 2.60GHz with 64GB RAM. Moreover, to mitigate the performance fluctuations of stochastic algorithms and produce a statistically fair analysis, each stochastic algorithm was tested 3 times and conclusions were drawn from the average of the obtained results. Finally, we opted for using the default algorithm’s configurations, thus emulating the case when the architect’s knowledge does not suffice to properly fine-tune the algorithm.

4.1. Single-Objective Optimization: Ericeira House

This case study was a room whose daylight conditions were modulated by a set of shading panels [3]. These panels are composed of a set of horizontal wood bars of different sizes, alternating between one full-length bar and a set of smaller bars, and defined in terms of the length’s step, the maximum distance separating two consecutive bars, and the minimum and maximum lengths of the smaller bars. In this case, the goal was to find a solution for the shading panels that maximized the room’s daylight performance, which was measured using the spatial Useful Daylight Illumination (sUDI) metric [14]. Figure 1 represents some design variations, ranging from denser patterns, with lower sUDI values, to sparser ones, with higher sUDI values.

Figure 1: Ericeira Solarium: Representation of the shading panels’ geometric pattern with different sUDI values (from left to right, 7%, 90%, and 100%).

We benchmarked the performance of 13 different derivative-free optimization algorithms: 5 direct-search, 3 metaheuristics, and 5 model-based. Given that each objective function evaluation takes 7 minutes to complete, we set a limit of 60 evaluations per run.

Table 1 shows the mean best results and the standard deviation of the three runs, discriminated by algorithm. According to the results, in average, the global model-based algorithms GPR, RBFCC, and RBFCL were able to find an optimal solution within the first 30 evaluations. Conversely, the local model-based algorithms COBYLA and BOBYQA performed rather poorly in this problem, converging to far from optimal solutions after 29 and 48 function evaluations, respectively. Regarding direct-search algorithms, the global algorithm DIRECT was able to find a close to optimal solution (with an sUDI value of 98%) in the last function evaluation. Its local variant, DIRECT-L, and the local direct-search algorithms Principal Axis (PRAXIS) and SUBPLEX fell short of the expected and were not able to improve over 80%. Nevertheless, the simplex-based direct-search algorithm NMS performed surprisingly well, having achieved an average result of 89.67% within the first 15 evaluations. Finally, although metaheuristics performed better than most local model-based and direct-search algorithms, they seem to stagnate in design solutions with sUDI values below the 88%, after 30 evaluations.

Figure 2 shows the average performance per algorithm class, also separating them in local or global algorithms. Overall, local algorithms seem to perform worse than all other algorithms, with local direct-search and model-based algorithms stagnating towards design solutions with sUDI values below 75% and 70%, respectively. Contrastingly, global algorithms were able to find design solutions with values of sUDI larger than 80%. Despite the good initial performance of metaheuristics algorithms for the first 20 evaluations, global direct-search algorithms quickly surpassed them, achieving close to optimal solutions with sUDI values of 90%. Lastly, global model-based algorithms were, on average, the best performing algorithms, achieving close to optimal solutions shortly after 24 evaluations.

Given the overall bad performance of local algorithms, we decided to assess their performance when initialized with different solutions. Notwithstanding their ability to quickly converge to locally optimal solutions, the quality of the found solutions

<table>
<thead>
<tr>
<th>Class</th>
<th>Global Algorithm</th>
<th>Mean</th>
<th>Best</th>
<th>Std Dev</th>
<th>Best</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based No</td>
<td>BOBYQA</td>
<td>68.67</td>
<td>6.03</td>
<td>29</td>
<td>10.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model-based No</td>
<td>COBYLA</td>
<td>71.33</td>
<td>13.20</td>
<td>48</td>
<td>15.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model-based Yes</td>
<td>GPR</td>
<td>99.00</td>
<td>1.00</td>
<td>25</td>
<td>16.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model-based Yes</td>
<td>RBFCC</td>
<td>99.00</td>
<td>0.58</td>
<td>30</td>
<td>9.27</td>
<td>30</td>
<td>31.00</td>
</tr>
<tr>
<td>Model-based Yes</td>
<td>RBFCL</td>
<td>99.00</td>
<td>0.58</td>
<td>30</td>
<td>9.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct-search No</td>
<td>NMS</td>
<td>89.67</td>
<td>16.17</td>
<td>15</td>
<td>9.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct-search No</td>
<td>PRAXIS</td>
<td>69.00</td>
<td>6.24</td>
<td>50</td>
<td>11.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct-search No</td>
<td>SUBPLEX</td>
<td>65.33</td>
<td>17.62</td>
<td>39</td>
<td>9.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct-search Yes</td>
<td>DIRECT</td>
<td>98.00</td>
<td>0.00</td>
<td>60</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct-search Yes</td>
<td>DIRECT-L</td>
<td>79.00</td>
<td>0.00</td>
<td>57</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metaheuristics Yes</td>
<td>CRS2</td>
<td>85.00</td>
<td>11.79</td>
<td>33</td>
<td>21.73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metaheuristics Yes</td>
<td>ESCH</td>
<td>78.67</td>
<td>4.93</td>
<td>54</td>
<td>5.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metaheuristics Yes</td>
<td>ISRES</td>
<td>87.67</td>
<td>6.81</td>
<td>35</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Mean best daylight results and mean evaluations to reach optimal solutions of each algorithm, averaged over three runs, each with 60 evaluations.
highly depends on the solution used to initialize the search. Therefore, we have also studied the impact of different initial solutions in the performance of these algorithms. To this end, we tested all 5 local algorithms with two different initial solutions: a bad solution, with a 7% value of sUDI, and a reasonable solution with a 78% value of sUDI. Moreover, we decided to further restrict the number of evaluations to 15, thus emulating a hypothetical scenario, where users lack knowledge about different optimization algorithms and opt for testing several of them. Ideally, this would allow them to identify the most promising algorithm and obtain a reasonable solution to hot-start other algorithms and, potentially, improve the overall optimization time. Results show that when provided with a mild solution, most local algorithms were able to converge to solutions with a 99% value of sUDI after 8 evaluations (or 56 minutes), against the 25 evaluations (or approximately 3 hours) required by global model-based algorithms. Conversely, as expected, local algorithms performed poorly when provided with a bad initial solution, barely managing to improve over the initial value.

4.2. Multi-Objective Optimization: Space Frame Optimization

This case study consists in the MOO of both the structural behavior and an ad-hoc measure of the irregularity of an arc-shaped space frame. The irregularities are caused by three attractors that cause a deformation in the shape of the truss. To measure the goals for each design variant, we used the Robot analysis tool to compute the maximum displacement of the structure, and the sum of the Euclidean distances between the attractors. Conversely, as expected, local algorithms performed poorly when provided with a bad initial solution, barely managing to improve over the initial value.

![Figure 2: Ericeira Solarium: Mean best daylight results in function of the evaluations number, discriminated per algorithms’ class.](image)

![Figure 3: Space Frame: Representation of two design variations of the arc-shaped space frame, with copper balls representing the attractors.](image)

frame. In fact, to reduce the maximum displacement, the attractors should be scattered across the space frame but this implies larger distances among the three attractors, thus worsening the second objective. Figure 3 illustrates two examples of the space frame structure.

We benchmarked 10 metaheuristics and 9 model-based algorithms. Each metaheuristic algorithm comprised a total of 15 individuals/particles per iteration, which were evolved for 15 iterations. Model-based algorithms produced 100 initial samples using the Latin Hypercube sampling algorithm, which were used to create the initial approximation to the expensive evaluation function, upon which another 125 evaluations were done. Overall, every algorithm was limited to a total of 225 function evaluations, each taking approximately 40 seconds to complete. In total, each run is composed of 4275 candidate solutions and takes approximately 2 days to complete.

Moreover, given the lack of more standardized approaches regarding the best way to evaluate the performance of MOO algorithms, we evaluate them using a set of indicators measuring the cardinality, the diversity, and the accuracy of the results returned by each algorithm, i.e., the approximated
Pareto Fronts (aPFs). Section 4.2 presents part of the obtained results averaged over the three runs, discriminated by the algorithms’ classes and subclasses.

Although some indicators measure aspects based exclusively on the aPFs, others require a reference set to compare with the aPFs. Ideally, this reference set would represent the real optimal solutions for the specified problem. Unfortunately, this set of optimal solutions, also called true Pareto Front (tPF), is not known for most BPO problems. In an attempt to better approximate it, we compute a fictitious Pareto Front, the combined Pareto Front (cPF), composed of the best solutions found by each algorithm.

Note, however, that we aimed at measuring the average performance of each algorithm regarding an unknown tPF. In general, computing an accurate approximation of the tPF would require running the algorithms for thousands of iterations, which is not feasible in most BPO problems. In an attempt to better approximate it, we compute a fictitious Pareto Front, the combined Pareto Front (cPF), composed of the best solutions found by each algorithm.

Spacing was used to measure the diversity of the aPFs, as it provides an estimate of the uniformity of the aPFs’ distribution. Considering this indicator, the two Evolution Strategy (ES) metaheuristic algorithms, PAES and CMA-ES, and one EA-based metaheuristic algorithm, ϵ-MOEA, achieved the most uniform aPFs. Conversely, model-based algorithms seem to yield more irregular aPFs, namely, MLP+NSGA-II and MLP+SMPSO achieved the worst values of Spacing. Note, however, that this indicator merely provides an idea of the regularity of distribution of the solutions. Ideally, this indicator would also suggest a good coverage of the cPF, i.e., that the aPFs found by each algorithm cover the same regions as the cPFs, instead of focusing on narrower regions. However, most of the algorithms that present the best Spacing scores achieve such values because most of the identified optimal solutions lie within the same small region but present a more uniform distribution. Besides this limitation, these indicators are also highly sensitive to outliers and to the number of retrieved solutions. Besides having an uniform distribution, it is also important to have a measure of the extent of each Pareto front (e.g., using the Maximum Spread indicator), as it implies that more relevant trade-offs are provided.

To measure the cardinality aspect, we consider the Overall Non-dominated Vector Generation Ratio (ONVGR) indicator, which computes the ratio of optimal solutions between aPFs and cPFs. In general, metaheuristics seem to retrieve the most nondominated solutions within each run, whereas model-based algorithms seem to retrieve the least. In fact, among the model-based algorithms, the algorithms exploring random search strategies, i.e., the algorithms suffixed with Random, yield fewer nondominated solutions, which may result from a poor exploration of the solution space. On average, the best performing algorithm, PAES, is able to find twice the number of solutions that compose each cPF, whereas model-based algorithms, including GPR+Random and MLPs algorithms, struggled to find a set of optimal solutions with at least half the size of the cPFs. While this indicator provides an intuition about the richness of each algorithm’s aPFs, many of the identified solutions might not be truly optimal, i.e., despite being optimal among all the evaluated solutions, these solutions might not belong to the corresponding cPF. To this end, other indicators should also be considered, such as the Error Ratio (ER).

Another important aspect of aPFs is their accuracy and how close their solutions are from the closest solutions in the corresponding cPFs. In this paper, we focus on the Generational Distance (GD) indicator, which measures the average approximation of each algorithm’s aPFs to the closest solutions in the corresponding cPFs. Section 4.2 shows that, on average, MLP+NSGA-II and PAES present the best convergence towards the cPF, and that GDE3 and OMOPSO present the worst convergence values. These can be explained by the number of nondominated solutions retrieved by each algorithm, as well as by the creation of clusters of optimal solutions near the

### Table 2: Space Frame: Mean values for the different aspects of the Pareto fronts, discriminated by algorithm. Results are averaged over 3 runs, each with 225 evaluations.

<table>
<thead>
<tr>
<th>Class</th>
<th>Subclass</th>
<th>Algorithm</th>
<th>ONVGR</th>
<th>Diversity</th>
<th>Accuracy</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metaheuristic</td>
<td>DE</td>
<td>GDE3</td>
<td>1.06</td>
<td>4.00</td>
<td>2.66</td>
<td>0.63</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>EA</td>
<td>e-MOEA</td>
<td>1.63</td>
<td>0.78</td>
<td>1.40</td>
<td>0.52</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>EA</td>
<td>MOEA/D</td>
<td>1.62</td>
<td>3.09</td>
<td>1.04</td>
<td>0.53</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>EA</td>
<td>NSGA-II</td>
<td>1.38</td>
<td>1.96</td>
<td>1.54</td>
<td>0.59</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>EA</td>
<td>PESA2</td>
<td>1.65</td>
<td>5.16</td>
<td>1.17</td>
<td>0.52</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>EA</td>
<td>SPEA2</td>
<td>1.27</td>
<td>6.65</td>
<td>1.15</td>
<td>0.55</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>ES</td>
<td>CMA-ES</td>
<td>1.46</td>
<td>1.39</td>
<td>1.48</td>
<td>0.43</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>ES</td>
<td>PAES</td>
<td>2.15</td>
<td>0.23</td>
<td>0.70</td>
<td>0.48</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>PSO</td>
<td>OMOPSO</td>
<td>0.78</td>
<td>6.13</td>
<td>1.74</td>
<td>0.69</td>
</tr>
<tr>
<td>Metaheuristic</td>
<td>PSO</td>
<td>SMPSO</td>
<td>1.36</td>
<td>1.96</td>
<td>1.69</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Note that for some algorithms, the number of evaluations is not 225, and the results are averaged over the available runs.
cPFs that were discovered by PAES and MOEA/D. In general, other model-based algorithms also present reasonable scores, with MLP+Random or all the RF-based algorithms even surpassing many metaheuristics algorithms, including CMA-ES, \( \epsilon \)-MOEA, NSGA-II, and SPEA2, thus suggesting better approximations.

In the end, we also used the Hypervolume (HV) indicator, as it appraises the quality of a Pareto front with regards to all three aspects simultaneously. The best performing algorithms were the PSO-based algorithms, SMPSO and OMOPSO, followed by GDE3. Surprisingly, the PSO model-based algorithms also present good performance when compared to other metaheuristics and even to other model-based algorithms that explore Random or evolutionary mechanisms. Conversely, the worst performing algorithms were the ES-based ones, CMA-ES and PAES, followed by GPR+Random.

Overall, no single algorithm was able to outperform the others in all indicators. Nevertheless, the PSO-based metaheuristics algorithms, OMOPSO and SMPSO, exhibited the overall best performance. Moreover, even though none of the model-based algorithms was able to surpass the SMPSO and OMOPSO, the model-based algorithms that use SMPSO also exhibited a reasonable performance, better than several well-known metaheuristics, including \( \epsilon \)-MOEA, MOEA/D, CMA-ES, and SPEA2. Figure 4 presents a combined view of all the algorithms for every run, where it is possible to visualize the extent of PSO-based algorithms and the high density region to which several EAs and ES algorithms converged.

5. Conclusions
Nowadays, with the threat of climate change, resource depletion, and worldwide urbanization, it is not enough to construct well-designed buildings, it is also necessary to optimize them [23]. Architectural practices have, therefore, grown to incorporate considerations about the building’s performance in various aspects. The development of computational simulation tools empowered designers with the ability to simulate and estimate a building’s performance. The emergence of these tools and the raising concerns about the environmental and economic impact of buildings led to the development of new design approaches, such as PBD, which seek more efficient design solutions by considering the designs’ performance. Taking PBD a step further, optimization has unveiled a new performance-based approach called BPO.

Unfortunately, traditional BPO methodologies require the evaluation of different design variations, which, in turn, implies spending a large amount of time with the manual application of changes to the design and often leading to difficulties when modeling complex geometry. Moreover, in order to evaluate a design’s performance, the corresponding analytical models must be produced, which also comprises a time-consuming and tiresome task. The emergence of algorithmic-based paradigms, like AD and AA, enabled the implementation of automated optimization processes, as they allow architects to generate multiple design variants with little effort, to automatically produce the corresponding analytical models and to automatically evaluate their performance.

Optimization algorithms can be coupled with the previously mentioned algorithmic approaches and simulation tools to more efficiently seek for optimal design solutions. Given that a single evaluation may take a considerable amount of time to complete, in order to speed up the optimization process, it becomes necessary to identify the optimization algorithms capable of handling the computationally complex problems that characterize BPO, and to devise strategies for its efficient application in architecture. Often disregarded, the selection of the appropriate optimization algorithm can have a significant impact in the overall efficiency of optimization processes and on the quality of the results [22].

However, most BPO practitioners adopt the simplest available algorithm, such as EAs [7], which typically require hundreds or thousands of evaluations - an infeasible scenario for most BPO problems. Conversely, direct-search and model-based algorithms are more promising, frequently yielding better results in fewer evaluations [21]. Particularly, model-based algorithms can considerably reduce the time spent in optimization [26], and, in fact, results show that these algorithms exhibit an average reduction of 50% on the total time spent in optimization.

Along these lines, we addressed optimization algorithms specifically tailored for handling simulation-based optimization problems, where the computational effort of simulation often restricts the number of function evaluations to a few dozens or hundreds. We introduced AO, an extension to the AD and AA approaches, that combines an optimization framework with an AD tool to address various design optimization problems, including BPO. Results show that no single class, subclass, or algorithm excels at every problem, thus corroborating the NFLTs for optimization [22].

Moreover, even though other tools rely on GA-based algorithms, these rarely achieved the best performance in the evaluated case studies. Instead, other categories, such as model-based or direct-search algorithms, yielded better results.
Different factors could change the obtained results (e.g., a different configuration of the algorithms or a lucky random step). Therefore, and contrarily to current architectural practices, we conclude that distinct algorithms behave differently according to the problems’ characteristics and that architects should first test various algorithms for a small number of evaluations or for a short amount of time. Furthermore, we also conclude that while global optimization algorithms are quicker to converge towards optimal solutions when no additional information is known, local algorithms can be quicker if provided with good starting points and, therefore, should be considered when such information is available.

Regarding the evaluation of different MOOs algorithms, the lack of consensus regarding the appropriate way to measure their quality makes it difficult to quantify the suitability of each algorithm for MOO problems. We conclude that algorithms’ quality should be measured through the combination of Pareto front plots and multiple MOO performance indicators. Particularly, some indicators should provide a measure of the diversity of the nondominated solutions across the solution space, while others should measure the overall accuracy of the results.

To overcome the identified limitations, the proposed optimization framework includes different categories of optimization algorithms and facilitates their application by abstracting them under a common interface, thus promoting automated optimization processes. To further facilitate the selection of the most appropriate algorithm, it also includes mechanisms to effortlessly test multiple algorithms. The framework was evaluated in the context of two BPO problems, which demonstrated its ability to solve real architectural problems characterized by computationally complex objective functions.

References


