TÉCNICO LISBOA

Optimization in Architecture

Catarina Garcia Belém

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. António Menezes Leitão

May 2019

Acknowledgments

I would like to express my respect and gratitude to my supervisor and friend Dr. António Menezes Leitão. He proposed an interesting theme, which proved to be intriguing and challenging. His efforts to arrange research grants and to supply better computational resources were inspiring and encouraged me to fight the difficulties found along the way. His constant support, preoccupation and first-class guidance were invaluable through this thesis. Thanks for everything, especially for encouraging me to pursuit my dreams and for providing me with the flexibility and free-will to tackle this theme as something that I would be proud of.

I would like to thank the members of the research group oriented by my supervisor, the Grupo de Arquitetura e Computação (GAC), for their support and valuable ideas and discussions which undoubtedly improved the practicality of this work - especially, Inês Caetano, Inês Pereira, Renata Castelo Branco, Guilherme Ilunga, and Luís Silveira Santos.

I would also like to thank the Department of Computer Science and Engineering at Instituto Superior Técnico, Universidade de Lisboa for providing me with the foundations for completing this work, as well as for the opportunities to lecture as a teaching assistant during my MSc Thesis. I would also like to thank the Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento (INESC-ID) for the financial support provided to me in the form of Bachelor's Research Grants.

I am also grateful to the staff and teachers of the Computer Engineering and Information Systems course for their effort, dedication, availability, and for providing an interesting working environment.

To all my friends whose support was invaluable during this period and which encouraged me to constantly push my limits when the task felt too large, I thank you deeply from my heart - especially, Carolina Pereira, Cristiana Tiago, Diogo Magalhães, Filipe Magalhães, Gonçalo Rodrigues, Guilherme Ilunga, Nuno Afonso, Pedro Simão, Rita Amaro, and Telma Correia.

Last but not least, I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my sister, brother, and sister-in-law, for their understanding, support, and preoccupation throughout this year.

To each and every one of you – Thank you.

Contributions

The development of this thesis resulted in several scientific contributions exploring different perspectives of optimization problems:

1. Caetano, I., Ilunga, G., **Belém, C.**, Aguiar, R., Feist, S., Bastos, F., and Leitão, A. (2018). Case Studies on the Integration of Algorithmic Design Processes in Traditional Design Workflows. Proceedings of the 23rd International Conference of the Association for CAADRIA, 1(Giedion 1941), 111–120.

2. **Belém, C.**, and Leitão, A. (2018). From Design to Optimized Design An algorithmic-based approach. Proceedings of the 36th eCAADe Conference - Volume 2, Lodz University of Technology, Poland, 549-558

3. Martinho, H., Leitão, A, **Belém, C.**, Loonen, R, and Gomes, M. G. (2019). Algorithmic Design and Performance Analysis of Adaptive Façades. Proceedings of the 24th Annual Conference of the Association for CAADRIA, Welligton.

4. **Belém, C.**, and Leitão, A. (2019). Conflicting Goals in Architecture: A study on Multi-Objective Optimization. Proceedings of the 24th Annual Conference of the Association for CAADRIA, Welligton.

5. **Belém, C.** , Santos, L. S., and Leitão, A. (2019). On the Impact of Machine Learning: Architecture without Architects?. 18th International Conference CAAD Futures 2019, Korea (Submitted).

Abstract

Keywords

Algorithmic Design; Algorithmic Analysis; Algorithmic Optimization; Derivative-Free Optimization; Machine Learning; Surrogate-based Modeling

Resumo

Palavras Chave

Design Algorítmico; Otimização de caixa-preta; Modelos baseados em aproximações; Aprendizagem Máquina.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AD	Algorithmic Design
AA	Algorithmic Analysis
BIM	Building Information Modelling
BPO	Building Performance Optimization
BPS	Building Performance Simulation
CAD	Computer-Aided Design
DIRECT	DIviding RECTangles
ES	Evolution Strategy
GUI	Graphical User Interface
HJ	Hooke-Jeeves
HVAC	Heating, Ventilation, and Air Conditioning
ML	Machine Learning
MOEA	Multi-Objective Evolutionary Algorithm
MOO	Multi-Objective Optimization
MOOA	Multi-Objective Optimization Algorithm
NFLT	No Free Lunch Theorem
NMS	Nelder-Mead Simplex
NN	Neural Network
PSO	Particle-Swarm Optimization

RBF Radial Basis Function

RF Random Forest

SOO Single-Objective Optimization

SVM Support Vector Machine

1

Introduction

Contents

The act of making something as fully perfect, functional or effective as possible is a behavior that is constantly sought by us, Humans, in a process known as optimization [Online, 2018]. Intuitively, through optimization one aims to improve a system in terms of different quantitative measurable aspects. Although usually striving to fully optimize these systems, i.e., to obtain *perfect* systems, it is often the case, that finding a better one or a near-optimal system suffices.

Generally, optimization processes are composed of two main parts: (1) the model of the system to be optimized and (2) the algorithm responsible for finding the optima. Conceptually, the model is a description of the system that is defined in terms of: a set of the system's characteristics, known as variables or unknowns, a set of quantitative measures of the system's performance, referred to as objectives or criteria, and, optionally, by a set of conditions that have to be satisfied to guarantee the system's feasibility, i.e., the system's constraints [Nocedal and Wright, 2011]. The objectives are usually functions of the variables being defined. Subsequent to the model definition, the obtained description can be interpreted as an optimization problem for which the optimal solutions are to be found, thus entering in the second part of an optimization process. In the second part, one executes an optimization algorithm, which encloses a description of the steps necessary to attain optimal solutions, which according to the user's intentions can be the maximization or minimization of the model's objectives.

Depending on the model representation, one is able to classify optimization problems differently with respect, for example, to its objective functions, variables, and determinacy. Due to their relevance in the developed work, in the next two paragraphs, we describe four different optimization classifications. However, we refer the interested reader to [Nocedal and Wright, 2011] for a more detailed and complete description of the different classifications.

One important classification is regarding the cardinality of the solutions sought by optimization processes, thus yielding the continuous and discrete optimization categories. In the former, optimal solutions lie in a potentially infinite set of candidate solutions, whereas in the latter, optimal solutions lie in a finite set. Optimization problems can also be classified as constrained or unconstrained, depending on whether the models explicitly define constraints or not. Moreover, optimization can also be distinguished in terms of the aim of the search that is performed, particularly, whether it is global or local. In local optimization, the search process strives to find a solution that is locally optimal, i.e., for which its value is better than all other points in its vicinity. The points that satisfy the previous property are known as local optima. On the other hand, there are optimization processes that strive to find the globally optimal solutions, i.e., the best of all the local optima.

Optimization is frequently required to address problems involving more than one objective. It is often the case that people face daily decisions involving two or more conflicting objectives, either to effectively manage resources, or just to ponder several factors associated with certain decisions. A simple example is the act of purchasing something, e.g., a computer or a car. In these settings, people

often ponder different aspects to determine the best choice, i.e., to optimize its decision. In general, when looking for a computer, one is usually interested in minimizing the price of the computer, while maximizing the processing speed, the RAM and storage capacity and speed, and the battery autonomy. However, while some aspects are independent of each other, e.g., processing speed and RAM capacity, other aspects are conflicting and preclude the simultaneous satisfaction of all the objectives, e.g., the higher the processing, storage or RAM speed, the higher the price, the higher the processing speed, the worse the battery autonomy. While in this example, we only consider a subset of aspects many other might be considered, possibly complicating the decision process. Because this example considers the optimization of more than one aspect, it belongs to a subset of problems, known as Multi-Objective Optimization (MOO) problems. Whereas the set of problems focused in the optimization of a single aspect are known as Single-Objective Optimization (SOO).

In addition to day-to-day life decisions, optimization can also be used with different decision and analysis purposes. As a result, several areas, including economy, science, engineering, among others apply optimization as auxiliary tool to maximize the efficiency of the decisions involved. Particularly, architecture is one of the areas where the potential of optimization becomes more visible. The architectural practice can benefit from optimization to reduce the building sector's economic and ecological footprint through the finding of more efficient buildings variants before their construction. Given its importance to the world's sustainability and economy, this thesis focus on the application of optimization processes to enhance the architectural practice, providing, in the following sections, an overview of the involvement and the evolution of such processes in architecture. We end this chapter by highlighting our research goals and by outlining the upcoming document's structure.

1.1 From design to Optimized design

The usefulness of optimization goes beyond architectural design applications, benefiting other engineering fields like Mechanics or Electronics, through the optimization of its components and circuits designs, respectively.

In the architectural practice, optimization has been gaining relevance for the past few years [Cichocka et al., 2017a], especially due to the impact of building construction and building maintenance in the economy and environment. For this reason, designers are shifting from a pure aesthetically-based to performance-based design, where buildings are being optimized to achieve the best possible values regarding different aspects of their design, such as thermal comfort, energy consumption, lighting comfort, structural behavior, cost, among others.

This has only been possible due to the technological improvements in the architectural practice over the last few decades. The adoption of computer science techniques was responsible for the dissemina-

tion of digital modelling tools, which allowed the more accurate and efficient design of highly complex buildings. These tools enabled a shift from traditional paper-based approaches to more computerized ones, such as Computer-Aided Design (CAD) and Building Information Modelling (BIM) applications, where changes to designs are trivial and do not require manually erasing and redrawing parts of the original design [Ferreira and António, 2015]. Figure 1.1 illustrates the general view of this computer-aided design process, as well as an example of a 3D modeling tool. The architect interacts directly with these modeling tools to incrementally concretize his design ideas.

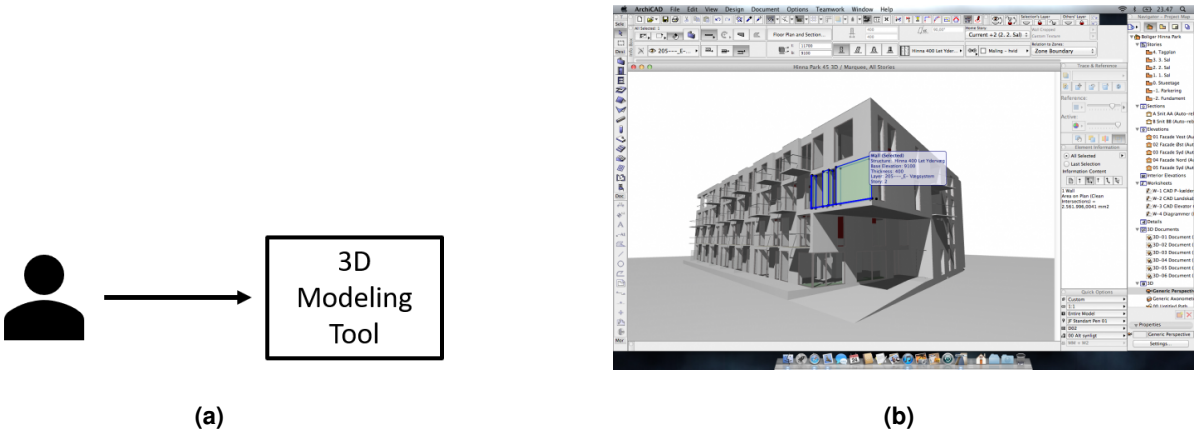


Figure 1.1: (a) Computer-aided design workflow (b) Building design example in a 3D modeling tool

Shortly after, the development of computer-based simulation tools allowed designers to simulate their building's behavior regarding specific criteria, and, thus get a measurement of its performance [Malkawi and Kolarevic, 2005]. Through this process, called Building Performance Simulation (BPS), designers could easily validate whether their building's performance satisfied the efficiency requirements and, ultimately, optimize their design by iteratively generating multiple variations of the same design, assessing their performance, and selecting the better ones. Albeit being very primitive, architects now had the elementary mechanisms required for optimizing their building's designs.

1.1.1 Building Performance Optimization

Building Performance Optimization (BPO), a simulation-based optimization approach, treats the results produced by the simulation tool as the functions to optimize. Although invariably suffering from some degree of imprecision and inaccuracy, using these simulations it becomes possible to estimate the performance of complex designs. Particularly, these estimates are beneficial in designs for which analytical solutions are often very difficult or even impossible to derive [Kolda et al., 2003]. In these cases, the objective function, i.e., the function to optimize, is derived from the simulations' results. These objective functions have a domain which corresponds to the range of acceptable designs as specified by the

architect.

A known drawback of simulation-based approaches is the time required to achieve reasonable results for complex systems [Law and Kelton, 1991] which is associated with different aspects of the problem, namely (1) its **domain** which, depending on the nature of the problem, might use different methodologies to produce the corresponding estimates (e.g., thermal *versus* structural); (2) its **intrinsic structure** which, depending on the attributes and relations of the system, might lead either to simpler or to more complicated computations (e.g., skyscraper *versus* a small house); and (3) its **analytical model**, which has the essential properties of the system we are trying to simulate and that will be used as input to the simulation tool. Generally, the domain and structure do not change for the same problem, albeit there are numerous ways to produce multiple analytical models. Depending on the level of detail of the analytical model, both the computational time and the result of the simulation might change.

Dar exemplo??

In architecture, the generation of each analytical model is a time-consuming and complicated task. On the one hand, it is often necessary to generate multiple models of the same design because of the simulation tools' specificity, i.e., in order to evaluate a design, each simulation tool requires a specialized model of the same design. On the other hand, simulation tools often yield time-consuming processes, where a single simulation can take up to seconds, minutes, hours, days, or even weeks to complete.

In addition to the simulations' specificity and complexity, architectural designs are inherently complex, thus leading to less predictable objective functions, for which mathematical forms are difficult to formulate [Machairas et al., 2014]. For this reason, information about the derivatives of such functions cannot be extracted and methods depending on function derivatives cannot be used to address architectural optimization problems. As a result, classical gradient-based optimization methods can not be used. Instead, other methods that do not rely on the existence of an explicit mathematical form should be used, i.e., methods that treat the optimization functions as black-boxes, relying uniquely on the outputs of numerical simulations.

Despite the flexibility provided by CAD and BIM tools, architects often face difficulties when modeling complex geometry. A BPO methodology requires to experiment with different design variations, which implies spending a large amount of time to manually make changes to the design. Since an optimization process requires evaluating several variations of the same design, the manual execution of the required changes implies a lot of effort, hence making the whole optimization process unviable.

1.1.2 Algorithmic Design

An approach capable of creating forms through algorithms is crucial for overcoming the aforementioned limitations. An example of an algorithm approach is the Algorithmic Design (AD) approach [Branco and Leitão, 2017], which is depicted in Figure 1.2. In this approach, the architect entails an algorithmic description of the intended design. After elaborating the algorithm, executing it will automatically generate

the corresponding 3D model in a CAD or BIM tool. Algorithmic approaches are inherently parametric, thus enabling the generation of different variations of the same design by making simple modifications to the values of the parameters [Leitão et al., 2014].

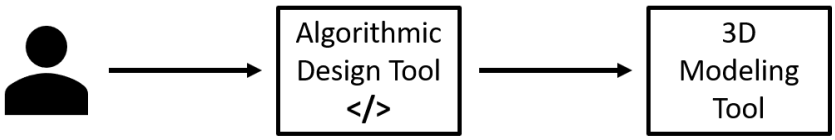
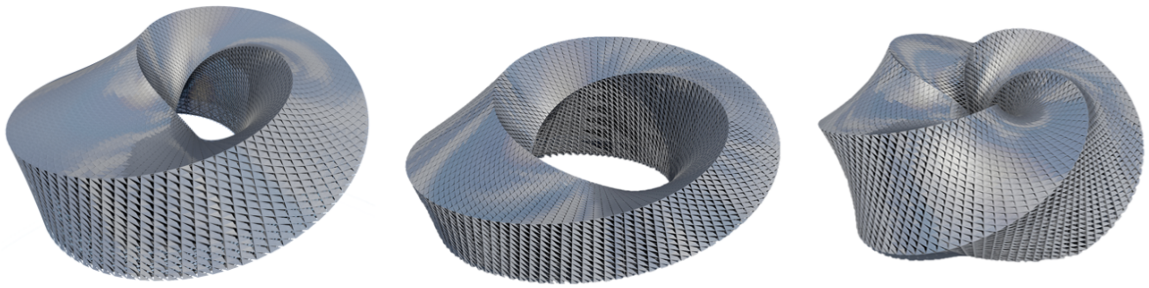


Figure 1.2: Algorithmic-based design workflow

As an example consider the algorithmic design of the Astana's National Library from BIG architects, illustrated in Figure 1.3(a). Initially, the architect selects an AD tool providing the necessary design primitives. Then, he uses these primitives to create a computational program enclosing the relative relations among the different design elements, so that when a modification occurs in one element, that modification is propagated throughout that element's dependencies. In the end, the architect creates a procedure responsible for creating the whole design `astana(radius, nturns)`, which when executed will produce the corresponding Astana 3D model. Using this procedure, he can easily generate different variations of the Astana by simply setting different values for the parameters: `radius`, defining the width of the whole design, and `nturns`, defining the number of turns in the design. Figure 1.3 illustrates the original design and two design variations are represented: (a) the original design (b) design variation generated by doubling the radius of the original design, and (c) design variations generated by doubling the number of turns.



(a) **(b)** **(c)**

Figure 1.3: Astana's National Library Design variations: (a) Original (b) Larger diameter (c) Two Moebius Turns

Only recently has the algorithmic paradigm begun to settle in the architectural practice. The requirement for programming knowledge is often an obstacle to the adoption of these approaches, since it requires larger initial investments for the architect to learn how to program. Despite the larger initial investments, the benefits obtained with algorithmic approaches surpass the ones obtained when us-

ing CAD or BIM tools for the design of complex buildings. Particularly, the initial investments can be quickly recovered when the need for the incorporation of changes arises or when it becomes necessary to experiment different design variations [Leitão et al., 2014]. This is especially important when facing design processes characterized by continuously changing constraints and requirements of the design. In these scenarios, a manual-based approach requires to constantly change the design by hand, thus incurring in a dreadful and tiresome process, whereas an algorithmic approach enables the effortlessly generation of a broader range of design solutions, as well as the easy modification of the algorithmic model to comply with new requirements. As a result, with algorithmic approaches, architects are able to explore larger regions of the design solution space, as well as to explore innovative solutions which were not previous considered due to their time and effort complexity [Leitão et al., 2014].

The appearance of AD was crucial for the automation of optimization processes as it enables the automated generation of multiple designs by simply changing the values of the design's algorithmic model. However, to optimization such designs, it is necessary to create the analytical models, which can be very dissimilar to the 3D models produced by the AD tool. Therefore, to evaluate the 3D models produced by the AD tool, the architect must manually generate the corresponding analytical models. Particularly, for complex buildings, this task requires large time and effort investments, which make most optimization processes impracticable.

1.1.3 Algorithmic Analysis

Faster and broader design space exploration prompted the creation of increasingly complex building designs, which became less predictable with respect to different aspects [Branco and Leitão, 2017], such as thermal, lighting, acoustics, among others. Moreover, recent requirements for efficient and sustainable buildings led to the demand for buildings that not only are well-designed, but also exhibit a good performance at those aspects.

Nowadays, most of the available simulation-based analysis tools are single-domain toolsets, each analysing different parameters within their domain [Malkawi and Kolarevic, 2005], i.e., while a lighting analysis tool measures daylight and glare coefficients, an energy simulation tool measures other coefficients related to thermal, energy consumption, and Heating, Ventilation, and Air Conditioning (HVAC) systems. Unfortunately, this often implies the production of different analytical models for the corresponding simulation tools. Moreover, the 3D models produced by 3D modeling tools are generally dissimilar to the specialized models required by each analytical tool. To evaluate the design performance on different domains (e.g., lighting, energetic, structural), the corresponding analytical models have to be produced either by hand, or through translation processes that convert generic 3D models into specialized models required for analysis.

Currently available techniques for the production of analytical models exhibit a few limitations, includ-

ing: (1) hand-made analytical models might be a more faithful representation of the original model but they require a considerable amount of effort to create; (2) despite the existence of tools that attempt to convert a 3D model into the corresponding analytical model, this conversion is frequently fragile and can cause loss of information or errors; (3) ideally, the results of the analysis would then be used to guide changes in the original design, but these changes require additional time and effort to implement, as does redoing the analysis to confirm the improvements. This explains why performance analyses are typically postponed to later stages of the design process, only to assess the fulfillment of the performance requirements.

To overcome the time and effort limitations associated with the production of analytical models, one can exploit the ideas of algorithmic approaches to automatically generate the necessary analytical models from an algorithmic description. Algorithmic Analysis (AA) is an extension to the AD approach that besides enabling the automatic generation of analytical models from a design's algorithmic model, also automates the setup of the analysis tool and the collection of its results [Aguiar et al., 2017]. Following this approach, in an AD tool, the architect creates the algorithmic model that describes his design's intents and then changes its configurations as to reflect the analysis tool to use, for example, by changing the values of the configuration parameters. Figure 1.4 illustrates the AD and AA design workflow, as well as examples of the Astana's National Library models produced for each tool. Note that, even though the abstract description of the design is the same for 3D and analytical models, the produced models can be very different, e.g., a truss is represented as a graph of nodes and edges when submitted for structural analysis, whereas for lighting analysis the truss is represented by its surfaces, and, in the 3D model, the truss is represented by a set of masses representing its bars and nodes.

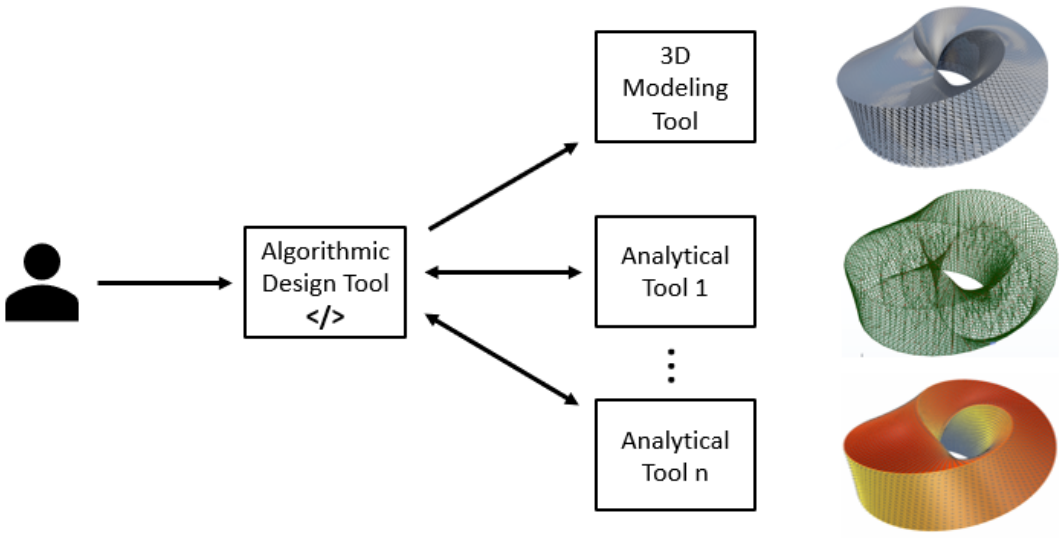


Figure 1.4: Algorithmic Design and Analysis design workflow with examples of the Astana's National Library design and analytical models: (top) 3D model; (center) Robot's structural analysis model; (bottom) Radiance's pos-radiation analysis model.

The AA approach is able to enhance performance-based design approaches, as it provides means to effortlessly perform analysis throughout the whole design process, instead of just at final stages. Depending on the performance requirements, architects might need to use different analysis tools: (1) for daylight analysis, DAYSIM and Radiance are very popular among the community, (2) for energy simulations, EnergyPlus, TRNSYS and DOE-2 are widely used [Nguyen et al., 2014], (3) for structural analysis, Robot Structural analysis is a well-known reputed tool, and (4) Olive Tree Lab and Pachyderm Acoustical Simulation are examples of good acoustic analysis tools.

The AA approach was also very important to allow the automation of optimization processes, as it abstracted the production of the analytical model, removing the need for direct human intervention and reducing the errors and information losses. Additionally, together with AD, it provides the mechanisms to quickly update a design, to generate the corresponding analytical model, to automatically evaluate the design in an analytical tool, and, finally, to collect the results and use them to guide the search for optimal solutions. Despite being possible to automate optimization processes, in order to do so, the architect must define a script entailing an optimization algorithm every time he intends to attain better designs. Besides lacking enough knowledge or expertise about optimization algorithms themselves, in order to entail one, architects would have to either create their own algorithm, or they would have to use one available in some optimization framework. Moreover, because of the efforts associated with the implementation of the optimization algorithm, architects will be tempted to adopt the same algorithm to optimize every design, thus instilling performance impacts in the overall optimization time. Among other obstacles, the large time and investment efforts, as well as the lack of knowledge are often main setbacks to the application of optimization processes in architectural design contexts.

1.1.4 Architectural Optimization Workflow

In architecture, design optimization might be approached differently. In the past, most optimization processes were comprised of different frameworks, which often had to be integrated with each other. Architects often attempted to integrate existing mathematical optimization and visualization frameworks within their workflow. Due to their specificity integration problems were often obstacles to the optimization process itself. More recently, the emergence of visual parametric approaches, such as Grasshopper, Dynamo, among others, coupled with the growing consciousness of both the limitations and the benefits of optimization in building design have led to the development of ready-to-use optimization toolsets (e.g., Galapagos, Goat, Octopus, Opossum, Optimo). However, despite enabling the optimization of several designs, visual parametric tools are known to scale poorly with the complexity of the design , thus diminishing and constraining its optimization capabilities.

COLOCAR REF

On the other hand, algorithmic approaches are known to scale well with designs' complexity. In addition to its scalability benefits, its growing popularity among building design practitioners [De Kestelier

and Peters, 2013], its models' flexibility, as well as its capacity to automate optimization processes allow the development of more robust and complete optimization tools. To successfully take advantage of such a tool, an AD-based design workflow optimization methodology must be followed. In this approach, the architect idealizes a design which he ought to produce in the corresponding AD tool. For this purpose, he creates the computational program, defining the parameters that represent the degrees of freedom in the design, i.e., the parameters which he is willing to manipulate in order to achieve more efficient designs. After the conception of the design's algorithmic model and, provided the values for the parameters, the AD tool generates 3D or analytical models for visualization and performance analysis purposes, respectively. Optionally, the architect may decide to optimize his design according to some particular aspects, potentially leading to the exploration of design solutions that were not previously considered. In that case, the optimization algorithm explores different design candidates, using the results produced by simulation tools as the functions to optimize. The execution of the optimization algorithm then yields optimal (or near optimal) design solutions.

Considering the previous view of an algorithmic-based design workflow, we identify four key dimensions in an optimization process:

1. **Analytical models**: when the optimization algorithm specifies a candidate design, i.e., a concrete configuration for the parameters of the model, analytical models are automatically generated by the AD tool. These models are then used as input for the corresponding analytical tools. These models can be improved, either through simplification of the analytical models, or by enriching them with context information. The former enables the simplification of the analysis itself and potentially reduces the simulation time, by providing an equivalent but simpler model to the tool, whilst the latter enables the attainment of a more detailed and realistic simulation, which is not always possible due to limitations in the AD tool.

2. **Optimization algorithms**: the algorithms that explore the design space in the quest for optimal (or near optimal) solutions. These algorithms use the results obtained from performance analysis of different design variations as the functions to optimize, i.e., as objective functions. Generally, the algorithm uses these inferred functions to guide the search for optimal solutions. The time complexity of the algorithm is typically dependent on the number of function evaluations. In architectural design, these functions entail time-intensive simulations, thus instilling optimization processes that may take minutes, hours, days, or even weeks to complete.

3. **Intelligibility of Results**: Cichocka et al. identify the need for intelligibility of the optimization processes within the architectural community [Cichocka et al., 2017a]. Having access to an explanation regardig the quality of a design solution allows architects to make more informed decisions. With these explanations, the archtiect can not only provide valuable arguments for its implementa-

tion, but also, depending on the quality of the explanations, learn with the process, thus fostering more efficient and faster future designs.

4. **Interactivity and Visualization**: interactive and visual aspects are highly important features in the context of an optimization process [Ashour, 2015]. On the one hand, an interactive optimization process enables its user to transfer knowledge about the problem at hand, for instance, by adding or removing constraints or by exploring different, yet unexplored regions of the design space, hence potentially increasing this process' performance. On the other hand, optimization processes providing better visualizations and representations of their own evolution can present their users with better feedback about the course of the search. This feedback is important, as it also allows the comparison of variable-objective correlations and the making of more informed decisions about the optimization process itself, e.g., whether the evaluations made so far suffice or if the algorithm is converging to non-conventional designs that he refuses to accept.

1.2 Goals

The interest in design optimization is evident within the architectural community, however the currently existing tools are often fragile or limited, frequently compromising the application of optimization in architecture. This thesis focus on optimization processes within the architectural domain, providing a framework for optimizing both single and multi-objective problems. The implementation of such framework requires the definition of: (1) a modeling language to support the specification of optimization problems, (2) a wide variety of optimization algorithms to solve optimization problems, and (3) a visual presentation of the obtained results to provide a more comprehensible feedback over the optimization results.

To achieve the goals that we proposed to, we reviewed different mathematical optimization modeling languages and optimization frameworks, pondering the benefits and obstacles of each one. Based on these languages and frameworks, we established the basis requirements for the framework that enable its seamless application within the architectural practice.

1.3 Organization of the Document

This thesis is is organized as follows:

Chapter 1 discusses optimization concepts and evidences its importance for different problems, ranging from simpler day-to-day decisions to more complex engineering problems, such as components, circuits, and building designs. Particularly, this chapter stresses the relevance of optimization in the architectural context, providing a comprehensive overview of the existing practices and the difficulties underlying the

adoption of optimization processes in architecture.

Chapter 2 provides an overview over the currently existing optimization practices in architecture and makes a balance of the benefits and drawbacks associated to each one.

Chapter 3 describes the architecture of the implemented framework and enumerates important design decisions that were made during its implementation.

Chapter 4 evaluates both quantitative and qualitative aspects of the proposed solution, evaluating its performance in the context of three real-world case studies.

Chapter 5 emphasizes the importance of optimization in architecture and draws some conclusions about the final work and how it can effectively influence the architectural practice. Finally, we reflect over future improvements for the proposed frameworks.

2

Background

Contents

The development of an algorithmic-based framework for optimization, applicable to architectural domains, requires a careful review over the current literature on BPO practices and limitations.

Firstly, the *ad-hoc* nature of the functions used for performance assessment in BPO motivates the application of a special class of optimization algorithms, the derivative-free algorithms. Within this class, different categories emerge, emphasizing algorithms' different properties and search strategies. Each algorithm is able to potentially increase the effectiveness of an optimization process, depending on the characteristics of the considered problem.

Secondly, there are multiple approaches to optimization that might be considered. Generally, BPO practices include the simultaneous optimization of multiple aspects. However, they often opt for simpler specifications, often disregarding all but one of the initial considered aspects.

Finally, currently available architectural design optimization tools explore the parametric models produced in visual programming environments, such as Grasshopper and Dynamo. These graphical environments are implemented as plug-ins, which are tightly integrated with a CAD and a BIM tools, respectively. As a result, the connection between optimization tools and visual design workflows becomes seamless and friendlier. These optimization tools usually expose a *ready-to-run* interface, which is very appealling to most BPO practitioners [Cichocka et al., 2017a].

2.1 Derivative-Free Optimization

Different optimization algorithms can solve more or less efficiently specific optimization problems, depending on their characteristics. Particularly, classical gradient-based algorithms are very efficient solvers for optimization problems explicitly defined by mathematical formulations. This results from the fact that gradient-based algorithms explore information about the derivatives extracted from the mathematical formulation to guide the search for optimal solutions. However, when neither the mathematical form is easily available, nor is the information about the derivatives, it becomes necessary to explore other classes of algorithms. In these cases, the class of derivative-free algorithms is remarkably suitable for addressing these problems, as they do not use information about the objective functions' derivatives to find optimal solutions, instead, treating the objective functions as *black-boxes* and guiding the search based on the result of previously evaluated solutions [Rios and Sahinidis, 2013].

In architecture, it is often impossible to mathematically model the underlying objective functions for complex designs. Alternatively, architects use simulation tools as means to replace closed-form mathematical expressions that relate the design's parameters to the objective functions: simulation tools' results and other quantitative measures for different configurations of a design define the objective functions to optimize [Wortmann and Nannicini, 2016]. Additionally, information about underlying objective functions is not easily attainable, often requiring excessive amounts of resources. The absence of in-

formation about objective functions prompts the need for algorithms that treat these functions as *black-boxes*. One simple approach is to systematicly experiment with different parameter values until the best solutions are found, whereas a second, and more complex, approach is to use derivative-free optimization algorithms, also commonly referred to as black-box optimization algorithms within the architectural community [Wortmann and Nannicini, 2016]. Despite its simplicity, experimentation-based approaches, such as Monte Carlo Sampling and Latin Hypercube Sampling [Giunta et al., 2003], might not always be advisable, particularly when dealing with time-consuming functions as is the case of architecture. In such cases, derivative-free algorithms might be more appropriate, yielding better solutions in less time.

Building design's complexity has been raising for the past few years, leading to more complex objective functions, for which analytical forms are difficult to derive [Machairas et al., 2014]. For this reason, derivative-free algorithms are sought as useful tools to optimize designs, having been applied extensively to optimize building designs' manifold aspects. Among the numerous studies that apply derivative-free optimization algorithms to optimize building designs, we refer to the distinct works of Wortmann [Wortmann and Nannicini, 2016, Wortmann et al., 2015, Wortmann et al., 2017, Wortmann, 2017b], Evins [Evins and Vaidyanathan, 2011, Evins et al., 2012, Evins, 2013], and Waibel [Waibel et al., 2018] which cover the optimization of various aspects, including, among others, the structural, the lighting, the thermal, the energy consumption, and the carbon-emissions.

For the past decades, the constant development and improvement of derivative-free optimization algorithms led to a diversified tools' gamut, each with its own characteristics and limitations. While the main ideas behind each algorithm's category seem to be more or less recognized throughout the architectural community, the lack of standards make it difficult to decide which definitions to convey [Rios and Sahinidis, 2013, Wortmann and Nannicini, 2017]. The currently most relevant classifications are: (1) the one presented by Rios et al. [Rios and Sahinidis, 2013] that, based on the functions being used to guide the search process, classifies the algorithms into direct search or model-based algorithms; and (2) the classification provided by Wortmann et al. [Wortmann and Nannicini, 2017], which first subdivides the algorithms in two groups according to the number of solutions generated in each iteration, namely metaheuristics and iterative algorithms, and only then proceeds to classify iterative algorithms as direct search or model-based algorithms depending on the function that is used during the search.

This thesis will consider an approach similar to the one proposed by Wortmann [Wortmann and Nannicini, 2017] by exploring the concepts of metaheuristics, direct-search, and model-based algorithms. Albeit the apparent chasm between these classifications, some algorithms draw ideas from distinct classes, thus emphasizing not only the blurred lines of such categorizations, but also the difficulties that lie with the definition of more standardized classifications.

The following sections describe each class and its intrinsic characteristics, proceeded by a brief comparison among them in light of the architectural design practice.

2.1.1 Direct Search Algorithms

Although there seems to be no precise definition for direct search algorithms [Kolda et al., 2003], these are often identified as algorithms that iteratively [Kolda et al., 2003, Wortmann and Nannicini, 2016]: (1) evaluate a finite sequence of candidate solutions, proposed by a simple deterministic strategy; and (2) select the best solution obtained up to that time. They are sought as valuable tools to address complex optimization problems, not only because most of them were proved to rely on solid mathematical principles, but also because of their good performance at initial stages of the search process [Rios and Sahinidis, 2013, Wortmann and Nannicini, 2016].

The main limitations of the algorithms in this class is their performance deterioration with the increase on the number of input variables and their slow asymptotic convergence rates as they become closer to the optimal solution [Kolda et al., 2003].

Some examples of relevant direct-search algorithms include Hooke-Jeeves (HJ) [Hooke and Jeeves, 1961], Nelder-Mead Simplex (NMS) method [Nelder and Mead,], SUBPLEX [Rowan, 1990], DIviding RECTangles (DIRECT) [Jones et al., 1993], among others.

Give a tiny description of each algorithm

2.1.2 Metaheuristics Algorithms

In the original definition [Glover and Kochenberger, 2003], these algorithms were solely based in the interaction between local improvement procedures, called heuristics, and higher-level strategies, called metaheuristics. On the one hand, heuristics are techniques that locate good solutions, but not necessarily the optimal, nor the correct solution, and that often consider the trade-off between precision and quality, and computational effort. On the other hand, a metaheuristic is an algorithmic framework that can be applied to different problems with a few modifications to add problem-specific knowledge [Glover and Kochenberger, 2003], if so is desired. Moreover, a metaheuristic is a higher-level strategy that extends the capabilities of heuristics by combining one or more heuristic methods (referred to as procedures), while being agnostic to each heuristic. Metaheuristics are “meta” because they control heuristics.

Throughout time, this class has grown to include any algorithm that includes simple heuristics to locate good solutions in complex design spaces, while considering the trade-off between precision, quality, and computational effort of the solutions. These algorithms often rely on randomization, and biological or physical analogies, to perform robust searches and to escape local optima [Glover and Kochenberger, 2003, Wortmann and Nannicini, 2016]. Additionally, their non-deterministic and inexact nature confers them the ability to effortlessly handle complex and irregular objective functions [Wortmann et al., 2017], as well as, to easily adapt to MOO contexts, or even to provide with domain-specific knowledge through the heuristics [Wortmann et al., 2017].

Metaheuristics are efficient optimization algorithms when provided with sufficient amount of time to do the necessary objective function evaluations [Conn et al., 2009]. However, advantages can quickly become disadvantageous by simply changing the application context. This is the case of BPO in the architectural practice, where each evaluation is a time-consuming task and the execution of thousands of evaluations rapidly becomes an infeasible scenario. Due to their stochastic nature, limiting the number of evaluations has severe repercussions both on the convergence and performance guarantees [Hasançebi et al., 2009].

In the architectural design context, some of the most relevant metaheuristics algorithms include the Particle-Swarm Optimization (PSO) algorithms, some evolutionary algorithms, such as **GA!** (**GA!**) and Evolution Strategies (ESs), and even local search algorithms like tabu search and simulated annealing. We refer the interested reader to [Blum and Roli, 2003, Glover and Kochenberger, 2003] for more details about these metaheuristics algorithms.

Give a tiny description of each algorithm

2.1.3 Model-based Algorithms

Problems with time-consuming computer models and where sensitive information is expensive to collect can be approached effectively by model-based algorithms [Forrester and Keane, 2009, Wortmann and Nannicini, 2016]. These problems are characterized by the large time complexity associated with the computation of the values of the objective function and by the absence of previous knowledge about the objective function. Model-based algorithms are able to provide instant estimates of a design's performance, by supplementing or replacing the original objective function by its approximation [Wortmann and Nannicini, 2016]. This approximation, called the surrogate, is generated from a set of known objective function values, and is then used to determine the promising candidate solutions to evaluate next. These candidate solutions are then used to improve the surrogate and this process is repeated until a stopping condition is satisfied [Koziel and Yang, 2011].

Despite having a well-defined analytical form, which makes computations on the surrogate model more efficient than on the original objective function, the surrogate is only an approximate representation of the original function, and, therefore, must be constantly updated to guarantee a reasonable locally accurate representation [Koziel and Yang, 2011]. Figure 2.1 illustrates a surrogate that is accurate near the initial solutions. However, as we analyse solutions far from the initial ones, the accuracy of the surrogate model worsens.

Nowadays, the existing plethora of techniques applicale to the generation of surrogate models, range from trust region methods to Machine Learning (ML) techniques. These techniques can be used to create (1) local surrogates, i.e., models where the approximation to the objective function is built around a certain point, and (2) global surrogates, i.e, models where the approximation is generated from all the

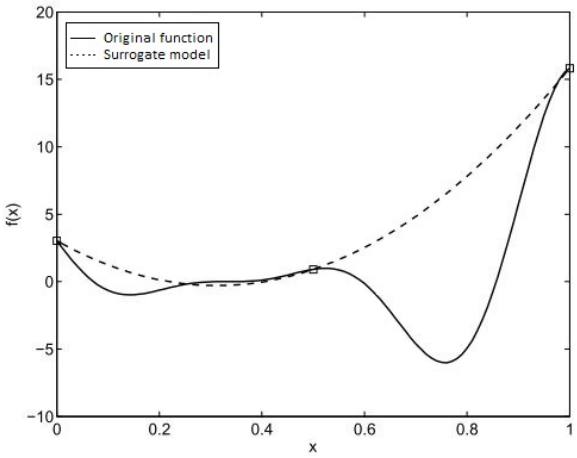


Figure 2.1: Original function and corresponding surrogate model, created based on three initial solutions (squares). This image was retrieved from [Koziel and Yang, 2011].

obtained points. Whilst the former relies on the construction of simple, partial models of the objective function, the latter relies on the creation of a full model. The creation of the full model, requires balancing the need for improving the accuracy of the model by exploring broader regions in the solution space, with the need for improving the value of the objective function by exploiting promising regions [Koziel and Yang, 2011]. This balance is determined by a strategy that selects the next promising solution to evaluate.

Undoubtedly, the best feature of model-based algorithms is the reduction in the total optimization time. This is particularly relevant in the context of BPO, where each simulation may take seconds, minutes, hours, days, or even weeks to complete. However, despite their benefits, the low availability and the technical knowledge frequently required to implement or incorporate these algorithms into optimization processes is not easily accessible to architects. Thus, despite the existence of different reports involving ML techniques for the creation of full surrogate models [Koziel and Yang, 2011, Forrester and Keane, 2009], such as Neural Networks (NNs), Support Vector Machines (SVMs), Radial Basis Function (RBF), and Random Forests (RFs), among others, 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 multi-objective optimization context.

Give a tiny description of each algorithm

2.1.4 Comparison

This section considers the applicability of different classes of derivative-free optimization algorithms for architectural design. The multidisciplinary character of building design raises distinct problems, ranging from well-behaved problems with simple, unimodal, convex functions to more ill-behaved problems with

irregular, multimodal objetive functions [Wortmann and Nannicini, 2017]. In addition to problem's diversity, the time complexity associated to function evaluations also becomes an important factor to consider when pondering each category's impact on BPO problems.

The problems' plethora within performance-based design is vast: a specific optimization algorithm may perform well for some problems and have a terrible performance in other problems [Wortmann et al., 2017, Fang, 2017]. This idea resembles the ones captured in Wolpert's No Free Lunch Theorems (NFLTs) for optimization, which state that any algorithm's worse performance over some classes of problems offsets its better performance in other classes. Because of the distinct nature of architectural design problems, the arguments herein provided are not necessarily applicable to other fields like science and engineering.

Inevitably, the same building design description might yield different problem descriptions according to the performance aspects being considered. Some algorithms might explore certain descriptions more effectively than others, for example, because the objective functions describing the lighting and structural behavior of a certain design may have completely different properties. In an attempt to exploit this property, BPO practitioners often dedicate a small amount of their total time budget to test various algorithms and different setup parameters, before finally settling for an optimization algorithm [Hamdy et al., 2016].

Regarding the different algorithms' categories, it is interesting to see the metaheuristics' popularity among researchers and practitioners. The main reasons behind the idolization of the metaheuristics are their: (1) inherent simplicity, (2) ease of implementation, and (3) wide applicability to different domains [Wortmann and Nannicini, 2017]. Unfortunately, other categories do not benefit from such properties, which is a limitation towards their application in architectural domains. For architecture, the major limitations to other categories lie in the absence of easy-to-use tools. Firstly, these tools are usually available as programming frameworks instead of integrated in architectural design workflows. As a result, to use the framework's optimization algorithms, architects often need some programming knowledge to create the scripts to integrate the algorithms into the design workflow. Unfortunately, since architects typically lack the required knowledge, they tend to struggle with the scripts production and, eventually, opt for using friendlier metaheuristics ready-to-use tools. Given this facts, it is not surprising that most of the existing building design optimization literature ends up focusing on the application of algorithms from the metaheuristics category [Hamdy et al., 2016, Nguyen et al., 2014, Evins, 2013].

check if any other properties are relevant, for example in wortmann's works

However, in the light of the NFLTs, the need for more short-term efficient optimization approaches fostered the development of tools exposing algorithms with different properties. Particularly, the plugins Goat [Simon Flöry, 2019] and Opossum [Wortmann, 2017b] enable the usage of algorithms from both direct search and model based classes. These tools expose optimization algorithms from the NLopt [Steven G. Johnson, 2010] and RBFOpt [Costa and Nannicini, 2014] frameworks, respectively,

providing friendly, ready-to-use interfaces within Grasshopper [David G. Rutten, 2007], a visual programming environment that enables the parametric design and performance evaluation of building designs for different values of the parameters. For the past few years, few works have compared different algorithms using these tools with the ones available in metaheuristics tools, such as Galapagos [David G. Rutten, 2010], Octopus [Vierlinger, 2013], Optimo [Zarrinmehr and Yan, 2015], Silvereye [Cichocka et al., 2017b], among others.

Although the results may vary, in general, direct search and surrogate-based algorithms seem to be more effective than the metaheuristics ones in initial stages of the optimization process [Wortmann, 2017a, Wortmann and Nannicini, 2016, Wortmann et al., 2017]. Even some metaheuristics algorithms can be very effective approaches for some optimization problems [Waibel et al., 2018]. One can explore these performance fluctuations to find the most effective optimization algorithm for a specific problem. This performance gain can be determining in the overall optimization time, especially, when complex and time-consuming simulations are involved. Indeed, several authors [Wortmann and Nannicini, 2016, Hamdy et al., 2016] suggest that the selection of the optimization algorithm should be based on the results of several tests with different methods for a fixed number of evaluations or a fixed amount of time.

Optimization is a useful tool to address both single and multi-objective problems. In architecture, most optimization applications focus on single-objective problems and cover the three different derivative-free algorithms classes. However, the same does not happen with multi-objective problems, with only one of the classes being extensively applied to MOO building design: the metaheuristics [Hamdy et al., 2016]. The main reason behind metaheuristics popularity is their broader adaptability to both varying degrees of complexity and to different problem domains [Blum and Roli, 2003].

Recent developments in multiple surrogate-assisted Multi-Objective Evolutionary Algorithms (MOEAs) in the fields of science and engineering [Zapotecas-Martínez and Coello, 2016, Hussein and Deb, 2016] made it possible to decrease the number of expensive evaluations in MOO problems. Generally, these techniques combine metaheuristics methods, which find more than one solution within a single execution, with surrogate models, which are approximations of the original objective functions. Diaz-Manriquez et al. [Díaz-Manríquez et al., 2016] provide a comprehensive overview of surrogate-assisted techniques for MOO from the engineering perspective.

The following sections focus on the current BPO practices both for SOO and MOO, the currently available tools, the advantages and disadvantages of each approach.

2.2 Single-Objective Optimization

SOO processes aim to find the best solution with respect to a unique objective function. This function is described in terms of the values of the problem's parameters. Equation (2.1) illustrates an example

of a mathematical unconstrained minimization SOO problem, where f represents the single objective function and x represents the problem's parameters'.

$$\min f(x) \tag{2.1}$$

Generally, the computational complexity of optimization processes is exponential on the number of objectives and, consequently, the more objectives, the more expensive these processes are. In particular, SOO processes rely exclusively on a single objective function and, therefore, are usually less time-consuming than the MOO processes. The gains in computational resources become particularly relevant when considering simulation-based objective functions. For instance, in the case of building design, most problems include simulation-based objective functions. As a result, architects commonly opt for SOO processes: either by simply considering a single objective, or, in the case of problems involving multiple conflicting objectives, by combining them in a unique function as it will be further explained in section 2.3.2.

A literature review over the architectural practice will evidence the prevalence of SOO algorithms. Firstly, single-objective problems are easier to model. Secondly, plug-ins, like Galapagos and Goat, allow to address single objective building design problems, hence enabling to solve simpler optimization problems and reducing the total complexity of optimization processes. Finally, these plug-ins expose different derivative-free optimization algorithms, enabling the selection of the best algorithm to specific problems and potentially improving optimization processes' complexity [Wortmann and Nannicini, 2016].

Despite its lower optimization time, for multi-objective problems, these processes are also usually less informative than the MOO ones. Particularly, in the architectural practice, building design often simultaneously involves different conflicting aspects, such as thermal, energy consumption, structural, cost, among others. In such situations, one is interested in obtaining a view over the compromises between conflicting aspects in order to make a informed decision.

2.3 Multi-Objective Optimization

MOO belongs to the set of problems concerned with the optimization of more than one objective simultaneously. The addition of other, potentially conflicting, objectives to the optimization process requires the re-definition of *optimality*.

While in SOO problems we expect the optimal solution to be the set of parameters that achieve the best[1] objective value possible, in multi-objective the best possible configuration for one of the objectives is rarely the best possible configuration for all other objectives as well, which results from the fact that

[1] For simplification purposes we simply refer to the optimal solution as being the best. When dealing with a minimization problem, the best solution is the one that achieves the lowest value of the objective function, whereas in maximization problems, the best solution is the one achieving the highest value of the objective function.

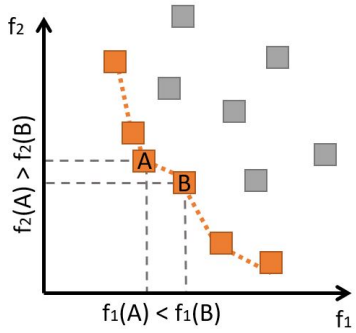


Figure 2.2: Representation of the set of non-dominated (orange squares) and dominated (gray squares) solutions for a two-objective minimization problem. The Pareto front is composed of all the non-dominated solutions.

these objectives are often contradictory. In order to be able to compare different multi-objective solutions, these problems are often addressed considering the Pareto optimality (or Pareto Efficiency) concept. This concept, named after the economist Vilfredo Pareto, defines an optimal solution as being a solution for which it is impossible to improve an objective value without deteriorating others. Such a solution is also said to be non-dominated or noninferior, and the set of non-dominated solutions is called the Pareto Frontier or Pareto Front. An example of a two-objective minimization problem is illustrated in Figure 2.2. The two objectives are f_1 and f_2 and the solutions shown in orange are non-dominated. The goal of optimization algorithms is to find, in the search space, design solutions that lie on the Pareto Front.

Building design is a complex task that frequently involves dealing with multiple conflicting objectives, such as maximum lighting comfort versus maximum thermal comfort, or minimum energy consumption versus maximum thermal comfort. The architect might apply follow different approaches depending on his knowledge and on his level of expertise. The following three sections describe the benefits and limitations of each approach.

2.3.1 Experimental Approach

The experimental or design of experiments approach is widely used in research and in practice to address both single and multi-objective problems [Fang, 2017]. Besides being intuitive and flexible, it can achieve a potentially better solution without having to deal with complex optimization algorithms. In fact, these processes typically use sampling methods, such as Full-factorial, Monte Carlo Sampling, and Latin Hypercube Sampling [Giunta et al., 2003] to generate different combinations of the design parameters, hence producing multiple design alternatives to be evaluated. One advantage of having multiple solutions instead of just one is that the choice of the best solutions can be exclusively relegated to the architect, who analyses all the generated variations, and subsequently decides on the solutions he thinks are the best (e.g., using preference articulation, pondering the conflicting criteria).

Unfortunately this approach does not guarantee that good solutions will be found. In fact, in most cases input configurations are generated a priori and do not take advantage of any additional information. Consequently, the performance of previously evaluated design candidates is not used to guide the search towards the most efficient designs. Concretely, this means that because the process does not use the information of previous design variations, useless candidate designs can be sometimes evaluated.

On the other hand, given its simplicity and flexibility, this approach allows to easily assemble more complex processes which are capable of directing the search towards better design variations. For example, consider an optimization problem where an architect selects a sampling method to generate different design variations, which are then evaluated. After analysing the results, the architect may wish to explore design regions around the most promising solutions. In that case, he must limit the design variations to lie within the promising regions, by updating the problem's definition. The redefined problem is then subsequently sampled and redefined until the architect is satisfied with the quality of the obtained solutions.

Despite the constant need for manual intervention, the previous technique can be adapted to extract information about the design problem itself. Using an automated version of this technique, we are able to study the behavior of the different objective functions, when the input variables are changed. This process is commonly known as *sensitivity analysis* [Saltelli et al., 2007] and has been applied to multiple problems in the context of building design optimization [Tian, 2013], either to enhance the performance of existing optimization algorithms, or to achieve better solutions.

Overall, while it does not provide guarantees on the solutions' optimality, this approach is simple, examples?? easy to use, and it is available in numerous tools and frameworks. Moreover, although this process is not intelligent *per si*, since the decisions are always made by the architect, it enables a more intelligent and informed design process, as it provides the full disclosure of the generated design variations, as well as their associated performance.

2.3.2 A Priori Articulation Approach

This approach allows combining multiple objectives according to one's preferences using an utility function [Marler and Arora, 2004]. Among all the possible utility functions, the most commonly used is the weighted sum or linear scalarization [Wortmann, 2017b]. This utility function reduces multi to single-objective problems by defining the objective functions to be the weighted sum of multiple objectives. The weights (or coefficients) represent the relative importance of each objective to the architect. These must be defined before the optimization and, in general, they require expertise and experience, as this optimization approach is highly sensitive to the selected coefficients and that a different articulation of preferences might yield completely different results. The final objective function is then provided to a

SOO algorithm, which tries to find an optimal (or a near optimal) solution. Equation (2.2) represents an unconstrained example of the mathematical definition of such approach, where w_i is the coefficient associated to the objective f_i, and n is the total number of objectives of the problem.

$$\min_{x \in X} \sum_{i=1}^{n} w_i f_i(x) \tag{2.2}$$

In architecture, the ease of use, the availability, the heterogeneity of ready-to-use SOO tools (e.g., Opossum, Goat, Galapagos, Silvereye), as well as the overall time of a priori articulation processes are virtues of this approach when compared to other MOO approaches. In general, because this preference-based approach focus in the retrieval of a single optimal solution that satisfies the previously defined preferences, it becomes more effective and less time consuming than other MOO approaches, that either lack a more guided search or that aim at finding solutions that are optimal under different preference articulations.

Overall, the *a priori* articulation approach enables a more intelligent design process because there exists an algorithm that exploits the knowledge about the previously evaluated solutions to guide the search towards optimal regions of the design space. However, because these algorithms are usually autonomous and independent of external manual interventions, architects are not usually integrated in the optimization loop, thus losing control over the design optimization process. Moreover, most of these algorithms retrieve a single optimum and provide no other design options. This is a drawback [Cichocka et al., 2017a], as the architect either complies to the retrieved solution or he must rerun the optimization with another articulation of preferences. Either way, the optimization no longer provides enough information to make informed decisions.

2.3.3 Pareto-based Approach

A more informative approach consists in the retrieval of a diverse and potentially heterogeneous set of Pareto-optimal solutions. When confronted with this set of optimal solutions, architects can compare different design options according to different performance criteria and make informed decisions about the compromises taken. Equation (2.3) is an example of a mathematical unconstrained minimization Pareto-based MOO problem, where $F(x)$ represents the vector of the k objectives, and f_i represent the objective function i.

$$\min \{F(x) = [f_1(x), f_2(x), ..., f_k(x)]\} \tag{2.3}$$

When compared to previous approaches, one must consider (1) the number of function evaluations, which is larger due to the need to find a set of optimal solutions instead of focusing in a single one, (2) the visual representation of the solution space, which becomes problematic when the number of objectives

is greater than three, and (3) the modeling of the optimization problem itself, which directly impacts the quality of the solutions.

Notwithstanding the considerations above, a few studies concerning Pareto-based Optimization have emerged in the past years [Evins, 2013, Hamdy et al., 2016]. Indeed, the utility of Pareto optimization approaches has been recognized by architects as a useful decision support tool to building design [Cichocka et al., 2017a]. Moreover, recent surveys show that even though scalarization approaches are are less time-consuming, they are not as desirable as Pareto optimization, since they do not aid in the decision-making process, nor do they provide a clear trade-off between the different objectives involved [Attia et al., 2013, Cichocka et al., 2017a]. Despite the growing trend of Pareto optimization approaches [Evins, 2013], the lack of relevant benchmarks comparing the performance of different Multi-Objective Optimization Algorithms (MOOAs) in architecture is evident, which justifies a comprehensive comparison among the performance of several algorithms applied to different problems in the architectural context.

2.4 Quantitative Performance Indicators

OMG
AINDA
FALTA
TANTO

The performance of each optimization algorithm is very

In architectural design, the performance of SOO algorithms is evaluated according to the number of necessary evaluations to reach a good solution and the solutions‘ optimality. However, when shifting to a multi-objective context, there is a lack of consensus regarding the criteria for evaluating the algorithm’s results. A literature review evidences the existing struggle to define a single metric to accurately represent the quality of MOOAs’ results, i.e., of the approximated Pareto Fronts (Knowles et Corne 2002, Riquelme et al. 2015). Nevertheless, there are three properties of the results that should be considered when evaluating their quality (Zitzler et al. 2000, Riquelme et al. 2015): (1) accuracy, meaning that solutions should be as close as possible to the true Pareto Front, (2) diversity, meaning that solutions should be as uniformly distributed along the true Pareto Front as possible, so as to obtain a representative and diverse set of design variants covering to larger extents the different trade-offs, and (3) cardinality, meaning larger sets of solutions.

2.5 Optimization Tools in Architecture

For years, multiple optimization frameworks have been developed. However, in order to be used within architectural practices, architects had to code the integration scripts to connect the simulation models for different variations of the parametric models and the optimization frameworks [Attia et al., 2013]. Moreover, these frameworks often lacked post-processing and visual features which antagonized the

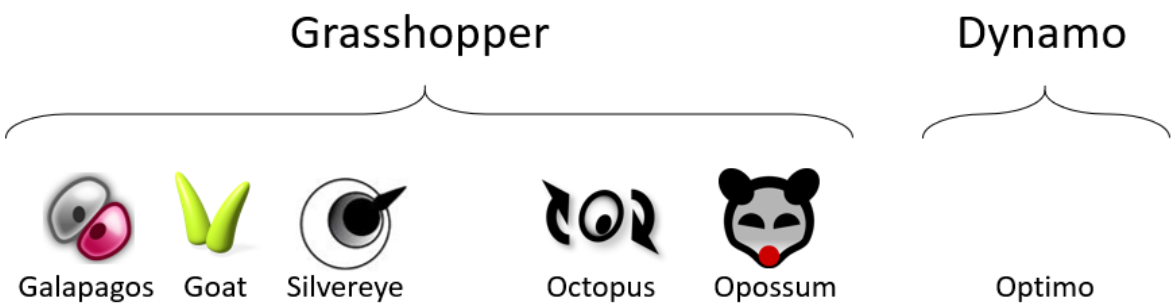


Figure 2.3: Optimization frameworks currently used in architectural practices.

readability and comprehension of the results [Attia et al., 2013, Nguyen et al., 2014].

Several plug-ins have been developed in an attempt to reduce the limitations associated to the coupling of simulation tools and mathematical optimization frameworks, thus providing a seamless connection between the parametric models produced in computational design tools, like CAD and BIM, the simulation or analytical models, and the optimization frameworks. Given the visual nature of architects, these plug-ins provide friendly, ready-to-use optimization interfaces, usually coupled with a few post-processing and visual features to enhance the intelligibility of the optimization results.

Currently, most optimization plug-ins are implemented on top of the visual parametric tools [Cichocka et al., 2017a] Grasshopper [David G. Rutten, 2007] and Dynamo [Ian Keough, 2011]. In fig. 2.3, we represent the most relevant optimization plug-ins among BPO practitioners: Galapagos, Goat, Octopus, Opossum, and Silvereye implemented on top of Grasshopper, and Optimo implemented on top of Dynamo. In the following sections we briefly discuss each plug-in.

2.5.1 Galapagos

Galapagos is a well-known metaheuristic-based optimization plug-in developed for designers and implemented on top of Grasshopper. David Rutten developed Galapagos, a generic platform designed to allow the application of metaheuristics algorithms by non-programmers to solve a wide variety of problems. Moreover, to enable its usage by non-experts, the optimization solver assumes default values for the different algorithms.

Galapagos provides two metaheuristics algorithms, namely, the genetic algorithm, inspired by biological evolution processes, and the simulated annealing, motivated by the metallurgical process of annealing [Brownlee, 2011]. Although both algorithms are basis for large variety of extensions and/or specializations, supporting different heuristics, we will not provide a full description of such extensions, instead referring the interested reader proper literature.

As evolutionary algorithms, genetic algorithms also explore Darwinian natural selection concepts, such as heredity, reproduction, and natural selection, and genetics concepts and mechanisms, includ-

ing genes, chromosomes, recombination, crossover, and mutation, in order to search for better solutions in the solution space. More concretely, genetic algorithms generate an initial random set of solutions, called population, which is then iteratively evolved, creating new 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) selection, where pairs of individuals are selected for reproduction, based on a probabilistic function which is proportional to each individual's fitness value, (3) crossover, where the genotypes 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 earlier generations are usually diverse, final generations are often very similar to the fittest individuals, i.e., we observe an intensification of the traits of the most suitable individuals, thus emulating the mechanism of natural selection, described by Darwin [Brownlee, 2011]. Besides genetic algorithms [Golberg, 1989, Holland, 1992], evolutionary algorithms encompass other algorithms such as Genetic Programming [Koza, 1992], Evolution Strategies [Schwefel, 1981], Differential Evolution [Storn and Price, 1997], among others.

Besides genetic algorithms, Galapagos also provides a metaheuristics global optimization physical algorithm, the simulated annealing algorithm. Resemblant of hill climbing algorithms, where new candidate solutions are randomly sampled, this algorithm iteratively re-samples the solution space aiming at finding an optimal solution. During the search, the algorithm is propitious to accept the re-sampled solutions with lower performance, according to a probabilistic function that becomes more discerning of the quality of the samples over the execution of the algorithm, thus resembling the natural annealing process [Brownlee, 2011].

To use one of Galapagos' optimization solvers, architects must define a Grasshopper's script in three parts: (1) Input, where he specifies the his design's parameters, (2) Generation, where he creates his design's algorithmic model, that when instantiated with the parameters' values will generate the 3D model, and (3) Analysis, where he defines the analysis function or objective function which he ought to optimize. To that end, the architect uses Grasshopper's components, such as *Sliders*, values lists, area, distance, and other analytical-like components to define the program. In order to use Galapagos, optimization variables must be defined in terms of *Sliders*, which enable the specification of numerical ranges with both lower and upper bounds, whereas the value of the objective function must be a Number component. Both the *Sliders* and the *Number* (*Num*) components connect to the input entry and to the output entry of the Galapagos' component, i.e., the genome and the fitness entries, respectively.

After defining the problem, the architect double clicks the Galapagos component and is presented with its Graphical User Interface (GUI) (see fig. 2.4). This interface is friendlier, simpler, and easier to use than previous approaches, requiring no integration efforts, nor any programming or algorithm's knowledge in order to setup and use it. For all these reasons, Galapagos is particularly popular amongst

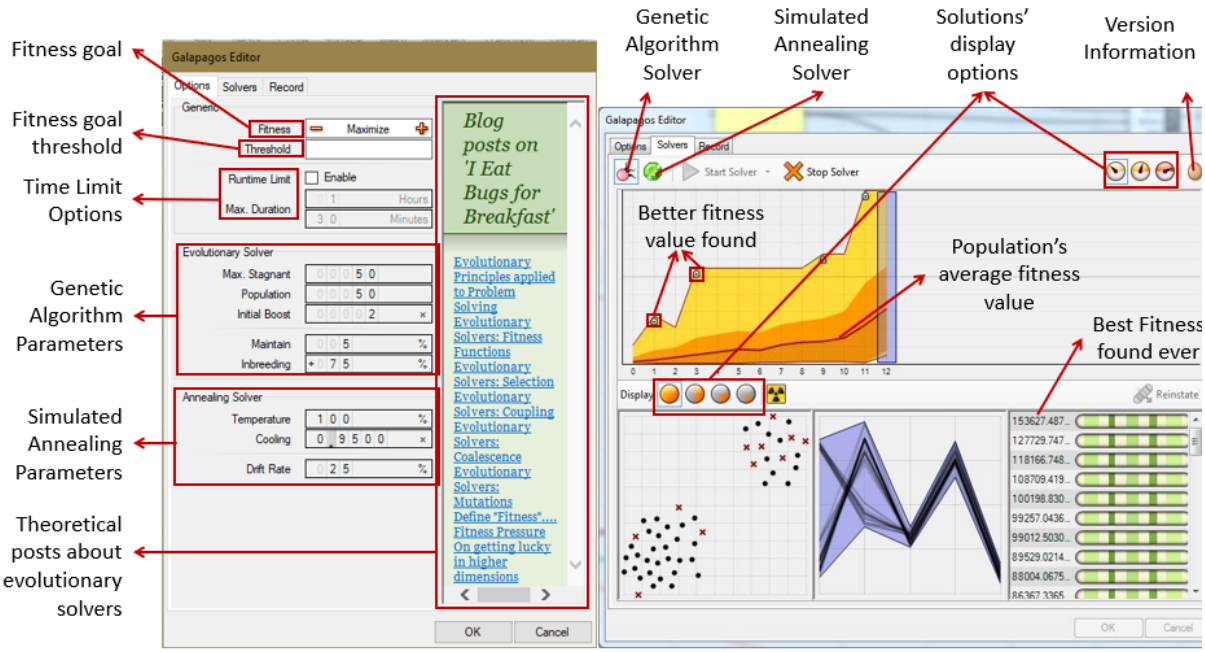


Figure 2.4: Galapagos' default setup menus

architects [Wortmann and Nannicini, 2017], that not only satisfies the visual nature of architects, but also supports four graphical views reporting the evolution of the optimization process: (1) line plot, representing different fitness measures per generation, (2) similarity representation view, where genetically similar solutions are closer to each other and where solutions contributing to the next offspring and solutions that do not contribute are represented with a black dot and a red cross, respectively, (3) parallel coordinates view, where solutions are represented as connected line segments and each vertical line corresponds to a parameter, (4) ranked list of the best solutions per generation. Although Galapagos views exhibit all the results of the most recent generation and highlights its best solution, it also provides the user with the flexibility to select multiple generations and to reinstate a particular solution, allowing the materialization of this design solution in the corresponding Grasshopper's script.

Post-processing features.

Its friendly interface and ready-to-use format facilitate its usage by architects.

2.5.2 Goat

Goat is an optimization plugin integrated into Grasshopper that was developed by Simon Flory.

Focus on single-objective optimization, but can also be used as a form of a priori preference articulation. It exposes the derivative-free algorithms existing in the NLopt framework in a friendlier and graphical interface.

2.5.3 Octopus

2.5.4 Opossum

2.5.5 Optimo

3

Solution

Contents

Donec gravida posuere arcu. Nulla facilisi. Phasellus imperdiet. Vestibulum at metus. Integer euismod. Nullam placerat rhoncus sapien. Ut euismod. Praesent libero. Morbi pellentesque libero sit amet ante. Maecenas tellus. Maecenas erat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

3.1 Architecture Overview

3.2 Architecture Design Requirements

3.2.1 Problem Modelling

3.2.2 Simple Solver

3.2.3 Meta Solver

3.3 Architecture Design Implementation

3.3.1 Problem Modelling

3.3.2 Simple Solver

3.3.3 Meta Solver

4

Evaluation

Contents

- Relembrar o objectivo do trabalho e dizer como o vamos avaliar de um modo geral introduzindo os proximos subcapitulos.

4.1 Qualitative Evaluation

- Number and Heterogeneity of Available algorithms - Differences / Benefits / Disadvantages when compared to Grasshopper's frameworks

4.2 Quantitative of Applications

- Dizer que de um modo geral começámos de forma incremental por considerar problemas single-objective, nomeadamente a casa da ericeira, que remonta a primeira publicação. Depois evoluimos para a avaliação bi-objetivo de dois casos de estudo reais - Pavilhão Preto para exposições e de uma arc-shaped space frame.

- Comentar a facilidade c/ que alguém que já tem um programa AD consegue acopolar optimização a AD.

4.2.1 Ericeira House: Solarium

4.2.2 Black Pavilion: Arts Exhibit

4.2.2.A Skylights Optimization

4.2.2.B Arc-shaped Space Frame Optimization

5

Conclusion

Contents

Pellentesque vel dui sed orci faucibus iaculis. Suspendisse dictum magna id purus tincidunt rutrum. Nulla congue. Vivamus sit amet lorem posuere dui vulputate ornare. Phasellus mattis sollicitudin ligula. Duis dignissim felis et urna. Integer adipiscing congue metus.

Rui Cruz
You should always start a Chapter with an introductory text

5.1 Conclusions

5.2 System Limitations and Future Work

5.2.1 Optimization Algorithms

5.2.2 ML models

5.2.3 Constrained Optimization

Aliquam aliquet, est a ullamcorper condimentum, tellus nulla fringilla elit, a iaculis nulla turpis sed wisi. Fusce volutpat. Etiam sodales ante id nunc. Proin ornare dignissim lacus. Nunc porttitor nunc a sem. Sed sollicitudin velit eu magna. Aliquam erat volutpat. Vivamus ornare est non wisi. Proin vel quam. Vivamus egestas. Nunc tempor diam vehicula mauris. Nullam sapien eros, facilisis vel, eleifend non, auctor dapibus, pede.

Bibliography

[Aguiar et al., 2017] Aguiar, R., Cardoso, C., and Leitão, A. (2017). Algorithmic Design and Analysis Fusing Disciplines. pages 28–37.

[Ashour, 2015] Ashour, Y. S. E.-D. (2015). Optimizing Creatively in Multi-Objective Optimization.

[Attia et al., 2013] Attia, S., Hamdy, M., O'Brien, W., and Carlucci, S. (2013). Computational optimisation for zero energy building design : Interviews results with twenty eight international experts. *13th Conference of International Building Performance Simulation Association*, pages 3698–3705.

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, 35(3):189–213.

[Branco and Leitão, 2017] Branco, R. C. and Leitão, A. M. (2017). Integrated Algorithmic Design: A single-script approach for multiple design tasks. *Proceedings of the 35th Education and research in Computer Aided Architectural Design in Europe Conference (eCAADe)*, 1:729–738.

[Brownlee, 2011] Brownlee, J. (2011). *Clever Algorithms*.

[Cichocka et al., 2017a] Cichocka, J. M., Browne, W. N., and Rodriguez, E. (2017a). Optimization in the architectural practice. *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2017,*, pages 387–397.

[Cichocka et al., 2017b] Cichocka, J. M., Migalska, A., Browne, W. N., and Rodriguez, E. (2017b). SILVEREYE – The Implementation of Particle Swarm Optimization Algorithm in a Design Optimization Tool. 724:151–169.

[Conn et al., 2009] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2009). *Introduction to Derivative-Free Optimization*.

[Costa and Nannicini, 2014] Costa, A. and Nannicini, G. (2014). RBFOpt : an open-source library for black-box optimization with costly function evaluations. *Optimization online no.4538*.

[David G. Rutten, 2007] David G. Rutten (2007). Grasshopper: Generative Modeling for Rhino.

[David G. Rutten, 2010] David G. Rutten (2010). Galapagos - an optimization solver component.

[De Kestelier and Peters, 2013] De Kestelier, X. and Peters, B. (2013). *Computation Works: The Building of Algorithmic Thought*, volume Computation Works: The Building of Algorithmic Thought.

[Díaz-Manríquez et al., 2016] Díaz-Manríquez, A., Toscano, G., Barron-Zambrano, J. H., and Tello-Leal, E. (2016). A review of surrogate assisted multiobjective evolutionary algorithms. *Computational Intelligence and Neuroscience*, 2016.

[Evins, 2013] Evins, R. (2013). A review of computational optimisation methods applied to sustainable building design. *Renewable and Sustainable Energy Reviews*, 22:230–245.

[Evins et al., 2012] Evins, R., Joyce, S. C., Pointer, P., Sharma, S., Vaidyanathan, R., and Williams, C. (2012). Multi-objective design optimisation: getting more for less. *Proceedings of the Institution of Civil Engineers - Civil Engineering*, 165(5):5–10.

[Evins and Vaidyanathan, 2011] Evins, R. and Vaidyanathan, R. (2011). Multi-objective optimisation of the configuration and control of a double-skin facade. In *12th Conference of International Building Performance Simulation Association, Sydney*, number November, Sydney.

[Fang, 2017] Fang, Y. (2017). Optimization of Daylighting and Energy Performance Using Parametric Design, Simulation Modeling, and Genetic Algorithms.

[Ferreira and António, 2015] Ferreira, B. and António, L. (2015). Generative Design for Building Information Modeling. *Proceedings of the 33rd Education and research in Computer Aided Architectural Design in Europe Conference*, 1:635–644.

[Forrester and Keane, 2009] Forrester, A. I. J. and Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1-3):50–79.

[Giunta et al., 2003] Giunta, A., Wojtkiewicz, S., and Eldred, M. (2003). Overview of modern design of experiments methods for computational simulations. *Aiaa*, 649(July 2014):6–9.

[Glover and Kochenberger, 2003] Glover, F. and Kochenberger, G. A. (2003). *Handbook of Metaheuristics*, volume 53.

[Golberg, 1989] Golberg, D. E. (1989). *Genetic Algorithms in Search Optimization & Machine Learning*. Addison-Wesley Longman Publishing Co.

[Hamdy et al., 2016] Hamdy, M., Nguyen, A.-t., and Hensen, J. L. M. (2016). A performance comparison of multi-objective optimization algorithms for solving nearly-zero-energy-building design problems.

[Hasançebi et al., 2009] Hasançebi, O., Çarbaş, S., Doğan, E., Erdal, F., and Saka, M. P. (2009). Performance evaluation of metaheuristic search techniques in the optimum design of real size pin jointed structures. *Computers and Structures*, 87(5-6):284–302.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

[Hooke and Jeeves, 1961] Hooke, R. and Jeeves, T. A. (1961). " Direct Search" Solution of Numerical and Statistical Problems. *Journal of the ACM*, 8(2):212–229.

[Hussein and Deb, 2016] Hussein, R. and Deb, K. (2016). A Generative Kriging Surrogate Model for Constrained and Unconstrained Multi-objective Optimization. *Gecco*, pages 573–580.

[Ian Keough, 2011] Ian Keough (2011). Dynamo BIM.

[Jones et al., 1993] Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181.

[Kolda et al., 2003] Kolda, T. G., Lewis, R. M., and Torczon, V. (2003). Optimization by Direct Search - New Perspectives on Some Classical and Modern Methods. *Society for Industrial and Applied Mathematics*, 45(3):385–482.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[Koziel and Yang, 2011] Koziel, S. and Yang, X.-S. (2011). *Computational optimization, methods and algorithms*, volume 356.

[Law and Kelton, 1991] Law, A. M. and Kelton, W. D. (1991). *Simulation modeling and analysis*, volume 2.

[Leitão et al., 2014] Leitão, A., Santos, L., and Fernandes, R. (2014). Pushing the Envelope: Stretching the Limits of Generative Design. *Blucher Design Proceedings*, 1(7):235–238.

[Machairas et al., 2014] Machairas, V., Tsangrassoulis, A., and Axarli, K. (2014). Algorithms for optimization of building design: A review. *Renewable and Sustainable Energy Reviews*, 31(1364):101–112.

[Malkawi and Kolarevic, 2005] Malkawi, A. and Kolarevic, B. (2005). *Performative Architecture: Beyond Instrumentality*, volume 60.

[Marler and Arora, 2004] Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395.

[Nelder and Mead,] Nelder, J. and Mead, R. A simplex method for function minimization. pages 308–313.

[Nguyen et al., 2014] Nguyen, A.-T., Reiter, S., and Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy*, 113:1043–1058.

[Nocedal and Wright, 2011] Nocedal, J. and Wright, S. J. (2011). *Numerical optimization*. Number 2.

[Online, 2018] Online, M. W. (2018). Merriam Webster Online - Optimization Definition.

[Rios and Sahinidis, 2013] Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

[Rowan, 1990] Rowan, T. (1990). Functional stability analysis of numerical algorithms. *Unpuplished Dissertation*, page 218.

[Saltelli et al., 2007] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2007). *Global Sensitivity Analysis. The Primer*.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA.

[Simon Flöry, 2019] Simon Flöry (2019). Goat - an optimization solver component.

[Steven G. Johnson, 2010] Steven G. Johnson (2010). The NLopt nonlinear-optimization package.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

[Tian, 2013] Tian, W. (2013). A review of sensitivity analysis methods in building energy analysis. *Renewable and Sustainable Energy Reviews*, 20.

[Vierlinger, 2013] Vierlinger, R. (2013). *Multi Objective Design Interface*. PhD thesis.

[Waibel et al., 2018] Waibel, C., Wortmann, T., Evins, R., and Carmeliet, J. (2018). Building Energy Optimization: An Extensive Benchmark of Global Search Algorithms. *Energy and Buildings*, under revi(October).

[Wortmann, 2017a] Wortmann, T. (2017a). Model-based Optimization for Architectural Design : Optimizing Daylight and Glare in Grasshopper. *Technology — Architecture + Design*, 1(2):176–185.

[Wortmann, 2017b] Wortmann, T. (2017b). Opossum. *Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia*, pages 283–292.

[Wortmann et al., 2015] Wortmann, T., Costa, A., Nannicini, G., and Schroepfer, T. (2015). Advantages of surrogate models for architectural design optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 29(04):471–481.

[Wortmann and Nannicini, 2016] Wortmann, T. and Nannicini, G. (2016). Black-box optimization methods for architectural design. (April):177–186.

[Wortmann and Nannicini, 2017] Wortmann, T. and Nannicini, G. (2017). Introduction to Architectural Design Optimization. 125(December).

[Wortmann et al., 2017] Wortmann, T., Waibel, C., Nannicini, G., Evins, R., Schroepfer, T., and Carmeliet, J. (2017). Are Genetic Algorithms Really the Best Choice for Building Energy Optimization? (June):51–58.

[Zapotecas-Martínez and Coello, 2016] Zapotecas-Martínez, S. and Coello, C. A. (2016). MONSS: A multi-objective nonlinear simplex search approach. *Engineering Optimization*, 48(1):16–38.

[Zarrinmehr and Yan, 2015] Zarrinmehr, S. and Yan, W. (2015). Optimo : A BIM-based Multi-Objective Optimization Tool Utilizing Visual Programming for High Performance Building Design Optimo : A BIM-based Multi-Objective Optimization Tool Utilizing Visual Programming for High Performance. In *33rd eCAADe*.