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 Research Grants.

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, 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 livre de derivadas; Aprendizagem Máquina; Modelos baseados em aproximações.

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
CRS	Controlled Random Search
DIRECT	DIviding RECTangles
ER	Error Ratio
ES	Evolution Strategy
GA	Genetic Algorithm
GD	Generational Distance
GUI	Graphical User Interface
HJ	Hooke-Jeeves
HV	Hypervolume
HVAC	Heating, Ventilation, and Air Conditioning
HypE	Hypervolume Estimation Algorithm for Multi-Objective Optimization (MOO)
IGD	Inverted Generational Distance
ML	Machine Learning

MOEA Multi-Objective Evolutionary Algorithm

MPFE Maximum Pareto Front Error

MOO Multi-Objective Optimization

MOOA Multi-Objective Optimization Algorithm

NFLT No Free Lunch Theorem

NMS Nelder-Mead Simplex

NN Neural Network

NSGA-II Non-dominated Sorting Genetic Algorithm II

ONVG Overall Non-dominated Vector Generation

ONVGR Overall Non-dominated Vector Generation (ONVG) Ratio

PSO Particle-Swarm Optimization

RBF Radial Basis Function

RF Random Forest

SPEA2 Strength Pareto Evolutionary Algorithm 2

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 [Webster, 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 comprised of (a) variables or unknowns, i.e., representations of the system's characteristics, (b) objectives or criteria, i.e., quantitative measures of the system's performance and that are usually functions of the system's variables, and, optionally, (c) constraints, i.e., system's conditions that have to be satisfied to guarantee the system's feasibility [Nocedal and Wright, 2011]. Subsequent to model definition, we reach the second part of optimization process~~es~~, that is, to search among the set of possible solutions for the optimal ones. The strategy used to search the solution space is the responsibility of optimization algorithms, which enclose a detailed description of the steps necessary to attain optimal solutions. The optimal solutions depend on the values of the model's objectives, and the search directions depend on whether they should be maximized or minimized.

In the mathematical sense of optimization, these processes can be classified differently depending on the numerous alternatives for representing models or on the strategy underlying the search for optimal solutions. Concretely, models representations may differ in the variables' type, the presence ~~of~~ absence ~~or~~ constraints, the number and nature of objective functions, among others, whereas search strategies might explore vaster or narrower regions of the solution space. Although we introduce four of these classifications, we ~~refer~~ the interested reader ~~to~~ [Nocedal and Wright, 2011, Nemhauser and Wolsey, 1988] for a more comprehensive treatments of these subjects. We selected four classifications due to their relevance and their ubiquitous in optimization problems.

The first classification differentiates continuous and discrete optimization problems depending on the variables' types. Continuous optimization refers to problems defined uniquely by continuous variables and, therefore, characterized by an infinite solution space, whereas discrete optimization refers to problems for which some or all their variables are discrete, hence yielding a finite solution space. In continuous optimization, the smoothness of functions make it possible to reason about the behavior of all points close to x and, consequently, to solve these problems more easily, whilst the commonly present irregularities of discrete optimization functions do not, in general, allow to draw any information about the behavior of points close to x. Moreover, the discrete classification encloses finer classes, such as integer optimization or combinatorial optimization [Nemhauser and Wolsey, 1988].

The second classification is related to the absence or presence of constraints on the variables. Unconstrained optimization problems result from many practical applications and have no restrictions on the

values of the variables. Contrastingly, constrained optimization problems usually emerge from systems ~~for which~~ constraints are crucial (e.g., economy problem, imposing cargo constraints) and, therefore, incorporate such constraints into the problem's definition [Nocedal and Wright, 2011]. Variables can be conditioned using hard or soft constraints. The former sets conditions for the variables that must be satisfied, ~~i.e.,~~ to vary within simple bounds (e.g.: $-1 < x < 1$) and to relate to other variables in certain ways (e.g.: $\sum_i x_i$), whereas the latter sets penalties for the variables whose value violates the condition. Constrained optimization are often converted to unconstrained optimization problems by replacing hard constraints by soft constraints, i.e., by adding penalization terms in the objective function to discourage the violation of constraints.

The third classification distinguishes optimization problems in terms of ~~the aim of the search~~, particularly, whether it is a global or a local search. 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, global optimization problems strive to find the globally optimal solutions or, in other words, the best of all the local optima.

The fourth, and final, dichotomy herein discussed ~~is with respect to~~ the number of objectives to optimize, which are distinguished in single-objective and multi-objective optimization. Indeed, optimization is frequently required to address problems involving more than one objective. For example, people often face decisions involving two or more conflicting objectives, either to effectively manage resources, or just to ponder several factors associated with certain decisions (e.g.: purchase a house). As opposed to simpler single-objective optimization problems, which focus on the optimization of one objective, these processes are examples of Multi-Objective Optimization (MOO) problems, as they attempt to simultaneously optimize multiple conflicting objectives.

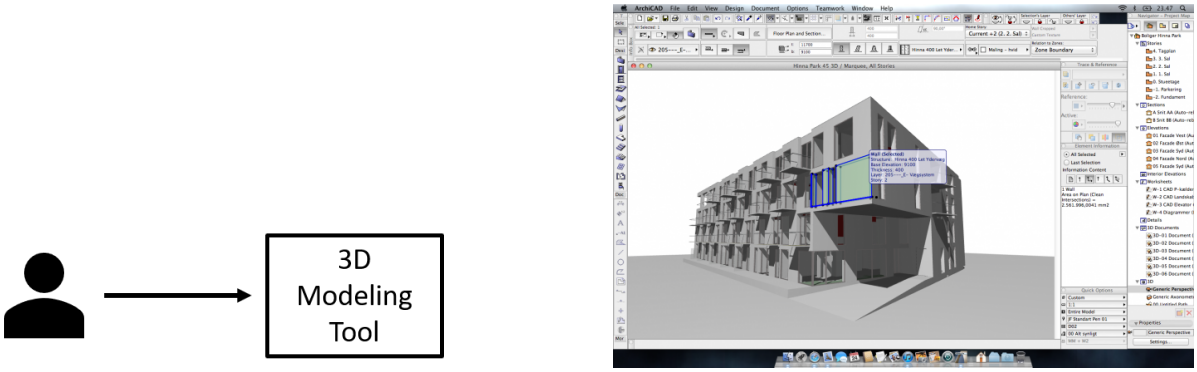
The application of optimization goes beyond day-to-day life decisions, also having a paramount impact on decisions ~~involved in~~ fields like economy, science, engineering, among others. As a case in point, optimization yields a great potential to ~~architectural practices, for~~ it directly impacts the building industry: optimization enables the reduction of the economic and ecological footprint of the building sector through the finding of more efficient building ~~variants,~~ prior to 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. The following sections provide an overview ~~of~~ the involvement and ~~the~~ evolution of these processes in the architectural field. We end this chapter by highlighting our research goals and ~~by~~ outlining this thesis structure.

1.1 From design to Optimized design

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

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 dissemination of digital modelling tools, which allowed for 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 Leitão, 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 realize his design ideas.



(a) **(b)**

Figure 1.1: (a) Simplification of a computer-aided design workflow (b) Building design example in a 3D modeling tool. Image retrieved from [Jared Banks, 2012]

Shortly after, the development of computer-based simulation tools enabled designers to simulate the behavior of building designs regarding specific criteria, i.e., to 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, which spurred a new performance-based approach.

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 (e.g., using a single plane *versus* a mesh to represent ~~a~~ non-planar surface), both the computational time and ~~the~~ result of the simulation might change.

In architecture, the generation of ~~each~~ analytical model is a time-consuming and complex 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. Particularly, classical gradient-based optimization methods cannot be used be-

cause they exploit the function's derivatives. 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 the experimentation of 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 such approach is Algorithmic Design (AD) [Branco and Leitão, 2017] and Figure 1.2 illustrates a simplified view of its application in a design workflow. 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].

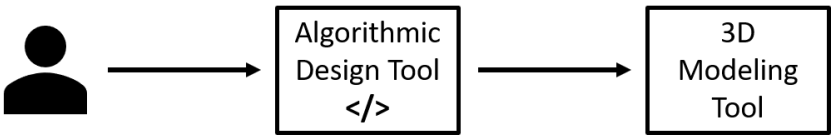
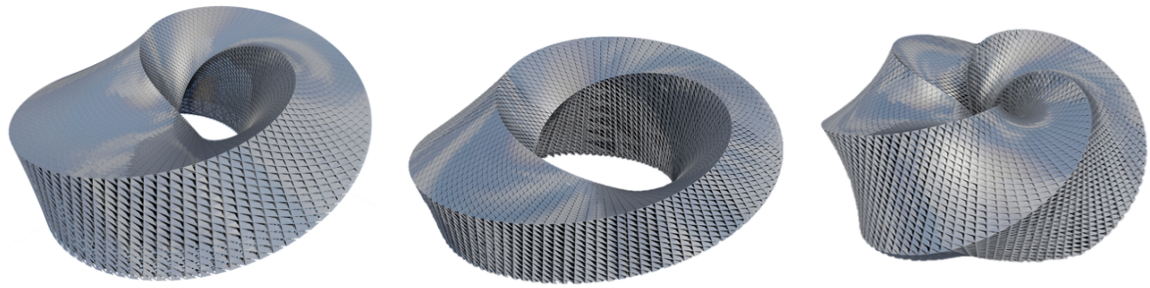


Figure 1.2: Algorithmic-based design workflow

As an example, consider the algorithmic design of Astana's National Library from the Bjarke Ingels Group (BIG) architects, illustrated in Figure 1.3(a). Initially, the architect selects an AD tool providing the necessary design primitives. Then, he uses those 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 same modification is propagated throughout that element's dependencies. In the end, the architect creates a procedure responsible for creating the whole design, which when executed will produce the corresponding Astana 3D model. Because the Astana's design resembles a *möbius* strip, define the procedure in terms of two parameters: the radius, defining the whole width of the design, and the number of twists of the strip. Now, he is able to easily generate different variations of the Astana by invoking the procedure with different values for these parameters. Figure 1.3 illustrates the original design, two design variations, where the radius and the number of turns in the design are increased, respectively.



(a) (b) (c)

Figure 1.3: Astana's National Library Design variations: (a) Original (b) Larger diameter (c) Two *möbius* twists

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 architects to learn how to program. Despite these investments, the benefits obtained with algorithmic approaches surpass the ones obtained when using CAD or BIM tools directly for the design of complex buildings. Particularly, 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 optimize 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 makes 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, including: (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 errors or loss of information; (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.

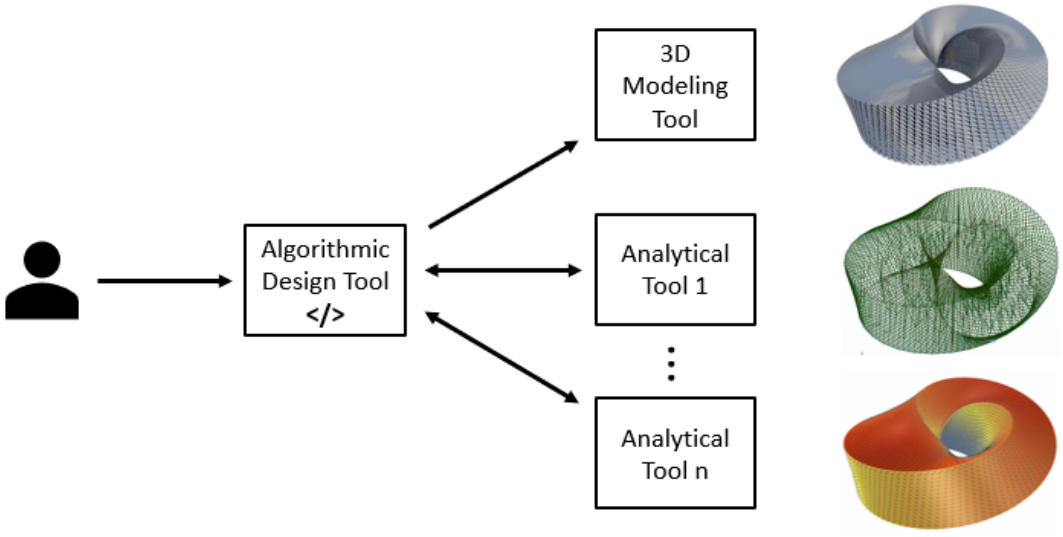


Figure 1.4: Algorithmic Design and Analysis design workflow with examples of the Astana's National Library design and analytical models: (top) 3D model, (center) Robot's structural analysis model; (bottom) Radiance's pos-radiation analysis model.

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~~.~~ ~~D~~ue to the tools' specificity, their integration was often an obstacle to the optimization process itself. More recently, the emergence of ~~visual~~ ~~parametric~~ approaches, such as Grasshopper~~,~~ Dynamo [David G. Rutten, 2007, Ian Keough, 2011]~~, among others,~~ ~~coupled~~ with the growing consciousness of both the limitations and ~~the~~ benefits of optimizati~~on 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 restricting its optimization capabilities. COLOCAR REF

On the other hand, ~~textual~~ ~~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 model~~s~~ 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, regarding the quality of a design solution, allows architects to make more informed decisions. With these explanations, the architect can not only provide valuable arguments for its implementation, 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 archi-

tecture. 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 comprehensive feedback over the optimization results.

To achieve the goals ~~that we~~ propose, 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 base requirements for the implementation of a simpler framework and its seamless application within the architectural practice.

1.3 Organization of the Document

The next chapters of this thesis are organized as follows:

Chapter 2 presents an overview of the current 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 framework~~s~~.

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 the algorithms' different properties and search strategies. Applying the adequate algorithm to a certain problem may potentially increase the efficiency of an optimization process.

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 initially considered aspects.

Finally, currently available architectural design optimization tools explore the parametric models produced in visual programming environments, such as Grasshopper and Dynamo. These visual programming environments are implemented as plug-ins, which are tightly integrated with CAD and BIM tools, respectively. As a result, the connection between optimization tools and visual design workflows becomes seamless and friendlier. Additionally, these optimization tools usually expose a *ready-to-run* interface, which is very appealing 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, for optimization problems explicitly defined through mathematical formulations, algorithms that explore the information from the derivatives of such formulations to guide the search for optimal solutions are very efficient. These algorithms are referred to as classical gradient-based algorithms. However, when neither the mathematical form, nor the information about the derivatives is easily available, it becomes necessary to explore other classes of algorithms. Fortunately, derivative-free algorithms are remarkably suitable for addressing these problems, as they do not use information about the objective functions' derivatives to find optimal solutions, instead, they treat the objective functions as *black-boxes* and guide the search based on the result of previously evaluated solutions [Rios and Sahinidis, 2013].

In architecture, it is often impossible to attain a mathematical formulation for the objective functions, especially for complex designs. Alternatively, architects frequently use simulation tools as means to replace the closed-form mathematical expressions relating design's parameters to the objective functions [Wortmann and Nannicini, 2016]. As a consequence, information about the objective functions becomes difficult to attain, often requiring excessive amounts of resources. This lack of information prompts the need for algorithms that treat these functions as *black-boxes*. A simple approach is to

systematically 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].

Derivative-free algorithms allow to overcome the difficulty of deriving analytical forms that has has been rising with the increase of building design's complexity. [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, structural, lighting, thermal, energy consumption, and 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 due to 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, 1964], SUBPLEX [Rowan, 1990], DIviding RECTangles (DIRECT) [Jones et al., 1993], among others.

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. The "'meta" classification of these algorithms results from the fact that they control the heuristics applied in the process.

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 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 Genetic Algorithm (GA), 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.

2.1.3 Model-based Algorithms

Model-based algorithms are effective handlers for time-consuming problems, where sensitive information is expensive to collect [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 applicable 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 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

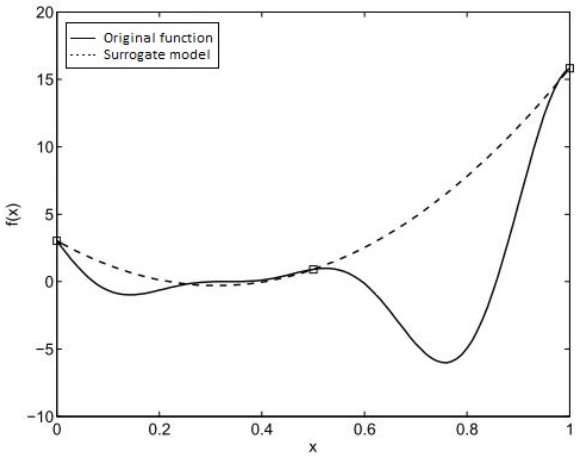


Figure 2.1: Original function and corresponding surrogate model, created based on three initial solutions (squares). This image was retrieved from [Koziel and Yang, 2011].

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, the lower availability and the lack of necessary technical knowledge to implement or incorporate these algorithms into optimization processes are still obstacles to a broader adoption of this approach. Notwithstanding the existence of different studies 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 Functions (RBFs), 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.

2.1.4 Comparison

This section considers the applicability of different classes of derivative-free optimization algorithms for architectural design. The multidisciplinary aspect of building design raises distinct problems, ranging from well-behaved problems with simple, unimodal, convex functions to more ill-behaved problems with irregular, multimodal objective 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 applied in architecture are not necessarily applicable to the 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, e.g., 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. Moreover, the lack of easy-to-use tools involving algorithms from other categories are also limiting their application in architectural contexts. Firstly, the existing non-metaheuristic tools are usually available as programming libraries, instead of being integrated in architectural design workflows. As a result, to use the optimization algorithms, architects often need some programming knowledge to create the scripts to integrate the algorithms into the design workflow. However, 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].

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, plug-ins like 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 other metaheuristics tools (e.g., Galapagos [David G. Rutten, 2010], Octopus [Vierlinger, 2013], Optimo [Zarrinmehr and Yan, 2015], Silvereye [Cichocka et al., 2017b]).

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 Single-Objective Optimization (SOO) and MOO, the currently available tools, and 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 vector of 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 easily 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 the optimization processes' complexity [Wortmann and Nannicini, 2016].

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

[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.

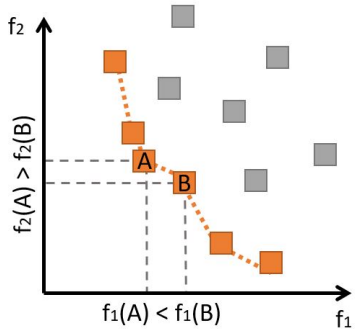


Figure 2.2: Representation of the set of non-dominated (orange squares) and dominated (gray squares) solutions for a two-objective minimization problem. The Pareto front is composed of all the non-dominated solutions.

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 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 Design of Experiments

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. This approach evaluates designs generated with different combinations of design parameters. These combinations can be obtained through sampling methods, such as Full-factorial, Monte Carlo Sampling, and Latin Hypercube Sampling [Giunta et al., 2003], which generate different combinations of design to be evaluated. This approach returns multiple design solutions instead of just one, leaving the final choice in the hands of the architect.

Unfortunately, this approach does not guarantee that good solutions will be found. In fact, in most cases, new solutions are generated without taking advantage of the information obtained from previously evaluated designs. Consequently, the old information is not used to guide the search towards the most efficient designs and useless candidate designs can sometimes be evaluated.

On the other hand, given its simplicity and flexibility, this approach allows architects to easily combine different processes in order to direct the search towards better design solutions. For example, the

architect can choose a sampling method to generate different design variations, which are then evaluated. After analyzing the results of the evaluations, the architect may wish to explore regions of the design space near the most promising design solutions. In that case, he might constrain 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 automatically extract information about the design problem itself, for example, to study the impact of design parameters in the performance. This process, commonly known as *sensitivity analysis* [Saltelli et al., 2007], has already been applied in the context of building design optimization [Tian, 2013], not only to achieve better solutions, but also to enhance the performance of existing optimization algorithms, e.g., by dropping irrelevant parameters.

Overall, while it does not provide guarantees about the solutions' optimality, this approach is simple, easy to use, and it is available in numerous tools. 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 presents all the design variations generated and their associated performance.

2.3.2 *A Priori* Articulation of Preferences

This approach allows combining multiple objectives according to one's preferences, using what is called a utility function [Marler and Arora, 2004], i.e., a function which ranks alternatives according to their utility for global performance. Among all the possible utility functions, the most commonly used is the weighted sum or linear scalarization [Wortmann, 2017b]. This function reduces multi-objective problems to single-objective ones by defining the objective function as the weighted sum of multiple objectives. The weights represent the relative importance of each objective to the architect and must be defined before the optimization. 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 weight associated to the objective f_i, and n is the total number of objectives of the problem.

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

One important consideration to have when following this approach is its sensitivity to the chosen weights, as different weights can yield potentially distinct results. Architects often use their experience or knowledge about the problem itself to set these weights properly, thus ensuring they obtain an optimal (or near optimal) solution.

Virtues of this approach, in architecture, include the ease of use, the availability, the heterogeneity of ready-to-use SOO tools (e.g., Opossum, Goat, Galapagos, Silvereye), and the time required.

Overall, this approach enables a more intelligent design process because the algorithm uses knowledge about previously evaluated solutions to guide the search towards optimal regions of the design space. However, since the algorithms used are usually autonomous, architects are often removed from 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 major 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, this optimization approach no longer provides enough information to make informed decisions.

2.3.3 Pareto-based Optimization

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 k objectives, and f_i represent the objective function i.

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

On the other hand, in this approach, (1) the number of function evaluations is larger due to the need to find a set of optimal solutions instead of focusing on a single one, (2) the visual representation of the solutions' objective space becomes problematic when the number of objectives is greater than three, and (3) the way the optimization problem is modeled has a direct impact on the quality of the solutions.

Notwithstanding the considerations above, a few studies concerning Pareto-based optimization emerged in the past years, evidencing its utility [Evins, 2013, Hamdy et al., 2016]. Moreover, recent works show that even though approaches based on the *a priori* definition of preferences are less time-consuming, they are not as desirable as Pareto-based approaches, as they return a single solution instead of multiple alternative solutions [Attia et al., 2013, Hamdy et al., 2016, Cichocka et al., 2017a]. In fact, Pareto-based approaches support the decision making process, providing a clear trade-off between the different objectives involved. Moreover, these multiple compromises represent different articulation of preferences from which the architect selects one. This approach is also called *a posteriori* articulation of preferences.

Despite the growing trend of Pareto optimization approaches [Evins, 2013, Hamdy et al., 2016], the lack of relevant benchmarks comparing the performance of different Multi-Objective Optimization

Algorithms (MOOAs) in architecture is evident.

2.4 Quantitative Performance Indicators

Despite the large interest in MOO, the question of how to quantitatively compare the performance of different algorithms still remains unanswered. Firstly, in multi-objective problems, the number of objectives in the objective space is greater than the number of objectives in single-objective problems: the former considers a collection of vectors representing the Pareto-optimal solutions, whereas the latter considers real numbers. Secondly, it is often the case that the application of exact methods to MOO contexts is impracticable due to the complexity introduced by underlying applications (e.g., simulation tools, physical experiments). In these cases, the generation of the true Pareto-optimal set is often infeasible, requiring vast computational resources to be generated. Thirdly, despite the availability of alternatives to exact methods, such as metaheuristics algorithms (e.g., evolutionary algorithms, particle swarm), these are not guaranteed to identify optimal trade-offs, instead yielding good approximations [Zitzler et al., 2003]. Finally, we are interested in knowing which of the non-exact algorithms yields better approximations for a given problem, hence prompting the need for assessing the performance of MOOAs.

The notion of performance includes not only the quality of the results, but also of the computational resources needed to generate such results. While the latter aspect is usually identical for both SOO and MOO, which either typically consider the number of expensive evaluations or the overall run-time on a particular computer, the quality aspect is considerably different. Because SOOs consider real-valued objective spaces, the quality is defined in terms of the objective function: the smaller (or larger) the value, the better the solution. However, MOOs consider vector-valued objective spaces, thus requiring another concepts like Pareto dominance. Unfortunately, when considering the Pareto dominance concept a few issues may arise, namely the possibility of two solutions being incomparable, i.e., when neither dominates the other, or having solutions in one set that either dominate or are incomparable to those in the other set of solutions and vice versa.

Literature review evidences the existing struggle to define the meaning of quality with respect to approximation of Pareto-optimal sets [Knowles and Corne, 2002, Riquelme et al., 2015]. However, the quality of Pareto sets is usually evaluated in terms of three aspects: (1) cardinality, meaning larger sets of solutions, (2) diversity, meaning that solutions should be as uniformly distributed as possible, so as to obtain a representative set of solutions covering to larger extents the different trade-offs, and (3) accuracy, meaning that solutions should be as close as possible to the true Pareto-optimal set or Pareto Front.

For the past decades, several indicators have been proposed to measure the quality of Pareto sets, ranging from unary quality measures, which assign each approximation set a number that reflects a

certain quality aspect, to binary quality measures, which assign numbers to pairs of approximation sets, among others. This thesis considers a small but representative set of quantitative indicators to measure the quality of MOOAs' results with respect to the three aspects: cardinality, diversity, and accuracy.

Literature review reveals the existence of dozens of indicators that consider either one of the mentioned aspects or a combination of them. In the following definitions we use the term *approximation set* to denote the Pareto Front returned by the optimization algorithm, and we use the term *reference set* to denote the true Pareto Front or, whenever that is not possible, an estimate of the true Pareto front. In this section, we list some of the most used indicators for assessing the performance of evolutionary MOOs [Riquelme et al., 2015].

Should I discuss disadvantages of these indicators?

2.4.1 Unary Indicators

With regards to the cardinality aspect, there are essentially two indicators:

- **Overall Non-dominated Vector Generation (ONVG)** computes the number of non-dominated solutions in the approximation set.
- **ONVG Ratio (ONVGR)** computes the ratio of non-dominated solutions in the approximation set with regards to a reference set.

Conversely, there are more indicators that measure the distribution of solutions in the objective space, i.e., the diversity aspect of the approximation set:

- **Spacing** (or Set Spacing) computes the variance of the Manhattan distances between each non-dominated solution and its closest neighbor. It measures how well-spaced the solutions from the approximation set are. A value of zero represents equally spaced non-dominated solutions.
- **Delta Indicator** (spread or Δ) is similar to Spacing. However it calculates the normalized variance and uses the Euclidean distance instead.
- **Maximum Spread** (or M_3^*) computes the Euclidean distance between the bounds of each objective dimension. It measures the extent of the objective space in each dimension by calculating the distance between the maximum and minimum of each objective. A greater value indicates larger coverage of the objective space.
- **Entropy** uses the Shannon's entropy concept to measure the uniformity of the approximation set distribution. This indicator makes the assumption that each solution provides some information about its vicinities, thus modeling each solution with Gaussian distributions. These distributions add up to form a density function capable of identifying peaks and valleys in the objective space, corresponding to dense and sparse zones, respectively.

- **Diversity Metric** is similar to the Entropy indicator. However it projects the solutions of both the approximation set and the reference set to an hyperplane which is then subdivided uniformly. Then it computes the ratio between the number of intervals that have at least one non-dominated solution from both sets and the number of intervals that have at least one non-dominated solution of the reference set. Higher values of the diversity metric imply a better distribution and higher diversity of the approximation set when compared to the reference set itself.

In terms of accuracy, the following indicators are frequently used to measure the convergence of the approximation set:

- **Error Ratio (ER)** computes the proportion of false-positives in the approximation set, i.e., the ratio of optimal solutions in the approximation set that are not optimal in a given reference set. Lower values of ER, represent better the approximated sets.

- **Maximum Pareto Front Error (MPFE)** computes, for each solution in the approximated set, the minimum Euclidean distance to the closest solution in a given reference set, returning the maximum of those distances. In other words, it returns the maximum error of the approximated Pareto Front. Lower values of MPFE imply better approximation sets.

- **Generational Distance (GD)** computes the average distance of an approximation set to a given reference set by computing the average distance of the solutions in the approximation set to the nearest points in a given reference set. A value of 0 indicates that all the solutions in the approximation set are in the reference set. This metric has also been called M_1^* in other works [Zitzler et al., 2000].

Finally, there are also some indicators that provide insights about both the distribution of the solutions of the approximation set and its accuracy:

- **Hypervolume (HV)** (or Lebesgue measure or S-metric) measures the size of the objective space covered by an approximation set, i.e., it measures the volume of the dominated space. It provides the unique and desirable properties of (i) Pareto *compliance*, i.e., an approximation set which completely dominates another, will necessarily have a greater volume than the latter, and (ii) convergence guarantees, i.e., any approximation set that achieves the maximum possible volume is guaranteed to contain all Pareto-optimal solutions.

- **Inverted Generational Distance (IGD)** is the opposite of GD, instead computing the average distance between a given reference set and the approximation set. IGD computes the distance between each solution in the reference set and its closest solution in the approximation front, averaged over the size of the reference set. Smaller values of IGD suggest better approximation

sets, given that the reference set is well distributed. IGD is identical to a previous metric, known as $D1_R$.

2.4.2 Binary Indicators

In situations where the original Pareto-optimal set is not available, the binary indicators provide a way to compare approximation sets with respect to all the aspects. Among the most used indicators, there are:

- **Two set coverage** (or C-metric) yields the number of solutions in one approximation set that are dominated by at least one of the solutions of the other approximation set. A value of 1 suggests that the second approximation set is completely dominated by some solutions in the first one, whereas a value of 0 represents the situation when none of the solutions of the second approximation set is covered by the first approximation set.
- **Epsilon Indicators** (ϵ) that gives a factor by which an approximation set is worse than another considering all objectives, i.e., given two approximation sets A and B, it computes the smallest amount ϵ by which A must be translated, so that every solution in B is dominated by at least one solution in A.
- **R-metrics** consider a family of indicators where the quality of each approximation set is defined according to a set of utility functions, and the larger the utility value of some set, the better the quality. R-metrics declare that the best approximation set will be the one that is better with regards to most utility functions:
  - $R1$ determines whether the an approximation set is better, equal, or worse than the other.
  - $R2$ computes the expected mean difference in the utilities of both approximation sets.
  - $R3$ computes the expected mean relative difference in the utilities of both approximation sets.

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 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 tools, the

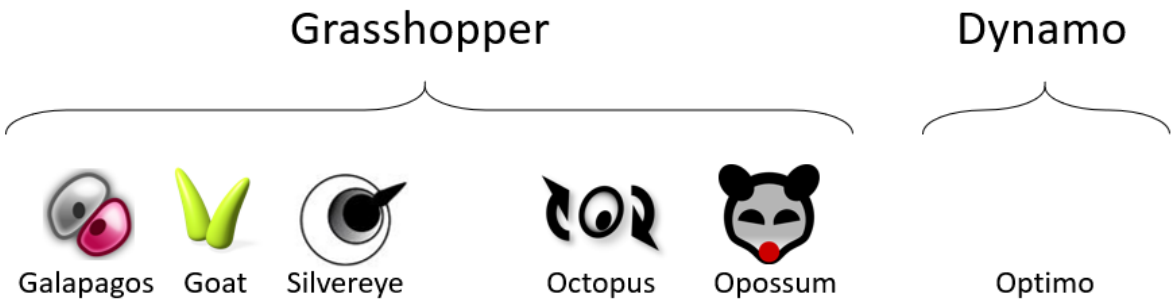


Figure 2.3: Optimization frameworks currently used in architectural practices.

simulation or analytical models, and the optimization frameworks. Given the visual nature of architects, these plug-ins provide friendly, ready-to-use optimization interfaces, which are usually coupled with a few post-processing and visual features to enhance the intelligibility of optimization results.

Currently, existing optimization plug-ins are implemented on top of the visual parametric tools: Grasshopper [David G. Rutten, 2007] and Dynamo [Ian Keough, 2011]. In Figure 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 available 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 the basis for a 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 to proper literature throughout this section.

As evolutionary algorithms, genetic algorithms explore Darwinian natural selection concepts, such as heredity, reproduction, and natural selection, and genetics concepts and mechanisms, including 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. 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 they specify the design's parameters; (2) Generation, where they create the design's algorithmic model that when instantiated with the parameters' values will generate the 3D model; and (3) Analysis, where they define the analysis function or objective function which they ought to optimize. To that end, architects use Grasshopper's components, such as *Sliders*, values lists, area, distance, and other analytical-like components to define the programs. 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* (*Num*) component. Both the *Sliders* and the *Num* components connect to the input entry and to the output entry of the Galapagos' component, respectively, i.e., the genome and the fitness entries.

After defining the problem, the architect double clicks the Galapagos component and is presented with its Graphical User Interface (GUI) (see Figure 2.4). This interface is friendlier, simpler, and easier to use than other 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 architects [Wortmann and Nannicini, 2017], not only satisfying the visual nature of architects, but also supporting graphical views that yield feedback about the course of the optimization process.

Particularly, Galapagos supports four graphical views for the genetic algorithm (see Figure 2.5-left), namely (1) the fitness graph, representing the distribution of fitness values in the population per genera-

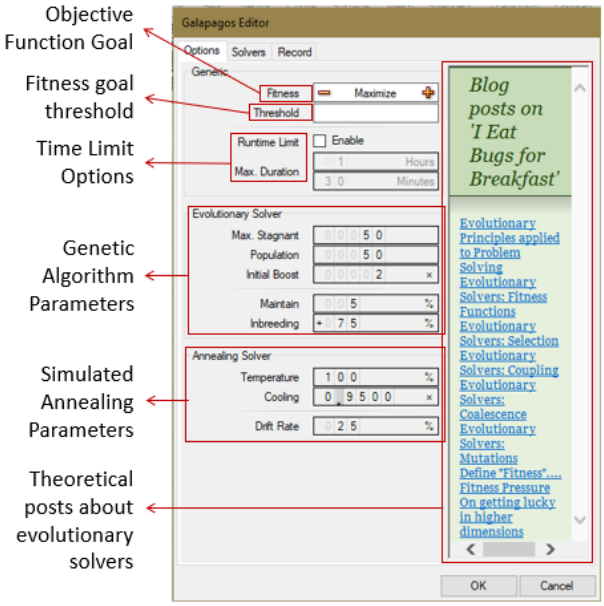


Figure 2.4: Galapagos Options menu for parametrizing the solvers

tion, (2) the similarity representation graph, 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) the parallel coordinates graph, where solutions are represented as connected line segments and each vertical line corresponds to a parameter, and the (4) ranked list of the best solutions found so far. These views exhibit all the results of the most recent generation and highlights its best solution. However, Galapagos provides the user with the flexibility to select multiple generations and to reinstate a particular solution, allowing the materialization of the desired design solution. When using the simulated annealing solver, Galapagos provide a slightly different view of the optimization process (see Figure 2.5-right): (1) a fitness graph depicting each solution's fitness value at each step; (2) a temperature graph, representing the temperature decrease rate with each time step; and (3) a ranked list of the best solutions found so far.

2.5.2 Goat

Goat is an optimization plug-in that is available in Grasshopper and exposes in a GUI the optimization algorithms from the NLopt mathematical optimization library [Steven G. Johnson, 2010]. Simon Flöry developed Goat as a generic platform designed to use algorithms from other classes of derivative-free algorithms. Goat exposes these algorithms via an interface resemblant of Galapagos, making it easier for non-programmers to solve problems.

One of the key differences between Goat and Galapagos is the number and diversity of the algorithms available. Whilst Galapagos supports two global metaheuristics algorithms, Goat supports five

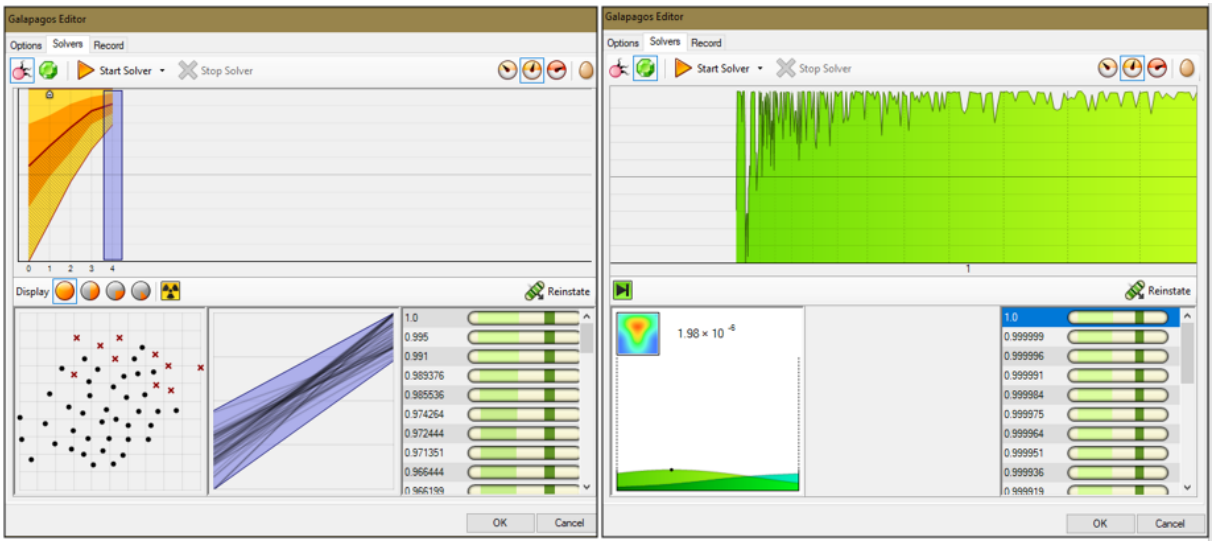


Figure 2.5: Examples of the Galapagos provided views for: (left) Genetic Algorithm, (right) Simulated Annealing algorithm

distinct algorithms: one metaheuristic, two direct-search, and two model-based algorithms, called CRS2, DIRECT, SUBPLEX, COBYLA, and BOBYQA, respectively. Due to space constraints, a full description of these algorithms will not be provided, instead we refer the interested reader to the relevant literature.

The CRS2 algorithm is a variant of the Controlled Random Search (CRS) algorithm for global optimization [Price, 1983]. A CRS algorithm is a population-based random search algorithm that creates an initial set of points, the population, which are then randomly evolved by means of heuristic rules. In the original CRS algorithms, heuristic rules modify a point at a time, replacing the worst point with a better one (called trial point), using a technique resemblant of the NMS algorithm [Nelder and Mead, 1964]. Similarly to NMS, CRS algorithms use a simplex, i.e., a generalized triangle in N dimensions, to envelope a region described by a random subset of points in the population. The worst point of the population is replaced with the reflection of the worst point in the simplex [Kaelo and Ali, 2006]. The CRS2 variant differs from the original in that it assumes that the worst population point will always be a part of the simplex. The actual version that is made available by Goat is a modified version of the CRS2 algorithm that introduces a local mutation component in an attempt to overcome situations where heuristic rules constantly fail to find a trial points that actually improve the worst point. Fundamentally, this local mutation generates a second trial point that results from the exploitation of the region around the best point in the population [Kaelo and Ali, 2006].

The second algorithm is a global deterministic direct search algorithm that relies on the division of rectangles, as explained in [Jones et al., 1993]. DIRECT, the DIviding RECTangles algorithm, recursively subdivides the design space into smaller multidimensional hyper-rectangles, estimating the quality value of each rectangle. DIRECT uses these values to focus the search on more promising regions of

Vantagens e desvantagens ? Lack sound convergence properties, robustness and performance are highly

the design space and to further subdivide those in smaller hyper-rectangles. The SUBPLEX algorithm is an unconstrained local optimization algorithm [Rowan, 1990]. As a generalization of the NMS algorithm, SUBPLEX subdivides the design space in low-dimensional subspaces and then applies the NMS algorithm to a set of these subspaces, in order to seek for a better solution. In contrast to NMS, which has difficulties in high-dimensional problems, SUBPLEX reduces the limitations through the decomposition of the problem in low-dimensional subspaces which are more efficiently optimized by NMS.

Besides metaheuristics and direct-search algorithms, Goat also provides two local model-based implementations, namely the COBYLA (or Constrained Optimization BY Linear Approximation) algorithm [Powell, 1994], and the BOBYQA (or Bound Optimization BY Quadratic Approximation) algorithm [Powell, 2009], that rely on the construction of simple, partial models of the objective function [Koziel and Yang, 2011]. The former uses the concept of simplex to iteratively generate linear approximations of the objective function, whereas the latter generates quadratic approximations instead.

One of the main advantages of Goat is the algorithms' diversity. By providing algorithms with different characteristics and strategies, architects can test the suitability of each algorithm to their problem and, thus, select the most effective. The right choice may result in large optimization gains, especially when complex and time-consuming simulations are necessary [Wortmann and Nannicini, 2016]. For this reason, and due to the uniqueness of each BPO, several authors suggest that the selection of the optimization algorithm should be based on the results of several tests with different algorithms for a fixed number of evaluations or a fixed amount of time [Hamdy et al., 2016, Wortmann and Nannicini, 2016]. Moreover, the distinction between global and optimal algorithms is also critical when striving for accurate and precise optimal solutions. Most global optimization algorithms invest most of their effort searching for the truly optimal solution across large regions of the search space and rarely focusing on promising regions. Consequently, they might return a not so precise global optimum. To overcome this lack of precision and accuracy, one should apply a local optimization algorithm and provide the globally imprecise optimum as input.

While in terms of usability Goat is very similar to Galapagos, it lacks the immediate feedback and post-processing features of Galapagos. On the one hand, Goat's GUI (see Figure 2.6) is simple and easy to use for non-experts, requiring no integration efforts to use one of its optimization solvers, nor any programming or algorithm's knowledge in order to setup and use it. Moreover, the option to specify the starting point using the *Sliders* components is beneficial for improving the efficiency of optimization processes.

On the other hand, Goat lacks the visual, and post-processing features. In fact, except for a final report stating the termination criteria (e.g., reach a threshold, running time exceeded), Goat provides no feedback regarding the course of the search for optimal solutions and as a result of the optimization process, Goat returns a single solution. The values for the optimal solution are exclusively exhibited

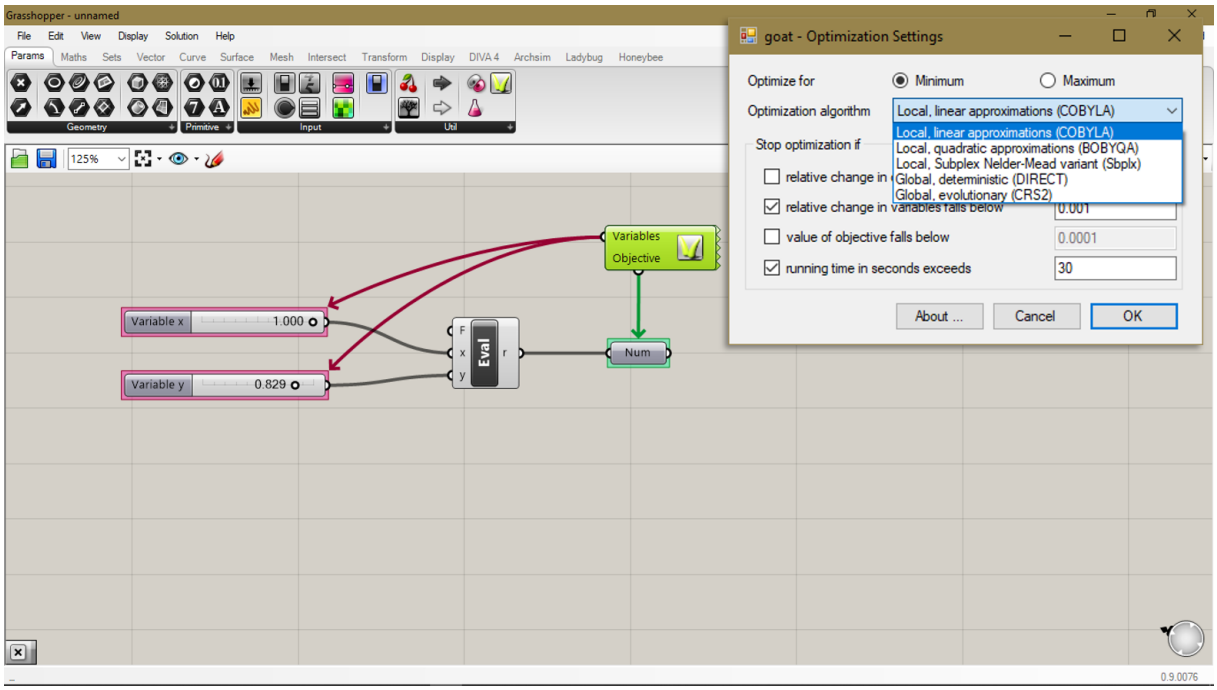


Figure 2.6: Simple example of a Grasshopper's problem definition with two variables and a function. The Goat component is represented in green and the *goat - Optimization Settings* menu lists the supported algorithms

using Grasshopper's *Sliders*, which is not a very robust feedback mechanism (e.g., susceptible to unwilling modifications, or to information loss). In addition to bad scalability and representation of the result, Goat provides no information about other close to optimal solutions, which spurs no confidence on the achieved results.

2.5.3 Silvereye

Silvereye is an optimization plug-in for Grasshopper, developed under the same user interface design as Galapagos, that enables non-experts to solve complex optimization problems. Silvereye interfaces a C# implementation of a single-objective PSO algorithm.

The PSO algorithm is a global metaheuristic algorithm inspired by biological systems, such as the collective behavior of flocking birds and schooling fish, which interact and learn from one another to solve problems [Brownlee, 2011]. In PSO, the intelligence is decentralized, self-organized, and distributed throughout the participating particles, also known as swarm. These particles maintain information about their velocity, their current and personal best positions, and also the global best position known to the swarm. At each time step, the position and velocity of each particle are updated according to the best swarm or close neighbor position [Brownlee, 2011].

In terms of usability, Silvereye is resemblant of Galapagos, requiring the definition of Grasshopper's

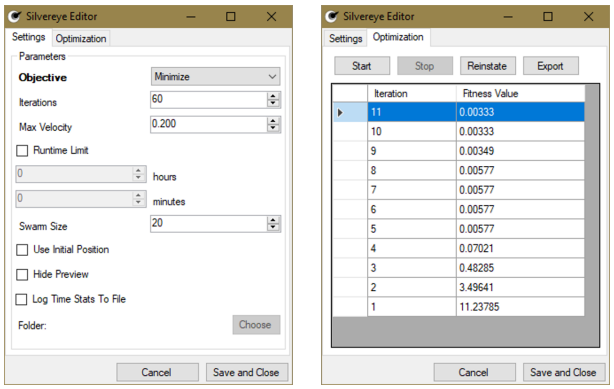


Figure 2.7: The Silvereye Editor menus: (left) Settings menu, (right) Optimization results

Sliders as variables and a Grasshopper's *Num* component as the objective function, which are then connected to the input and output entries, respectively. Upon double-clicking the Silvereye component, the user is presented with the settings menu (see Figure 2.7) where he is able to fine-tune the optimization solver: (1) `maximum velocity`: in each iteration, a variable can change its value by an amount that is less or equal than the value specified for this option, (2) `swarm size`: the number of particles used by the PSO algorithm, and (3) `Use initial position`: to define one starting point, users must narrow down the range of the slider's variables, update the velocity in the solver's settings, and check this option in the Settings Menu. Moreover, Silvereye enables the user to save the solver's configurations and use them in the next run.

Silvereye supports two information mechanisms, neither of them being graphical. The first mechanism is a time log of the optimization run, discriminated by iteration and by particle. This log enables the user to control the optimization run and detect any irregularity during the run. Unfortunately, the log does not have information about the values of the solution's parameters, which makes the traceback of errors impractical. The second mechanism is a list of the best result obtained in each iteration, which can be either exported to a file or reinstantiated in Grasshopper to visualize the corresponding design. Nevertheless, similarly to the first mechanism, this list presents no explicit information about the parameters' values of explored solutions and the user is restricted to the best result listed, not being able to verify other close to optimal solutions that might have been explored in the course of the optimization.

2.5.4 Opossum

Opossum (OPtimizatiOn Systems with SUrrogate Models) is a single-objective plug-in developed for Grasshopper that interfaces the model-based RBFOpt library [Costa and Nannicini, 2014]. Its GUI is similar to the one of Galapagos.

Opossum is a model-based optimization tool for Grasshopper that uses the RBF machine learning technique to create global approximations of the objective function [Forrester and Keane, 2009]. These

approximations are simply the weighted sum of other, simpler, real-valued functions, the radial functions. These functions are defined on the Euclidean space $\mathbb{R}^n$ and their value depends on the distance to a center c, so that $\phi(x, c) = \phi(\|x - c\|)$. In a RBFs technique, the weights are estimated based on the interpolation of data. A comprehensive detailed explanation of the RBF's estimation process is provided in [Forrester and Keane, 2009].

Since its invention, multiple implementations of RBF have been proposed. RBFOpt implements two well-known versions, namely, the Gutmann's [Gutmann, 2001] and the Regis and Shoemaker's [Regis and Shoemaker, 2007], commonly known as MSRSM. These techniques differ in the search strategy for the next candidate solution to be evaluated using the original objective function. The former uses the solution, which is likely to yield the largest improvement in the surrogate's accuracy, whereas the latter tries to balance the surrogate's accuracy amelioration with the exploitation of promising solutions, using other search strategies, such as genetic algorithms, sampling algorithms, or other mathematical solvers [Wortmann, 2017b]. Moreover, RBFOpt provides five different types of radial basis function: linear, multi-quadratic, cubic, thin plate spline, and automatic selection, which are also provided in the Opossum interface.

In terms of usability, Opossum resembles Galapagos. In contrast to other plug-ins, Opossum provides a more flexible interface, with menus tailored for different expertise levels, offering different levels of control in each one. By providing multiple menus (see Figure 2.8), both less and more experienced users are able to use a tool that suits their needs, while providing the more experienced ones the ability to fine-tune the optimization solvers more accurately, thus potentially achieving more efficient optimization processes.

One other important aspect of Opossum is concerned with the exposition of results. On the one hand, Opossum provides a graphical view of the best value ever found up to each iteration (see Figure 2.8-left), thus providing immediate feedback to the architect about the state of the optimization process. On the other hand, Opossum can create a log file with the record of all the solutions explored during the optimization process. This log file can then be used with other post-processing tools for purposes of more insightful visualizations of the data, or even to re-use this information to perform sensitivity analysis or to run other optimization algorithms (e.g., benchmarks).

2.5.5 Octopus

Octopus is a multi-objective plug-in developed for Grasshopper. Designed in the same way as Galapagos, Octopus exploits evolutionary principles to search for multiple Pareto-optimal solutions.

The installation of Octopus creates a new tab within Grasshopper, which encloses seven menus discriminated by different functionality, including Evolutionary Learning, Octopus, Supervised Learning, among others. Octopus enables the simultaneous search for more than one objective, producing a set

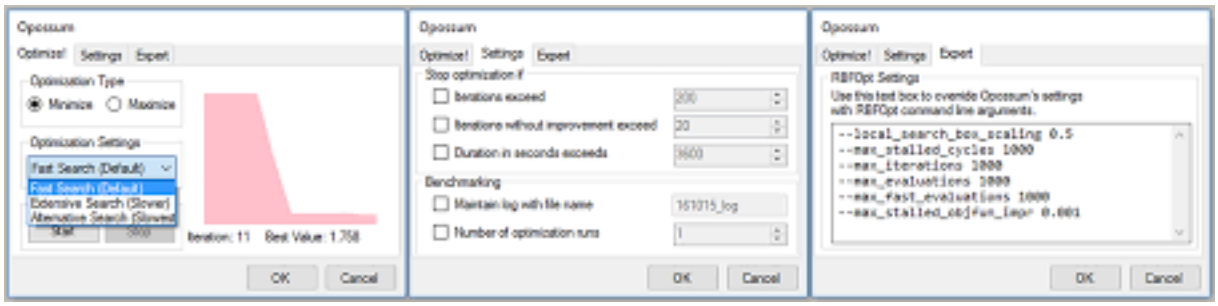


Figure 2.8: The three menus of the Opossum's GUI

of possible optimal solutions that ideally represent the real trade-offs between each objective. In order to search for the Pareto-optimal solutions, Octopus provides two main evolutionary algorithms: Strength Pareto Evolutionary Algorithm 2 (SPEA2) and Hypervolume Estimation Algorithm for MOO (HypE). On the one hand, due to their population-based nature, evolutionary algorithms are able to approximate the Pareto Front in a single run [Zhou et al., 2011]. On the other hand, these algorithms have been reported to yield promising results on numerous test problems [Zitzler et al., 2001, Bader and Zitzler, 2011].

Similarly to evolutionary algorithms, MOEAs adopt the evolutionary principles discussed in section 2.5.1 but, generally, have an additional archive to store the non-dominated solutions found [Zitzler et al., 2001]. The archive technique is incorporated to prevent losing current non-dominated solutions due to random mutations or recombinations. Most MOEAs differ in the selection and reproduction operators used to iteratively evolve populations. These differences are usually related to the very own goals of approximating the Pareto Front: (1) maximize the accuracy of the approximation (by minimizing the distance to the optimal front) and (2) maximize the diversity of the solutions within the front. While the first goal is related to the search strategy and how to assign fitness values in such a way that the individuals selected for offspring production will be closer to the Pareto-optimal front, the second goal is related to the time and storage constrains of the evolution process and which individuals to keep in each generation [Zitzler et al., 2001]. Although a thorough description of different MOEAs and their mechanisms is not herein presented, we refer the interested reader to [Zhou et al., 2011].

Most modern MOEAs realize the accuracy and diversity ideas through some implementation of the following mechanisms [Zitzler et al., 2001]:

- Selection (or environmental selection): Besides the population, most MOEAs maintain an archive with the non-dominated front among all the solutions that were evaluated. The archive preserves individuals during several generations, only removing them if (1) a new solution is found to dominate them, or (2) if the archive size is exceeded and they happen to be in crowded regions of the front.

- Reproduction (or mating selection): At each generation, individuals are evaluated in two stages.

The first stage compares them regarding the relation of Pareto dominance, using this information to define a ranking among these individuals. The second stage refines these rankings through the incorporation of density information, i.e., if the individuals lie in crowded regions.

These mechanisms can be completely indifferent from one another, e.g., the first one applying a Pareto-based criteria and the second one applying weighting approach. However, many MOEAs implement both concepts similarly. In the particular case of SPEA2, the algorithm explores two independent sets of individuals: the population and the archive. In this algorithm, the archive size is fixed and, therefore, whenever the number of non-dominated individuals is less than its size, some dominated individuals are added to the archive. At each iteration, the algorithm computes each individual's fitness value (a *strength* value, defined as terms of the number of solutions it dominates, the *raw fitness* value, defined in terms of the strength of its dominators, and a density estimate, defined as the inverse of the distance to the k^{th} to the nearest neighbor) and copies the non-dominated individuals from the population to the archive, removing any individual that is dominated, whose objective values are duplicated, or that, when the size of the updated archive is exceeded, lies in crowded regions of the non-dominated front. After filling the archive, pairs of individuals are chosen from the archive to reproduce and produce the offspring through recombination and mutation operators that will make the population of the next generation [Zitzler et al., 2001].

Unfortunately, algorithms incorporating the Pareto dominance relation and diversity measures appear to have difficulties in optimization scenarios with more than two objectives, which spurred the development of algorithms using other quality measures, including quality indicators. To overcome the objective limitation and provide the user with the flexibility to use a more efficient algorithm, Octopus provides the option to use HypE, an algorithm that explores the HV indicator to rank individuals. Particularly, to minimize time penalties associated with the computation of the HV, in scenarios with more than three objectives, HypE uses Monte Carlo simulations to estimate the HV value of each individual [Bader and Zitzler, 2011].

In terms of usability, Octopus is very similar to Galapagos. Figure 2.9 illustrates the Octopus editor after running SPEA2 for fourteen generations for a 3-objective optimization problem. This interface lacks organization of its functionalities and is overloaded with information, thus hindering its overall usability and comprehension.

One key aspect of this interface is the ability to configure multiple parameters of both the algorithm and the optimization run, even during runtime. Moreover, by default, Octopus disables the real-time design generation of design candidates with the aim of reducing time penalizations. The consequence of such option is an optimization process where no information is provided to the user.

Another key aspect of this interface is the post-processing and visualization mechanisms. On the one hand, it provides multiple views of the evolution of the optimization process (see views 1, 8, and 10

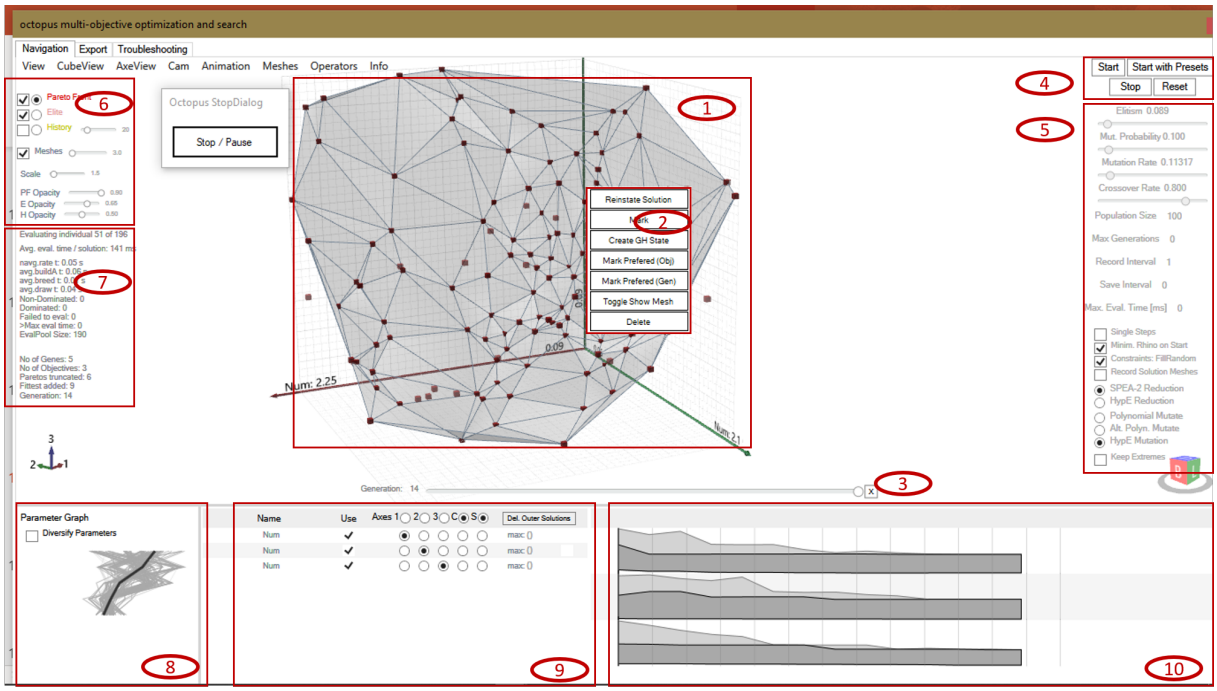


Figure 2.9: Octopus Editor: (1) Solutions' Objective Space, (2) Solution's context menu, (3) Generation's history slider, (4) Process control options, (5) Algorithm Settings, (6) Display Settings, (7) Statistics, (8) Parallel Coordinates Graph, (9) List of Objectives (10) Convergence graphs per Objective

in Figure 2.9). While these views provide a comprehensive feedback about the course of optimization for problems with less than or equal to five objectives (e.g., using color and dimension as additional features), it is unclear how Octopus represents solutions with more than five objectives in the main view port. On the other hand, it allows the user to export different information to files.

2.5.6 Optimo

Implemented on top of Dynamo, Optimo is an evolutionary-based multi-objective optimization plugin that enables users to optimize both single and multi-objective problems. The current version of Optimo uses an Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al., 2002] from the jMetal.NET library to seek for Pareto-Optimal solutions.

As a MOEA, NSGA-II attempts to achieve an approximation to the Pareto front that is both accurate and diverse. In this particular algorithm, both the selection and reproduction mechanisms rely on the same basis criteria: Pareto dominance relations and a crowding measure. To define the order among the individuals, the pool of individuals is split into different Pareto fronts and ranked accordingly, i.e., the first non-dominated front is assigned the highest rank, the second front is assigned the second highest rank, and so on. Each individual in each rank is ordered according to a crowding measure, represented in terms of the sum of distances to the two closest individuals along each objective. The archive and

the generation's population are combined, deleting the worst 50%. Afterwards, binary tournaments are carried out on the remaining individuals (the archive members) in order to generate the next offspring population.

In order to help designers optimize multiple conflicting objectives and approach to a set of optimal solutions, Optimo includes four nodes in Dynamo: (1) Initial Solution List, which generates the initial set of random design configurations within a provided range and with the specified size of population; (2) Assign Fitness Function Results, which evaluates and assigns the objective values to each configuration; (3) Generation Algorithm, which takes the parent population and generates the children population; and (4) Sorting, which uses the Pareto Front sorting to sort the solutions. Comparing to the other plug-ins, Optimo presents more detailed and complex node structure, as it provides the user four different blocks to manipulated instead of enclosing them in a single abstraction.

Regarding the post-processing features, Optimo creates a CSV file with the results of the optimization run. However, it provides no visual feedback about the evolution of the optimization run nor about the best evaluated solutions, thus lacking post-processing features and requiring the usage of external applications or frameworks.

2.5.7 Comparison

2.6 Problems to Address

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.

[Bader and Zitzler, 2011] Bader, J. and Zitzler, E. (2011). HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evolutionary Computation*, 19(1):45–76.

[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.

[Deb et al., 2002] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[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 Leitão, 2015] Ferreira, B. and Leitão, A. (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.

[Gutmann, 2001] Gutmann, H.-M. (2001). A Radial Basis FunctionMethod for Global Optimization. *Journal of Global Optimization*, 19(3):201–227.

[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.

[Jared Banks, 2012] Jared Banks (2012). Work Environments in ArchiCAD - part 2D.

[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.

[Kaelo and Ali, 2006] Kaelo, P. and Ali, M. M. (2006). Some variants of the controlled random search algorithm for global optimization. *Journal of Optimization Theory and Applications*, 130(2):253–264.

[Knowles and Corne, 2002] Knowles, J. and Corne, D. (2002). On Metrics for Comparing Nondominated Sets. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, pages 711–716.

[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, 1964] Nelder, J. and Mead, R. (1964). A simplex method for function minimization. 7:308–313.

[Nemhauser and Wolsey, 1988] Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*. John WIley & Sons, New York.

[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.

[Powell, 2009] Powell, M. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge University.

[Powell, 1994] Powell, M. J. D. (1994). A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In Gomez, S. and Hennart, J., editors, *Advances in Optimization and Numerical Analysis*, number 275, pages 51–67. Springer, Dordrecht.

[Price, 1983] Price, W. L. (1983). Global Optimization by Controlled Random Search. *Journal of Optimization Theory and Applications*, 40(3):333–348.

[Regis and Shoemaker, 2007] Regis, R. G. and Shoemaker, C. A. (2007). A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509.

[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.

[Riquelme et al., 2015] Riquelme, N., Von Lucken, C., and Baran, B. (2015). Performance metrics in multi-objective optimization. *2015 Latin American Computing Conference (CLEI)*, 1:1–11.

[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).

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

[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*.

[Zhou et al., 2011] Zhou, A., Qu, B.-y., Li, H., Zhao, S.-z., and Nagaratnam, P. (2011). Multiobjective evolutionary algorithms : A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49.

[Zitzler et al., 2000] Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary computation*, 8(2):173–195.

[Zitzler et al., 2001] Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm Eckart. pages 12–19.

[Zitzler et al., 2003] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert, V. (2003). Performance Assessment of Multiobjective Optimizers : An Analysis and Review. 7(2):117–132.