TÉCNICO LISBOA

Single-Objective Optimization for Architecture

Guilherme Ilunga

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. António Paulo Teles de Menezes Correia Leitão

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. António Paulo Teles de Menezes Correia Leitão
Member of the Committee: Prof. João Emílio Segurado Pavão Martins

February 2019

Agradecimentos

Agradeço ao meu orientador. O professor António Leitão, ao longo de todo este tempo, mais do que pensar apenas no trabalho, pensou em mim. Dedicou tempo mesmo quando não tinha, para eu poder cumprir os meus objetivos.

Agradeço ao Grupo de Arquitetura e Computadores, grupo com o qual trabalhei durante o desenvolvimento desta tese. Muito obrigado Bruno Ferreira, Catarina Belém, Francisco Loio, Inês Caetano, Inês Pereira, Renata Castelo Branco, Rita Aguiar, Sofia Feist, Sofia Sousa, e aos restantes membros do grupo.

Agradeço aos professores Luís Rodrigues, João Barreto, e Miguel Pardal. Durante a realização desta tese, tive a oportunidade de trabalhar como bolseiro de apoio à docência para as cadeiras de Sistemas Operativos e Sistemas Distribuídos. Muito obrigado a estes professores, e aos restantes docentes destas cadeiras, por me terem permitido trabalhar enquanto realizava a tese.

Agradeço aos meus amigos. Os meus amigos do Instituto Superior Técnico, Carolina Fernandes, Catarina Belém, Cláudia Belém, Filipe Magalhães, Gonçalo Rodrigues, Nuno Afonso, Telma Correia, Valentyn Hulevych, e os meus amigos do Externato Frei Luís de Sousa, Ana Rita Bello, Ana Teresa Martins, Francisco Henriques, Inês Franco, Paulo Santos, e Tomás Fernandes. Mesmo tendo as suas próprias teses ou outros deveres, todas estas pessoas me apoiaram no meu percurso.

Por fim, agradeço à minha família, cujo apoio incondicional foi a base de todo este trabalho. Muito obrigado aos meus pais, avós, e tios.

Abstract

The focus on efficiency has grown over recent years and, nowadays, it is critical that buildings exhibit good performance regarding different criteria. This need prompts architects to explore (1) algorithmic design approaches, which allow the generation of several design variations, (2) analysis tools, to evaluate a design's performance, and (3) optimization algorithms, to find the best performing variation of a design. Many optimization algorithms exist, and not all of them are adequate for a specific problem, however Genetic Algorithms are frequently the first and only option in architectural optimization. This may be because existing optimization frameworks require the usage of specific design or analysis tools and only offer a small subset of optimization algorithms, which leads to the simplest algorithm being the usual choice. This dissertation studies existing approaches and optimization algorithms for architecture. It also proposes a framework for architectural optimization that includes several types of algorithms and a prototype implementation. This prototype is tested using case studies, and the results show that Genetic Algorithms perform poorly, while other algorithms achieve better results. However, they also show that no algorithm is consistently better than the others, which suggests that a framework with several different types of algorithms should be used.

Keywords

Black-box Optimization; Derivative-free Optimization; Architectural Optimization; Performance-based Design; Algorithmic Design.

Resumo

O foco em eficiência tem crescido nos últimos anos e, atualmente, é crítico que os edifícios demonstrem um bom desempenho em diferentes critérios. Esta necessidade leva os arquitetos a explorar (1) projeto algorítmico, que permite a geração de múltiplas variações do projeto, (2) ferramentas de análise, para avaliar o desempenho de um projeto, e (3) algoritmos de optimização, para encontrar a variação do projeto com o melhor desempenho. Existem vários algoritmos de optimização e nem todos são adequados para um determinado problema, contudo Algoritmos Genéticos são frequentemente a primeira e única opção em optimização arquitetural. Isto ocorre porque apesar de existirem algumas frameworks de optimização, a maioria delas necessita de ferramentas de projeto ou análise e apenas oferecem um pequeno subconjunto de algoritmos, o que leva a que o algoritmo mais simples seja a escolha usual. Esta dissertação estuda as abordagens existentes e os algoritmos de optimização para arquitetura. Também propõe uma framework para otimização arquitetural que inclui vários tipos de algoritmos e que implementa num protótipo. O protótipo foi testado utilizando casos de estudo, e os resultados demonstram que os Algoritmos Géneticos não têm um bom desempenho, enquanto que outros algoritmos obtêm melhores resultados. Contudo, os resultados também demonstram que nenhum algoritmo é consistentemente melhor que os restantes, o que sugere que uma framework com vários algoritmos de diferentes tipos deve ser utilizada.

Palavras Chave

Optimização de Caixa Preta; Optimização Sem Derivadas; Optimização Arquitetural; Projeto Baseado em Performance; Projeto Algorítmico

Contributions

During the development of this dissertation, two scientific articles were published:

- *Case Studies on the Integration of Algorithmic Design Processes in Traditional Design Workflows*, published in the 23rd International Conference of the Association for Computer-Aided Architectural Design Research in Asia [Caetano et al., 2018]
- *Derivative-free Methods for Structural Optimization*, published in the 36th Education and Research in Computer-Aided Architectural Design in Europe Conference [Ilunga and Leitão, 2018]

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AD Algorithmic Design . . . 6

ADA Algorithmic Design and Analysis . . . 11

AEC Architecture, Engineering, and Construction . . . 5

API Application Programming Interface . . . 8

BIM Building Information Modelling . . . 5

BOBYQA Bounded Optimization By Quadratic Approximation . . . 25

CAD Computer-Aided Design . . . 5

CMA-ES Covariance Matrix Adaptation - Evolutionary Strategy . . . 24

COBYLA Constrained Optimization By Linear Approximation . . . 25

CRS2 Controlled Random Search 2 . . . 23

DIRECT Dividing Rectangles . . . 23

ESCH Evolutionary Strategy with Cauchy Distribution . . . 24

GA Genetic Algorithm . . . 16

GP Gaussian Process . . . 26

ISRES Improved Stochastic Ranking Evolutionary Strategy . . . 24

ML Machine Learning . . . 26

MLP Multi Layer Perceptron . . . 26

NFL No Free Lunch . . . 15

PBD Performance-based Design . . . 3

PRAXIS Principal Axis . . . 22

PSO Particle Swarm Optimization . . . 25

RBF Radial Basis Function . . . 26

SA Simulated Annealing . . . 23

1

Introduction

Contents

The field of architecture has seen many changes over the years, and recent advances in computational approaches have significantly altered the architectural design process. Modern architecture needs to handle concerns related with lack of resources and sustainability, as well as cost concerns. These concerns are also changing the design process, by adding standardized building regulations and energy certificates, among other limitations and restrictions. These changes have lead to the creation and development of new design approaches such as Performance-based Design (PBD), where the creative design process takes into account the design's performance, possibly measured using computational tools [Oxman, 2006]. This type of approach has been applied to several buildings, resulting in improvements of the overall design.

One example of the PBD approach is the Soumaya museum in Mexico city, shown in figure 1.1. The museum's external facade is composed of over 15,000 panels placed next to each other. The initial approach for designing these panels involved creating completely distinct panels. This approach was extremely costly and time-consuming to fabricate, since each panel needed to be fabricated separately. Since this was unfeasible, the leading architects decided to request the help of Gehry Technologies, a technology company created by renowned architect Frank Gehry, which specialized in complex designs and computational approaches for architecture. Their approach attempted to optimize the panels' shape, by analyzing the overall structure's shape and creating groups of equal panels which could be placed in specific zones. As a result of this approach, they discovered that it was possible to use just 300 groups of panels, instead of the initial 15,000, which drastically reduced the cost and time required to fabricate them [Sidelko, 2013].

Figure 1.1: A representation of the different panels of the Soumaya Museum in Mexico City. The number of different panels was optimized to be minimal as a cost reduction technique.

Another example where cost and sustainability concerns influenced design is the 30 St. Mary Axe building in London[1], also known as the Gherkin, shown in figure 1.2. Some of the windows placed along the tower automatically open and close to control airflow and temperature (black windows in figure 1.2). Internally, the design also attempts to tackle the cost concerns. Each floor has a gap connecting it to the upper and lower floors. In total, there are six shafts which span the entire tower, serving as a ventilation system, which allows for an easier control of internal temperatures. Also, since all panels are made of glass, the tower receives a large amount of direct sunlight. Overall, these aspects reduce the costs in lighting, heating, and cooling to 50% of a similarly sized tower, showing the potential impact of a PBD approach, especially when considering the cost reduction over the building's lifetime.

Figure 1.2: The 30 St. Mary Axe building in London, also known as the Gherkin, designed by Foster and Partners, together with the Arup Group.

The examples of the PBD approach mentioned so far still required several manual decisions by architects and engineers. This dissertation focuses on attempting to automate the process, by proposing a framework which simplifies the usage of optimization algorithms in the architectural design process, enabling the creation of optimized buildings. However, in order to automate this process, the entire design process must be addressed. Overall, there are three main stages that need to be considered: design, analysis, and optimization. The design stage encompasses the creation and description of a design, which must allow some degree of variability, in order to find an optimal variation of it. In the

[1] https://www.fosterandpartners.com/projects/30-st-mary-axe/, accessed 11 July 2018.

analysis stage, analysis tools may be used to evaluate the design. Finally, in the optimization stage, optimization algorithms can be used to discover the best performing variation of a design.

The following sections of this chapter describe the different stages with more detail and lay the foundation for the creation of an automatic optimization framework for architecture.

1.1 Design Stage

Over the past years, the evolution in computational tools has changed the design process in the Architecture, Engineering, and Construction (AEC) industries. Computer-Aided Design (CAD) tools are one type of tools that has been adopted in those industries, replacing pen and paper [Kalay and Mitchell, 2014]. These tools can be used for sketching purposes, allowing the user to create lines and curves, similarly to paper sketches, and they can also be used to create rigorous documents and views, e.g., floor plans, site plans, elevations, and cross sections. Also, they enable easy distribution, storage, and visualization of designs as digital files. Nevertheless, these types of tools share some of the problems of the pen and paper approach. For example, if there is an error in a section of the floor plan that needs to be corrected, then all other drawings need to be manually updated, due to the lack of connection between them. Also, most CAD tools only store geometrical information, thus relevant information is missing, such as structural details and the overall cost - essential elements for the construction stage. Another issue with storing only geometrical information is that CAD tools are unable to distinguish the various constructive design elements, such as doors, walls, or windows, and are unable to detect errors such as unintended misalignments or element collisions, which may cause problems during construction, e.g., unanticipated field costs and delays [Eastman et al., 2008].

Another important type of tools, created to address the problems of CAD tools, are Building Information Modelling (BIM) tools. BIM tools are capable of holding all information relevant to the overall architectural design process in a coordinated way, such that changes in one view are automatically updated in all views [Eastman et al., 2008]. With BIM tools, all data required for analysis and other tasks, including the documentation for all stages, can be generated from a single model. These new features have prompted the adoption of BIM tools, even causing some governments and local authorities to deem BIM models mandatory for the approval of new constructions. In order to guarantee these features, BIM tools require a team of architects and engineers to work in the model and in an environment thought to enable the modeling of only feasible structures, which limits the creative freedom of the architects during the early and exploratory stages of the design process. During these stages, the architects' initial designs might change extensively, depending on the suggestions made by the engineers and other team members. Constantly changing a design solution in a BIM environment is time-consuming, since it requires updating a large amount of data, which is why several architects prefer to use less rigorous

but more flexible CAD tools during initial design stages.

A common issue that affects both CAD and BIM tools regards repetitive change. For example, in a design of a space frame which follows a sinusoidal shape, as shown in figure 1.3, changing the amplitude of the sinusoidal shape should be an easy task. Unfortunately, both types of tools recognize the geometrical aspects of the design, but lack information about the overall concept of a sinusoidal shape. Because of this, if architects wish to alter the sinusoidal function, they must change the position and size of each of the bars in the space frame, which is extremely time-consuming. To support these types of changes, while also attempting to maintain the advantages of both CAD and BIM tools, architects may follow the Algorithmic Design (AD) paradigm.

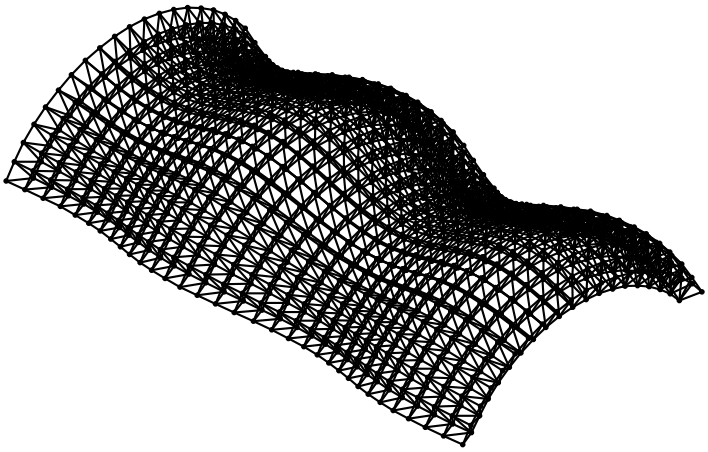

Figure 1.3: A space frame with a sinusoidal shape.

The AD paradigm can reduce the time taken by complex changes and design creation. In this paradigm, the architect specifies a parametric algorithm which is capable of generating the design, based on the supplied parameters [Terzidis, 2006]. For the sinusoidal space frame, the specified algorithm may be simple commands to generate N bars in a loop, where the position and height of each bar depends on the supplied sinusoidal function and the current iteration of the loop. This approach can effectively create a design based on the given parameters, enabling even more complex changes, e.g., modifying the sinusoidal shape, or changing from a sinusoidal to a square function, by simply modifying the algorithm, and not the bars themselves.

The ability of AD to facilitate the generation of a design also introduces the possibility of generating several variations of the same design with ease, giving architects creative freedom to explore the space of possible designs and choose the one they prefer. Figure 1.4 shows an example of the AD approach applied to the Astana National Library, a three-dimensional mobiüs strip designed by BIG architects[2]. In this case, an algorithm capable of creating the library was specified, and then different parameters were used. All of the designs were generated simply by altering the value of the parameters, and did not

[2] http://www.big.dk/#projects-anl, accessed 20 April 2018.

require additional manual changes [Branco and Leitão, 2017].

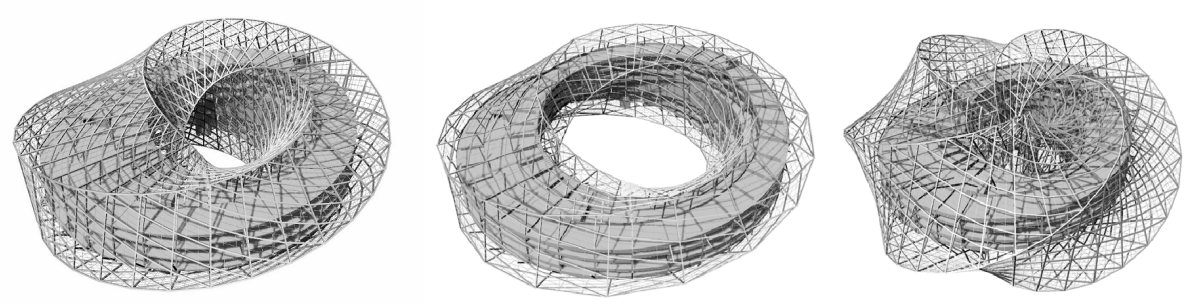

Figure 1.4: Original design of the Astana National Library (left) and variations created with AD (center and right). Adapted from [Branco and Leitão, 2017].

The problem of AD is that although CAD and BIM tools may have basic operations defined, such as creating a line given a size and position, or creating a bar given width, thickness, material, and position, they typically do not provide an environment where an architect can easily define his algorithm. In recent years, several tools have been designed specifically with this paradigm in mind. These tools have several differences among them, but one of the most relevant is the representation of the algorithm either as a flow of connected components (visual programming), where each component represents a specific architectural element or a function, or using a formal language (textual programming). This dissertation focuses on two specific AD tools, Grasshopper3D and Rosetta, which follow the visual and textual programming approaches, respectively.

Grasshopper3D[3] is a plugin for the Rhinoceros3D CAD tool[4] and is the most popular AD tool among architects [Cichocka et al., 2017a]. Both Grasshopper and Rhinoceros are developed by Robert McNeel & Associates. Rhinoceros natively supports textual programming with Python and its own custom programming language, Rhinoscript. Alternatively, Grasshopper follows the visual programming paradigm. Its simplistic approach is intuitive and easy to use with little practice, which may justify its popularity. As a result of its popularity, several tools and plugins have been developed for Grasshopper, some of which allow for analysis and optimization, which will be discussed on sections 1.2 and 1.3. The problem of visual programming tools, such as Grasshopper, is that the visual approach does not scale well. Figure 1.5 shows one example of an algorithm for a complex design, specifically a large tennis center. In this case, it is clear that creating or changing the algorithm may be very difficult, due to the amount of existing connections and dependencies between the different components.

Rosetta is an AD tool created with the ideas of interoperability, portability, and textual programming, which allows its users to program in several different programming languages, e.g., Racket, Python, and JavaScript, and to generate models on several CAD or BIM tools [Lopes and Leitão, 2011]. Rosetta is an independent tool which provides commands, i.e., a set of functions which can be invoked as an

[3] http://www.grasshopper3d.com/, accessed 25 April 2018.
[4] https://www.rhino3d.com/, accessed on 25 April 2018.

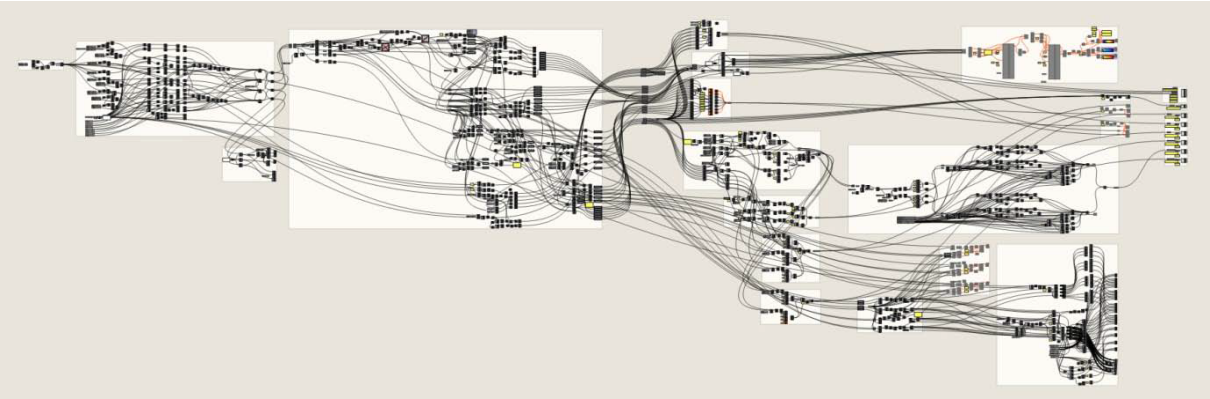

Figure 1.5: The complete Grasshopper algorithm for the Hangzhou Tennis Center. Adapted from [Miller, 2011].

Application Programming Interface (API) by its users. Its ability to be used with several design tools is due to the implementation of different tool-specific backends, which implement the same overall API. Therefore, by simply altering the value of a parameter corresponding to which design tool to use, the same algorithm in Rosetta can generate a design in other tools [Branco and Leitão, 2017]. Also, by using known and already established programming languages, the Rosetta tool allows its users to benefit from native language constructs, such as loops or lists, facilitating the definition of the algorithm, especially if the user is already acquainted with the language. Figure 1.6 shows one example of using Rosetta with the Racket programming language to create a design on several design tools.

Overall, the AD paradigm is advantageous over other approaches, since it allows not only the definition of parametrical designs, but also because it enables the automatic generation of several variations of a design. These properties are important in order to create the automated optimization framework for the architectural design process which is discussed in chapter 3. In the next section, the second stage of the process, the analysis stage, will be explored.

1.2 Analysis Stage

In the common workflow of an architectural project, the architect is responsible for creating a design which is then given to the engineering team. The engineers perform calculations to ensure, for instance, the structural stability of the project. If there is an issue with the design, they must inform the architect and make suggestions, in order to resolve it. This process is repeated until there are no more issues with the design.

Due to the complexity and costs usually associated with buildings, the role of the engineer is not only to ensure that a particular building complies with the regulations, but to also assess how efficient it is. For example, it is not a good solution to have a building that is structurally sound but that costs much more than what is needed. As a result, one important aspect of the engineer's job is the evaluation

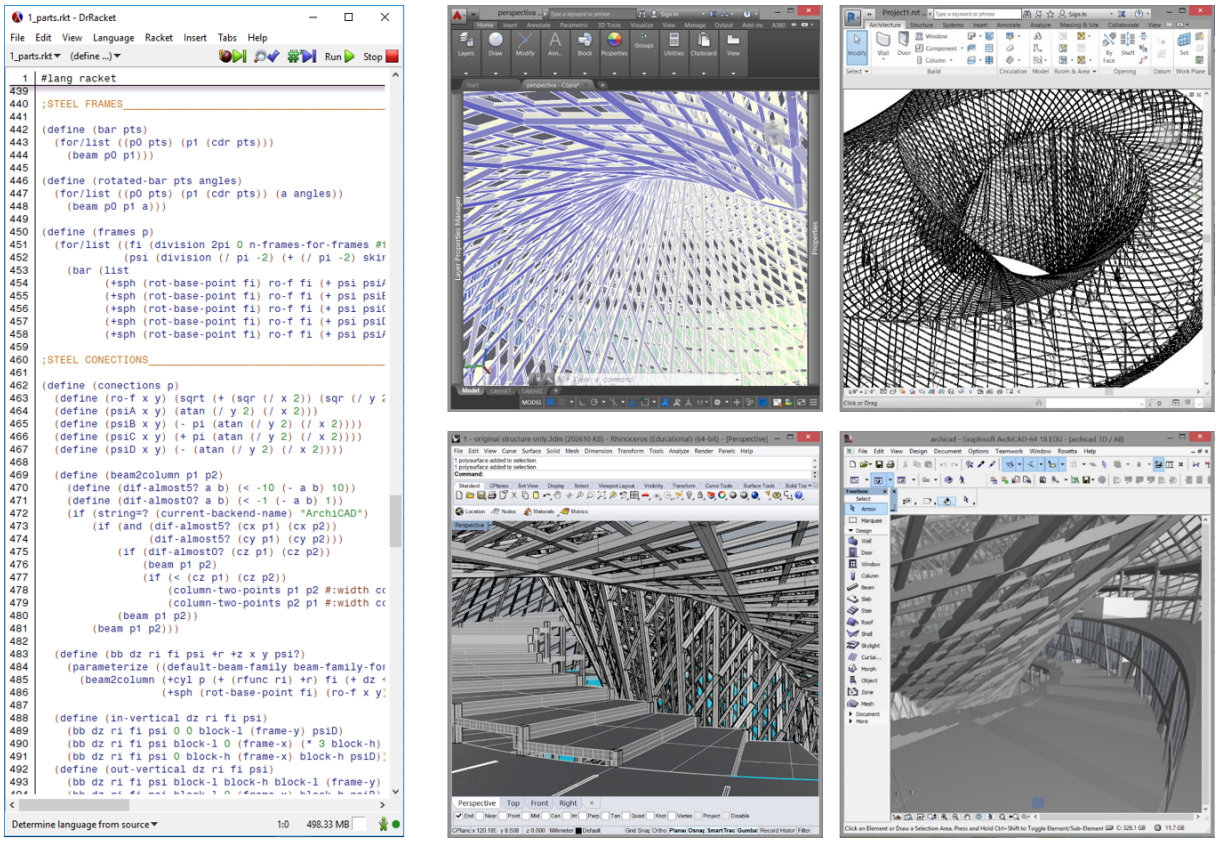

Figure 1.6: An example of using Rosetta to generate a design on several design tools. Adapted from [Branco and Leitão, 2017].

of the building's performance regarding different criteria, e.g., structural, acoustic, energy consumption, and cost.

The increase in computational power has led to the adoption of numerical analysis and simulation-based tools to calculate performance. These analysis tools are required because there is no known mathematical expression capable of providing, for example, the energy consumption of a design. These tools are able to calculate those values using approximations, such as raytracing and finite element analysis. Therefore, using these tools is mandatory to perform optimization, as it is the only way of obtaining performance values.

The most requested types of analysis are daylight availability and structural analysis [Cichocka et al., 2017a]. Radiance [Ward, 1994] is a lighting simulation system which is capable of performing raytracing by combining both deterministic and stochastic techniques for its calculations. By using this tool, architects can know the amount of solar radiation across their entire design, and make informed decisions based on these results. Figure 1.7 shows one example of the results of Radiance on the Astana National Library. In this case, the radiation map produced by the tool was useful to determine the position and size of solar panels placed across the outer surface of the structure.

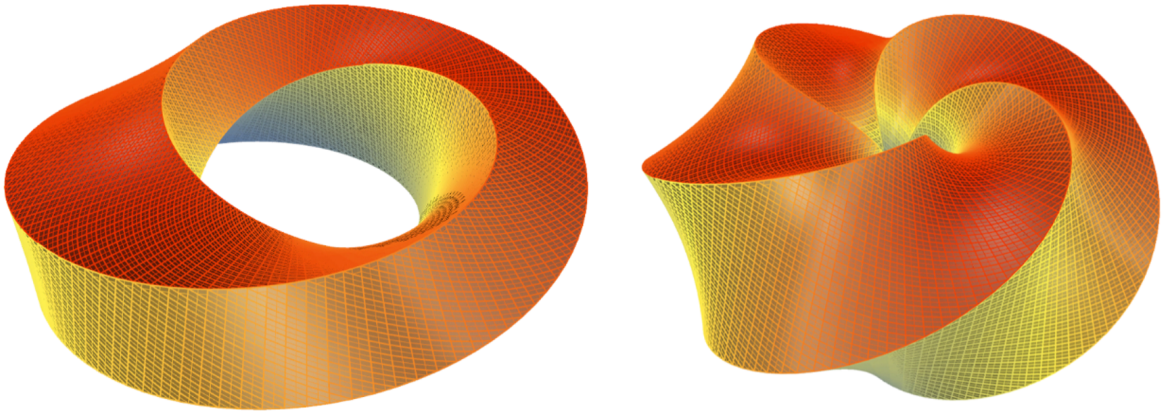

Figure 1.7: An example of the radiation map obtained by using the Radiance analysis tool on two variations of the Astana National Library. Adapted from [Aguiar et al., 2017].

Robot[5] is a professional structural analysis tool developed by Autodesk. This tool is able to simulate several types of loads on a structure, such as dead loads caused by its own weight, live loads, and environmental loads, and it can calculate stress and displacement, among other performance criteria. The tool itself was created mainly for structural engineers and it is difficult for most architects to gain information about their design if they use it themselves. Figure 1.8 shows one example of using Robot, in this case to obtain the tensions at each bar of a space frame. By using this information, it is possible to understand that the regions marked in red are suffering a larger amount of tension, i.e., they are the more problematic sections of the design.

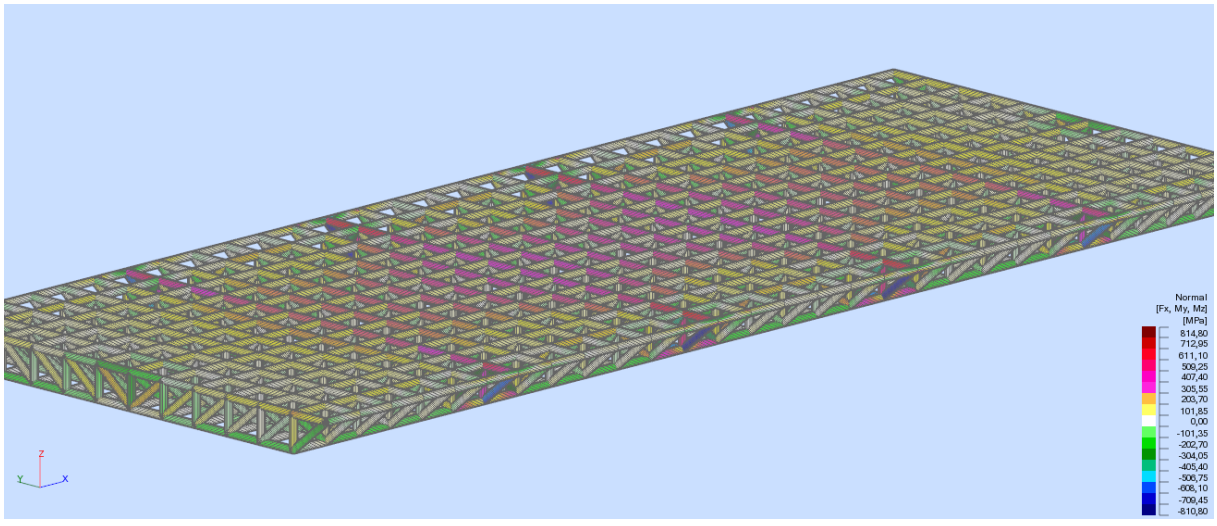

Figure 1.8: An example of the tension map obtained by using the Robot analysis tool on a space frame.

These analysis tools have been used and independently validated on several projects, therefore their results for a specific design can be trusted, yet they are not frequently used in the early design stages.

[5] https://www.autodesk.com/products/robot-structural-analysis/overview, accessed 26 April 2018.

This may be due to the fact that the information which is usually available in geometrical 3D models, i.e., the common designs discussed so far, is not enough. These tools require the creation of an analytical model containing all the information necessary to perform their analysis. Figure 1.9 shows an example of three different types of models from the Shenzhen Bao'an International Airport Terminal 3[6]: on the left, a 3D model containing geometric information; on the center, an analytical model for radiation analysis with information regarding materiality and reflectivity of the building's surfaces, as well as the sensor nodes' location to evaluate the incident radiation; and on the right, an analytical model for structural analysis, with the geometry of the structural elements, represented by lines and nodes, as well as data on material properties, cross-sections, supports, and loads.

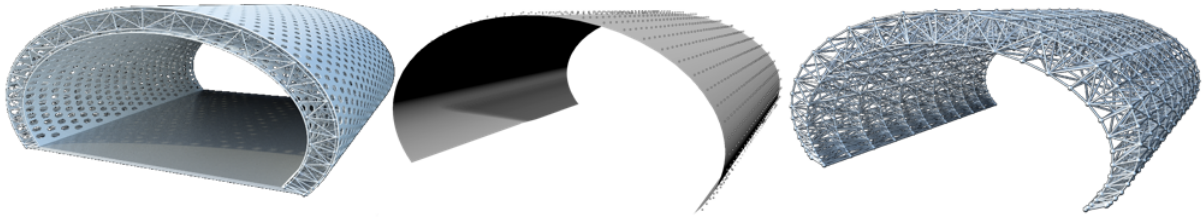

Figure 1.9: Diferent models of the Shenzhen Bao'an International Airport Terminal 3. Geometrical 3D model (left), analytical model for radiation analysis (center) and analytical model for structural analysis (right). Adapted from [Aguiar et al., 2017].

Analytical models can be generated in BIM tools that have that capability or by directly using the API of the analysis tool. The former method can be easily employed by architects, while the latter is more complex and difficult to apply. To enable the more complex approach, architects can use plugins in their development tools to generate the analytical model from the geometrical one. For example, users of the Grasshopper AD tool can use the Ladybug plugin [Roudsari and Pak, 2013] to enable radiation analysis using Radiance, or the GH2Robot [Christensen et al., 2014] plugin to enable structural analysis using Robot. Another approach involves the creation of analysis tools inside the design tool itself, as is the case of Grasshopper's Karamba plugin [Preisinger and Heimrath, 2014], which can be used to generate a structural analytical model and analyze it within Grasshopper.

The problem of the approaches discussed so far is that for each analysis tool more than one plugin may exist, and these plugins may have different interfaces, forcing architects to adapt the connection of their algorithm to each plugin. This adaptation goes directly against the advantages of using AD, i.e., no changes should be required in the overall algorithm to use different analysis tools. Recent approaches attempt to solve this problem by giving the AD tool the ability to generate different analytical and 3D models from the algorithm provided by the user, in a approach named Algorithmic Design and Analysis (ADA) [Aguiar et al., 2017]. Figure 1.10 shows the ADA workflow in which the user, i.e., the architect, needs only to define an algorithm for the design and select the analyses to perform, leaving

[6] http://fuksas.com/?p=196, accessed 2 February 2019.

the rest to the AD tool, which ensures each analysis tool receives the correct model. The Rosetta AD tool follows this approach in a similar way to how it enables the usage of several CAD and BIM tools, i.e., its API is fixed, and to generate a specific analytical or geometrical model the architect simply needs to change a single parameter.

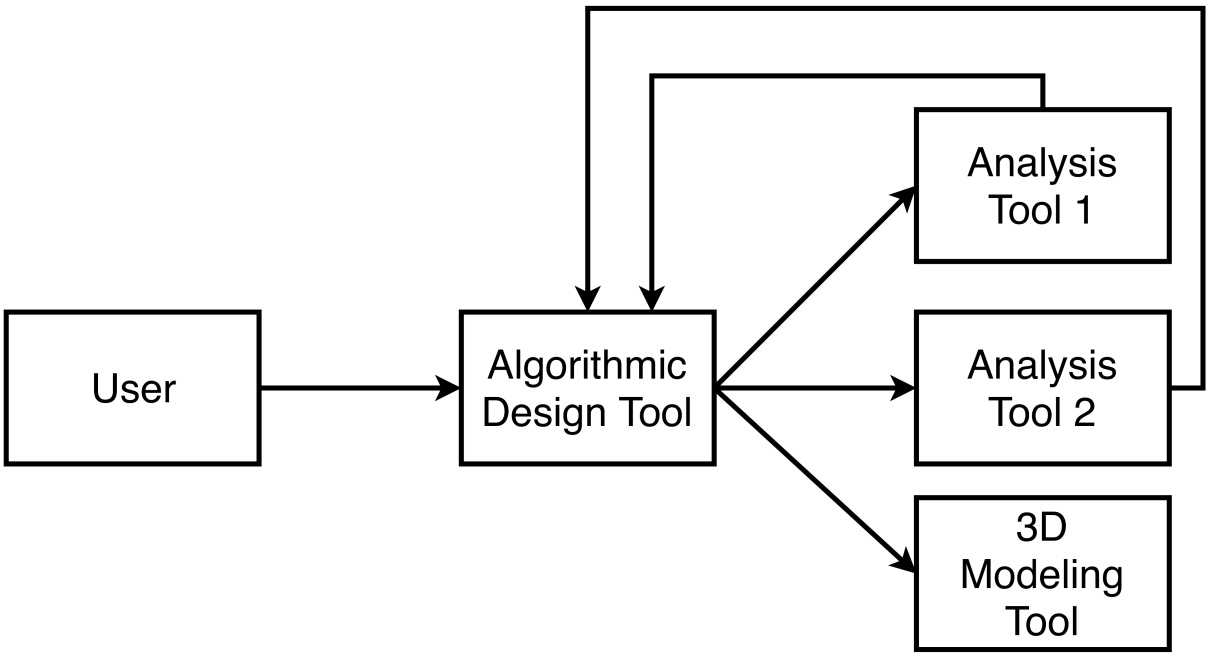

Figure 1.10: The ADA workflow, where the AD tool is capable of generating geometrical or analytical models from the same algorithm.

Since the AD approach already enables the automatic generation of several variations of the design, the ADA approach enables the automatic generation and analysis of different variations of the design [Aguiar et al., 2017]. Using this, it is possible to find a cheaper design, a design with better lighting or heating conditions, or a design with less structural risks, early on in the project's development. To that end, the architect can use AD to generate different variations of the design, and then use the analysis tools to evaluate those variations, selecting the ones that have better performance. Despite these advantages, the number of analyses performed in a typical architectural project is quite small. One of the main reasons for this is that each simulation run by an analysis tool takes too much time. Simple designs may take only a few minutes to analyze, but more complex designs can take several hours or even days. Architectural projects suffer heavily from unexpected delays, hence this type of approach, despite being commonly desired and often needed, is rare in small and medium scale projects.

With the techniques described in this section, it is possible to generate several variations of a design and find the best one automatically, but this still requires several interventions by the architects, namely in the generation stage, since they must choose the values of the parameters for the variations. In the next section, the optimization stage will be explored, and it will be explained how it can help to automate

this process.

1.3 Optimization Stage

One advantage of the ADA approach is that it makes it possible to fully automate the generation of variations, the corresponding analyses, and the selection of the better-performing solutions. It also enables the usage of mathematical optimization in architecture. In this scenario, an optimization process would be responsible for guiding the generation of different designs, returning the best design in regard to a set of metrics. Approaches such as ADA could be modified so that an optimization algorithm generates and finds the best designs, instead of the user manually generating variations. However, to understand how this is possible, it is necessary to understand exactly what optimization is.

In the field of mathematical optimization, an optimization problem is commonly described by the following set of equations [Koziel and Yang, 2011]:

$$\text{minimize} \quad f_i(x),\ (i = 1, 2, \ldots, I) \tag{1.1a}$$

$$\text{subject to} \quad h_j(x) = 0,\ (j = 1, 2, \ldots, J), \tag{1.1b}$$

$$g_k(x) \leq 0,\ (k = 1, 2, \ldots, K) \tag{1.1c}$$

In simple terms, optimization is defined as the process of finding which values for a set of parameters yield the best performance result in regard to specific criteria. Typically, the performance is represented as a function (Single-Objective) or a set of functions (Multi-Objective), also called fitness functions, as defined by equation 1.1a. The goal of the process can be to either obtain the maximum or, as shown in equation 1.1a, minimum possible value of the function. To reach the goal, an optimization algorithm changes the parameters, in this case the vector x, according to a set of restrictions, as defined by equations 1.1b and 1.1c. Besides the restrictions, the parameters themselves may have bounds, i.e., minimum and maximum possible values.

In order to use the knowledge and ideas from this field of mathematics for the benefit of architectural projects, it is required to translate an architectural optimization problem into a mathematical optimization problem. The definition of optimization contains three important parts: the parameters, the performance, and the actual process of finding values for the parameters. To illustrate how these parts can be identified in an architectural design, the Meiso no Mori Crematorium[7], shown in figure 1.11, will be used as an example.

The parameters defined in an AD approach can be used for optimization, e.g., in the case of Meiso

[7] http://www.toyo-ito.co.jp/WWW/Project_Descript/2005-/2005-p_07/2005-p_07_en.html, accessed 2 February 2019.

Figure 1.11: The Meiso no Mori Crematorium in Japan, designed by Toyo Ito and Associates. The building's roof has a non-uniform shape, similar to a cloud, and its supporting pillars are assymetrically placed.

no Mori, the parameters could be the position of the pillars or their heights. The parameters should be bounded, i.e., there should be a minimum and maximum position or height, and restrictions could also be applied to prevent unfeasible designs, such as overlapping pillars. Regarding the performance, it is typically described as a function, and it could be, for example, the maximum vertical displacement of the roof when subjected to its own weight. As discussed previously on section 1.2, it is generally impossible to write an expression that would yield this value, given the parameters. This is why analysis tools need to resort to simulation techniques, such as raytracing or finite element analysis, to approximate results. In this case, to calculate the maximum vertical displacement, the Robot analysis tool could be used. These problems, where there is a lack of a closed-form expression of f, are a special case of optimization, labeled both simulation-based and black-box optimization, in which the function is treated as a black-box that can be used for optimization. Finally, the actual process of finding values for the parameters varies based on the chosen algorithm, and is described more in depth on section 2.1.

With this description of the problem, we are now able to perform optimization on an architectural design, according to the following generic pseudocode:

Algorithm 1.1: Generic Optimization Algorithm for Architecture

```
initializeBestResult()
while not isFinished do
    x ⟵ selectNewParameters()
    result ⟵ generateAndEvaluateDesignInSimulator(x)
    if result < bestResult then
        bestResult ⟵ result
        bestParameters ⟵ x
    end
    isFinished ⟵ checkTerminationConditions()
end
return bestResult, bestParameters
```

With this approach, existing optimization algorithms can be used directly in architectural problems. The problem of optimization is, similarly to analysis, the time cost. Each individual analysis can take a considerable amount of time, and during an optimization process hundreds of analyses can be executed. This time cost is feasible for small and medium scale projects, because the analyses take less time, but for large scale complex projects, which benefit the most from optimization, the time cost can be prohibitive. Because of this time cost, the selection of which algorithm to use is quite important and is one of the main focuses of this dissertation.

1.4 Dissertation Objectives

This dissertation considers the problem of architectural design optimization, namely Single-Objective optimization. Although Multi-Objective optimization is used when multiple performance criteria exist, this dissertation focuses only on Single-Objective optimization, since it requires less evaluations, which is relevant when considering the time needed to perform the optimization. Also, Multi-Objective problems can be transformed into Single-Objective problems using the weighted sum method, where a weight is assigned to each objective, and the weighted sum is used as a single objective [Marler and Arora, 2009].

The goal, then, is to explore and study existing Single-Objective optimization algorithms and how they can be applied to architectural problems, specifically, how to automate an architectural design optimization process. The dissertation will focus extensively on existing algorithms, their properties, and their classification. This is because, according to the No Free Lunch (NFL) theorems, no algorithm can consistently perform better than the others on all problems [Wolpert and Macready, 1997]. Therefore, the optimization framework proposed in this dissertation also allows the usage of several different types of algorithms for the same architectural design problem. To evaluate this framework, a prototype optimizer module will be created and test with different case studies.

Finally, we will draw conclusions from studying the different algorithms and from the evaluation of the prototype, regarding which types of algorithms may be preferable on these types of problems. This

also involves studying the existing beliefs of the architectural community regarding design optimization, discussed in section 2.2, specifically the belief that Genetic Algorithms (GAs) are the best algorithm to use.

2

Optimization

Contents

This chapter focuses on the most relevant stage of an automated architectural design optimization process - the optimization stage. In this stage, one or more algorithms are used to find the values for parameters which yield the best performing design. The problem of architectural optimization is the time cost, i.e., each evaluation of a set of parameters takes a long time, because simulation-based analysis tools must be used. Therefore, it is important to study existing algorithms and their ability to yield good results using few evaluations.

Although the field of mathematical optimization has a rather large set of algorithms which have been proposed over the years, this dissertation focuses mainly on algorithms which can be directly used in an architectural optimization problem. By studying the properties and the different classes or types of algorithms, the goal is to understand how these properties may influence the algorithms' ability to produce good results in these problems.

Besides the algorithms, which are discussed in section 2.1, this chapter also focuses on studies which have been performed in architectural optimization, as well as current standards and practices. This analysis is done in section 2.2, and enumerates a few standards which will be tested with a prototype optimizer described on chapter 4.

2.1 Optimization Algorithms

In the field of mathematical optimization, several strategies for optimization have been considered over the years, which has resulted in the creation of several optimization algorithms with different properties, advantages, and disadvantages. Despite this, most users often choose the simplest approaches because they are easier to understand and implement [Conn et al., 2009]. However, to achieve better results, the algorithm should be selected considering its properties and the type of problem. Therefore, to allow choosing the appropriate algorithm for each problem, existing algorithms and their properties will be studied.

Optimization algorithms may be classified according to several different properties. One possible way of classifying algorithms is according to their determinism. Some algorithms may be given a starting point, or initial guess, from which they will begin their path towards a solution. Deterministic algorithms are sequential algorithms that always follow the same sequence of steps, thus returning the same value given the same starting point. On the contrary, stochastic algorithms include some form of randomness, i.e., the sequence of steps may be random, therefore they may return different values for the same starting point. The degree of randomness is important when analyzing stochastic algorithms. A completely random algorithm will most likely achieve very different results in different runs, while a more conservative algorithm, which takes only small random steps, can achieve similar results in different runs.

Besides their deterministic or stochastic properties, another way of classifying algorithms is according to their mobility. Global optimization algorithms attempt to find the best solution across the entire solution space, while local ones attempt to find the best solution only within a region of that space. However, local algorithms can be adapted for global optimization by using stochastic strategies, such as random starting points or random steps [Koziel and Yang, 2011]. While the previous definitions may appear to imply that global algorithms should always be chosen if the goal is to find the overall best value, in practice this statement is not always true. The problem is that in order to find the best global value, global algorithms conduct an exhaustive search, which requires excessive evaluations. On the contrary, local algorithms typically converge faster to a local solution, i.e., they require less evaluations. Therefore, if previous knowledge of the problem suggests that finding a local solution is sufficient, then it would be better to use local algorithms.

Another interesting property to analyze is if the algorithms themselves have parameters, e.g., to specify the probability of taking certain steps or to define some internal step of the algorithm. These parameters which are set by the user before starting the algorithm are called hyperparameters. Algorithms with a large amount of hyperparameters are more general. The issue is that the process of fine-tuning these hyperparameters for a specific problem is itself an optimization problem. When dealing with costly function evaluations, it is unfeasible to test several variations of these hyperparameters for a single algorithm.

Another possible classification considers if the algorithm takes advantage of information regarding the derivatives of the function. Derivative-based algorithms use the partial derivatives of a function, the gradient, to discover the direction of greatest increase of the function. They may also use the actual value of the gradient, which indicates the slope of the function, i.e., how much the function increases. With this information, it is possible to discover the best path to follow in order to reach a local solution. In the case of black-box functions, i.e., functions which do not have a closed-form expression, the derivatives of the function are unavailable, therefore it is impossible to obtain the gradient. One possible option would be to approximate the derivatives with the finite-difference method. However, for functions with costly evaluations, that option is unfeasible, since it would require several extra evaluations. Therefore, for costly black-box problems, derivative-free algorithms are the only feasible option.

Regarding derivative-free algorithms, in their optimization textbook, Conn, Scheinberg, and Vicente consider two classes of deterministic derivative-free algorithms: direct-search and model-based algorithms [Conn et al., 2009]. A third class, called metaheuristics, is only briefly mentioned, despite being one of the most widely used and highly cited classes of derivative-free algorithms [Koziel and Yang, 2011, Hare et al., 2013, Rios and Sahinidis, 2013, Wortmann et al., 2017]. The authors label metaheuristics "methods of last resort (...) applicable to problems where the search space is necessarily large, complex, or poorly understood (...)" [Conn et al., 2009]. In a recent review of derivative-free algorithms,

Rios and Sahinidis also consider the existence of direct-search and model-based algorithms, however they consider that these methods may be deterministic or stochastic [Rios and Sahinidis, 2013]. Given the contradicting classification approaches, this dissertation follows the approach of [Wortmann et al., 2015], and considers three classes of derivative-free optimization algorithms: direct-search, metaheuristics, and model-based algorithms. Of these three classes, direct-search algorithms are deterministic, metaheuristics are stochastic, and model-based algorithms may be deterministic or stochastic.

The following subsections discuss the classes of algorithms that will be used on this dissertation, namely sampling, direct-search, metaheuristics, and model-based algorithms, and also provides examples of specific algorithms. Table 2.1, at the end of this section, provides a summary of the presented algorithms and their properties.

2.1.1 Sampling algorithms

Sampling algorithms are algorithms that select a subset of a discrete population or a set of points from a continuous search space. These algorithms are simple and fast, since the selection of each point does not consider previous results. Because of how easy they are to implement and use, these algorithms are commonly used as a baseline or initial approach. In this type of approach, a sampling algorithm selects several sets of parameters to evaluate and returns the best performing set. However, sampling algorithms are not true optimization algorithms, they are only methods for selecting possible points, and should be used only as an initial approach or as an internal step of another algorithm.

Figure 2.1 shows an example where nine points were sampled from a two-dimensional space using three different sampling procedures, namely Random, Grid, and Latin Hypercube sampling. In Random sampling (left), points are chosen at random, which can lead to large gaps of unexplored possibilities, such as the squares on the bottom left, top center, and top right. Grid sampling (center) divides the space in equal sections, according to the number of sample points. By dividing the space in equal sections, this approach always selects points at the same distance, which may be quite large depending on the number of sample points, and leaves other possibilities unexplored. Latin Hypercube sampling (right) attempts to mix both approaches by dividing the space into equal section grids and choosing a random sample within each grid [McKay et al., 1979]. This approach introduces randomness and variability while also sampling each grid once, which makes it one of the best approaches.

2.1.2 Direct-search algorithms

Direct-search algorithms are deterministic derivative-free algorithms which choose their next action using a set of points that are sampled directly from the function at each iteration. Due to being deterministic, several of these algorithms have proven convergence capability, given a few conditions regarding

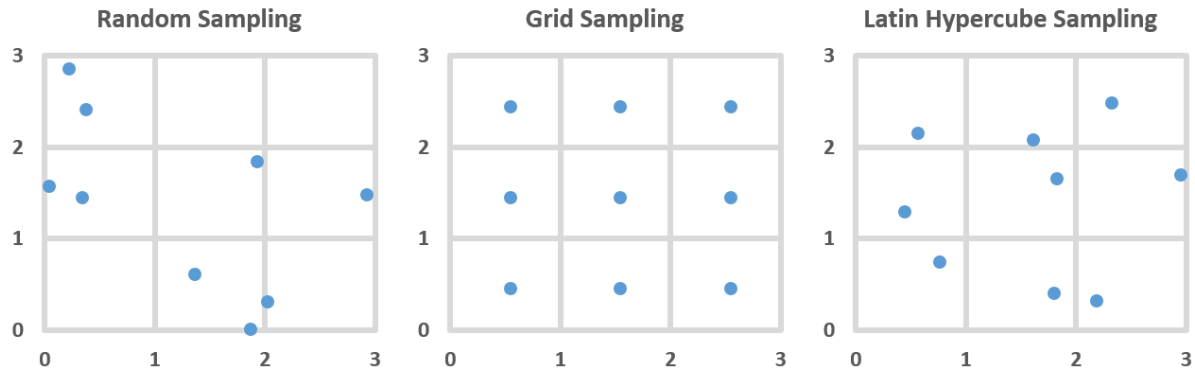

Figure 2.1: Example of sampling nine points using three different sampling algorithms: Random, Grid, and Latin Hypercube sampling.

the function. Despite this, direct-search algorithms have not been studied as much recently, when compared with the other derivative-free algorithm classes.

There are several interesting examples of direct-search algorithms. The Nelder-Mead algorithm [Nelder and Mead, 1965], probably the most famous of this category, manipulates a simplex over the search space in its attempt to find the best result. A simplex is a generalization of a triangle to arbitrary dimensions, e.g., a triangle in two dimensions or a tetrahedron in three dimensions. By taking into account the performance of each vertex, the algorithm performs reflection, expansion, contraction, or shrinking operations over the simplex. In each iteration, the algorithm typically requires only one or two function evaluations, meaning that each evaluation has a large direct impact in the search for the best result. However, since the algorithm follows a path depending only on the performance of each vertex, it is only a local optimization algorithm.

For several decades after its introduction, the Nelder-Mead algorithm was the algorithm of choice for any complicated optimization problem. This popularity influenced the creation of other algorithms, such as the Subplex algorithm [Rowan, 1990]. The Subplex algorithm attempts to resolve some of the issues of the Nelder-Mead algorithm, namely its difficulty to adapt to higher dimensional problems. To do this, the Subplex algorithm divides the search space into several subspaces and applies the Nelder-Mead algorithm on each subspace. Since each subspace has less dimensions than the original problem, the algorithm preserves the features of the Nelder-Mead algorithm, while improving its performance on higher dimensions. However, since this algorithm is simply applying the Nelder-Mead algorithm on different subproblems, it is still a local optimization algorithm. Another similar algorithm is the Principal Axis (PRAXIS) algorithm [Brent, 1973]. Instead of subdividing the space, this algorithm applies line search to each dimension, and uses the information of the best solution in each dimension to determine its next step.

Besides the algorithms discussed so far, there are other direct-search algorithms, including al-

gorithms for global optimization. One of the most well-known examples is the Dividing Rectangles (DIRECT) algorithm [Jones et al., 1993]. The DIRECT algorithm uses center-point sampling, a strategy which divides the space into three equally sized intervals, with the center of the overall space being the center of the middle interval. In arbitrary dimensions, these intervals are hyperrectangles which are sampled across all dimensions. The algorithm then proceeds to select the most promising hyperrectangle based on the obtained samples, and subdivides that space, following a similar procedure as before. If the algorithm has not improved its best result by a certain amount, then it stops searching the current hyperrectangle and chooses one of the previously unexplored hyperrectangles. The DIRECT algorithm is a simple algorithm which can be applied to most cases and performs global optimization with convergence guarantees, given a sufficiently high evaluation limit. Based on the original algorithm, another algorithm named DIRECT-L was also created [Gablonsky and Kelley, 2001]. The DIRECT-L algorithm is quite similar to DIRECT but it is more biased towards local search, i.e., it explores each hyperrectangle further, before deciding to jump to a previous one, in an attempt to find a solution faster.

Overall, this class of algorithms is not as popular as the others, but its convergence proofs make it a very interesting and promising choice for architectural optimization.

2.1.3 Metaheuristics

Metaheuristics are stochastic algorithms that also do not require derivatives. These algorithms are inspired by aspects of Nature, such as natural selection, evolution, and swarm intelligence. Metaheuristics are popular due to the fact that they can be applied to almost any problem and are the focus of active research, despite the fact that most do not offer convergence guarantees [Koziel and Yang, 2011].

Several examples exist, with the Simulated Annealing (SA) algorithm being one of the most well-known classical metaheuristics [Kirkpatrick et al., 1983]. The SA algorithm at each iteration randomly selects and evaluates a point in the solution space, i.e., a set of values for the parameters. Then, depending on the difference between the quality of the current point and the new point, the algorithm assigns a probability to the event of changing to the new point and changes based on that probability. Also, the algorithm has a parameter, known as temperature, which affects the probability of changing state at each iteration. At the start of the search, the temperature has an high value, so the algorithm is biased towards changing state, while as time goes by and the temperature decreases, the algorithm is biased towards maintaining good solutions. The ability to choose what appears to be worse solutions is helpful to guide the algorithm towards the global optimum, instead of only focusing on its current solution. However, the initial value and decay rate of the temperature parameter should be carefully tested and set depending on the problem, which, as discussed previously, is not possible for costly optimization problems.

Another example of metaheuristics are the random search algorithms, such as Controlled Random

Search 2 (CRS2) [Price, 1983]. The CRS2 algorithm considers a set of possible solutions, called a population. The size of the population is a parameter which must be carefully chosen for each problem: if it is too large, the algorithm may spend all of its evaluation budget initializing the population, while if it is too small it might not properly sample the solution space. Similarly to the Nelder-Mead algorithm, the CRS2 algorithm creates a simplex, but it is a simplex generated from randomly selecting points of the population and always including the best solution. A new trial point is generated from the simplex and if it is better than the worse solution of the population, it replaces it. Over time, the entire population should cluster around a solution. While the algorithm was able to obtain good results in practice, it is highly dependent on random steps and spends several evaluations randomly generating a population, which, in a time-consuming optimization problem, is not feasible.

The most common metaheuristic is the GA [Goldberg, 1989]. The GA takes the ideas of biological evolution and applies them to optimization. It begins by creating a population of possible solutions, similarly to CRS2. Then it selects part of the population to produce offspring, i.e., new possible solutions. This offspring may still suffer a mutation step, i.e., be randomly changed, and then the population is updated. Although simple in concept, the GA has several internal steps which have been implemented in dozens of different ways. These steps are selection, crossover (offspring generation), and mutation. Each step can also include several hyperparameters. This makes the GA very general, i.e., it can be used in almost any problem. However, if the internal steps and population size are not carefully selected, the algorithm may take several more evaluations to converge, which is problematic for time-consuming optimization problems. The GA considered in this dissertation and for the framework described in chapter 3, uses tournament selection, blend crossover, and polynomial mutation. In tournament selection, I individuals are chosen at random and the best individual is selected. This procedure repeats K times, until all parents are selected. In blend crossover, the values of both individuals are added using random weights that add up to one, to prevent values outside the range of the parent values. In polynomial mutation, the value of each parameter is changed using a random polynomial function.

Several algorithms have been created by refining the evolution principles of GAs. One recent idea is the Evolutionary Strategy with Cauchy Distribution (ESCH) algorithm [Santos, 2010], which replaces the typical mutation procedure of GAs with a procedure which relies on the cauchy distribution for random number generation. The Improved Stochastic Ranking Evolutionary Strategy (ISRES) algorithm makes even larger changes and removes some of the components of GAs, such as crossover and mutation, relying on improved selection methods and probabilistic generation approaches instead [Runarsson and Yao, 2000, Runarsson and Yao, 2005]. The Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) algorithm is similar, but instead relies on a normal distribution which can be sampled to generate solutions [Hansen and Ostermeier, 2001]. The mean and variance of the distribution are updated on each iteration, given the results of the population, such that over time the distribution is

tightly focused around a global optimum.

Another well known metaheuristic is the Particle Swarm Optimization (PSO) algorithm [Kennedy and Eberhart, 1995]. Similarly to the GA, this algorithm takes inspiration from Nature, but instead of using the ideas of evolution it uses the ideas of social behavior and influence. The algorithm contains a population of candidate solutions, called particles, which is initialized uniformly across the search space. During the process, each particle will store its best solution and the overall best solution found so far. At each iteration, the velocity of the particles is updated based on their best solution and on the swarm's best solution using random weights which decide the contribution of each solution. Furthermore, this update depends on three hyperparameters set at the start of the optimization, which are also used to control the contribution of the previous velocity and best known positions. After recalculating the velocity, each particle moves and the best known positions are recalculated. Eventually, the entire swarm should converge over a global optimum.

Overall, metaheuristics are stochastic algorithms which apply strategies to combine random search and guided search, the former being used to explore the search space avoiding local optima and the latter being used to exploit known information about the solutions found so far. The most popular algorithms being used nowadays are the GA and the PSO algorithm. While both are good general algorithms with extensive uses, their reliance on random number generation and hyperparameters poses a risk - with specific random numbers and hyperparameters, these algorithms may quickly find a great result, but with another set of numbers and hyperparameters their performance could be entirely different. This is an even greater risk on costly black-box problems, such as architectural optimization, since the optimization algorithm should be able to find a good result with few evaluations without requiring intervention from the user to set the hyperparameters.

2.1.4 Model-based algorithms

Model-based methods create and update a high-fidelity surrogate model which is used to guide the optimization process [Rios and Sahinidis, 2013]. This surrogate model is an approximation of the unknown black-box function, i.e., a model which can be used to estimate the performance value of a given set of parameters. In general, most methods perform an initial sampling step to generate the model, use it to select new candidate solutions to evaluate with the real function, and then update the model with the results from the candidate solutions.

Over the years, several different strategies have been analyzed. Trust-region methods are a local model-based approach, where the model is believed to be accurate within a neighborhood, i.e., a trust region. Depending on the quality of the model, at each step the region may be expanded or contracted. The most well-known trust-region algorithms are Constrained Optimization By Linear Approximation (COBYLA) [Powell, 1994] and Bounded Optimization By Quadratic Approximation (BOBYQA) [Powell,

2009], which create linear and quadratic approximations, respectively.

Another approach employs surrogate models from fields such as Statistics and Machine Learning (ML). The goal of using such surrogate models is to speed up and improve the quality of the optimization, since using the surrogate is orders of magnitude faster than using the actual costly black-box function. One example is the stochastic response surface method proposed by [Regis and Shoemaker, 2007], which uses an adaptive sampling strategy to refine a surrogate model and uses it to find the best result. Several surrogates could be used such as cubic or linear Radial Basis Functions (RBFs) and Gaussian Processs (GPs).

Other simpler and less robust options have also been considered, such as the strawman algorithm described by [Marsden et al., 2004] as an introduction to surrogate optimization. This algorithm produces a surrogate based on an initial set of points, finds the optimal point of the surrogate, and tests that point on the actual function. The actual result of this new point is then used to update the surrogate and the process repeats. Despite not having any convergence guarantees, the algorithm is simple to create and could be used as a baseline approach.

For this dissertation, a version of the strawman algorithm was created, called the simple surrogate algorithm. The algorithm uses latin hypercube sampling to generate or update the surrogate at the start of each iteration. Then, the surrogate is optimized using the PSO algorithm and the five best solutions are evaluated using the real function. Finally, at the end of each iteration the bounds of all parameters are decreased and centered around the best solution found so far. This simple surrogate algorithm allows using any type of surrogate model. In this dissertation, common ML models were chosen, namely decision trees (tree), random forest (forest), linear regression (linear), third degree polynomial regression (polynomial), Support Vector Machine Regression (SVMR), and Multi Layer Perceptron (MLP). Each model is different and yields different results, so, for the purpose of clarity, they are considered different algorithms throughout this dissertation.

Overall, model-based algorithms prove to be an interesting approach to costly optimization. Their ability to model the costly function with a surrogate has the potential to speed up the optimization process, which is quite desirable in architectural design optimization. Plus, although there have been extensive studies in the past regarding model-based optimization, the surge of ML techniques has also increased the rate of active research into this type of optimization. Therefore, studying these algorithms is relevant for this dissertation.

Table 2.1 provides a summary of all algorithms discussed this far which are implemented in the prototype optimizer discussed in section 3.2.

Table 2.1: Summary of the discussed algorithms which are implemented in the prototype and their properties.

Class	Algorithm	Deterministic	Global
Sampling	Grid	✓	✓
	Latin Hypercube	✗	✓
	Random	✗	✓
Direct-search	DIRECT	✓	✓
	DIRECT-L	✓	✓
	Nelder-Mead	✓	✗
	PRAXIS	✓	✗
	Subplex	✓	✗
Metaheuristic	CMA-ES	✗	✓
	CRS2	✗	✓
	ESCH	✗	✓
	GA	✗	✓
	ISRES	✗	✓
	PSO	✗	✓
Model-based	BOBYQA	✓	✗
	COBYLA	✓	✗
	Elastic Net	✗	✓
	Forest	✗	✓
	GP	✗	✓
	Lasso	✗	✓
	Linear	✗	✓
	MLP	✗	✓
	Polynomial	✗	✓
	RBF-Cubic	✗	✓
	RBF-Linear	✗	✓
	Ridge	✗	✓
	SVMR	✗	✓
	Tree	✗	✓

2.2 Optimization in Architecture

With the recent advancements in AD and analysis tools, performing optimization on an architectural design is an easier task from the architect's viewpoint. This has led to optimization being an increasingly desired step in the typical design process, especially structural and daylight optimization [Cichocka et al., 2017a]. Due to the popularity of the Grasshopper tool, several optimization plugins have been developed for it. A large portion of these tools are not new algorithms or new implementations of known algorithms, instead they connect Grasshopper with existing, well-known, and tested optimization software packages.

Some examples of optimization plugins for Grasshopper are Galapagos[1], Goat[2], Silvereye[3] [Ci-

[1] http://www.grasshopper3d.com/profiles/blogs/evolutionary-principles, accessed 15 May 2018.
[2] https://www.rechenraum.com/en/goat.html, accessed 20 May 2018.
[3] https://www.food4rhino.com/app/silvereye-pso-based-solver, accessed 10 February 2019.

chocka et al., 2017b], and Opossum[4] [Wortmann and Nannicini, 2017]. In recent versions of Grasshopper, the Galapagos plugin is automatically included. This plugin offers two implementations of metaheuristics, namely a GA and the SA algorithm. On the other hand, the Goat plugin offers several types of algorithms, namely two direct-search methods (DIRECT and Subplex), a metaheuristic (CRS2), and two trust-region model-based methods (COBYLA and BOBYQA). Goat is able to offer these algorithms by using NLopt[5], a popular open-source library for nonlinear optimization. Silvereye is an implementation of the PSO algorithm, a metaheuristic. Finally, Opossum connects the RBFOpt library [Costa and Nannicini, 2018] to Grasshopper, enabling the usage of RBFs for global model-based optimization.

In recent years, optimization has been applied more frequently in architecture. One example used sampling strategies for real-world architectural problems, namely the optimization of truss structures and also lighting optimization [Caetano et al., 2018]. This example used the Rosetta AD tool and the prototype optimization module discussed in the next section. Wortmann has conducted several studies on black-box optimization methods for architectural design, including structural, daylighting, and building energy optimization [Wortmann et al., 2015, Wortmann and Nannicini, 2016, Wortmann et al., 2017, Wortmann and Nannicini, 2017], using the Grasshopper AD tool and its plugins. Other studies have also been made for structural optimization: [Hare et al., 2013] studies derivative-free algorithms and their usage in structural optimization, and [Zavala et al., 2014] studies the effectiveness of Multi-Objective metaheuristics.

In these studies, the authors denote a trend of using GAs as a primary choice [Hare et al., 2013, Rios and Sahinidis, 2013, Wortmann et al., 2017]. One of the possible reasons for this choice is that GAs can be easily applied to most problems. However there are still several other existing methods and it is usually preferable to use methods with proven convergence properties, which is not the case of metaheuristics [Conn et al., 2009]. Also, when tested against other algorithms, e.g., in building energy optimization problems, GAs tend to perform poorly [Hare et al., 2013, Wortmann et al., 2017, Wortmann and Nannicini, 2017]. Therefore, it is interesting to assess whether GAs can realistically be the first and only choice, or if better options exist.

[4] https://www.food4rhino.com/app/opossum-optimization-solver-surrogate-models, accessed 10 February 2019.
[5] http://ab-initio.mit.edu/nlopt, accessed 5 March 2018.

3

Optimization Framework for Architecture

Contents

3.1 Framework Description

As discussed in chapter 1, there are three main stages in an architectural optimization process: design, analysis, and optimization. The current process in popular tools, such as Grasshopper, involves using several plugins to connect to analysis tools and to perform optimization. Optimization plugins such as those discussed in section 2.2 allow Grasshopper users to experiment with optimization algorithms, however there is no plugin containing a large set of algorithms from the different categories. Therefore, if users wish to try several algorithms, they are forced to use several different plugins. Ideally, the users should be mostly concerned with creating the design, and not with issues arising from the desire to use optimization. However, there is currently no framework which is able to solve these issues, allowing simple use of several algorithms.

The framework proposed in this dissertation attempts to fill some of the gaps mentioned previously, by connecting an optimizer module with an AD tool. In this workflow, as described in figure 3.1, the user creates the design in the AD tool, and the tool is responsible for communicating with the other modules, similarly to the ADA approach, where the tool communicates with the analysis tools. In this case, the optimizer module suggests one set of values for the parameters in each evaluation, sends them to the AD tool, the tool generates an analytical model, sends it to the analysis tool, receives the performance value, and sends it back to the optimizer. Then, based on the performance value and the algorithm being used, the optimizer suggests a new set of values for the parameters, restarting the loop. Since this process is automated, the user only needs to select a few options, such as the metric to evaluate, the parameters to vary, the algorithms to use, and the constraints.

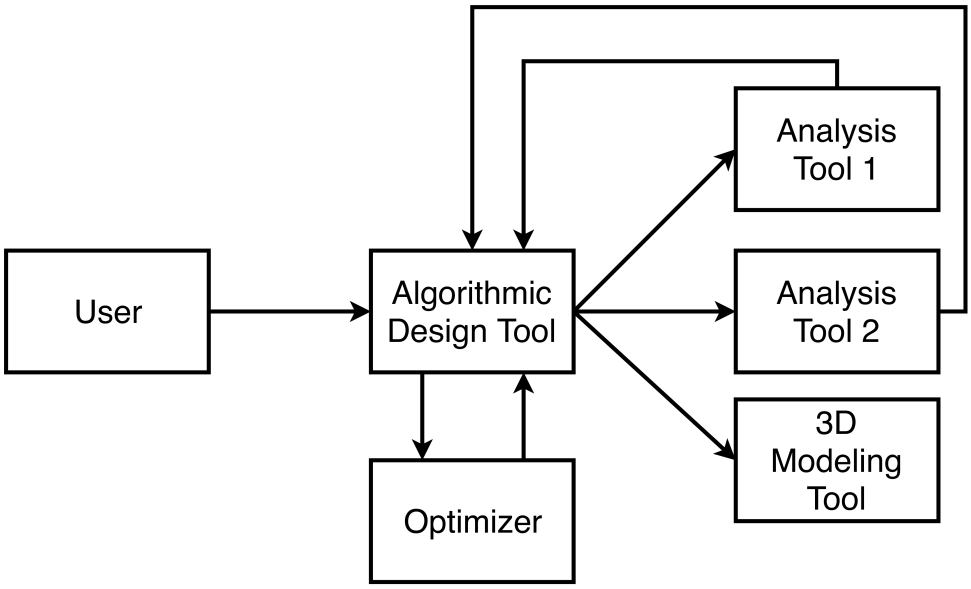

Figure 3.1: The ADA workflow with an optimizer module. In this workflow, the user only needs to interact with the AD tool and to select the options for the optimizer, enabling the automatic optimization of the design.

In order for the optimizer module to offer a large set of different algorithms, and to facilitate integration with the AD tool, the algorithms should be implemented with a uniform interface. The module should also be easily extendable to add more algorithms without any major changes to the interface, so that several algorithms can be easily made available to the user.

Another important aspect, which optimization tools typically do not consider, is the fact that the best solution for a specific performance metric may not be the preferred solution by the architects. Besides performance metrics, architects may have subjective opinions which go against the performance metric. Also, solutions with similar performance may be significantly different, but optimization algorithms typically only return the best solution, so the architect has no knowledge of the other solutions. To approach this issue, one of the proposed features of the framework is the visualization of the explored design variations and their performance values. As part of the optimization procedure, a scatter plot of the performance of each evaluation is produced. Then, since the framework is capable of generating analytical and 3D models, it is possible for the architect to select a specific evaluation to visualize the corresponding models in the chosen design tools. This gives architects the ability to quickly visualize the best designs according to the performance metric and select the design they prefer using their subjective assessment. Figure 3.2 shows an example of this feature, where an architect selected the seventh evaluation from the performance scatter plot, leading to the automatic generation of the corresponding 3D model in a CAD tool. This image was taken from a prototype of the proposed framework, which will be discussed in the next section.

3.2 Prototype Description

The proposed framework requires three types of tools: an AD tool, an analysis tool, and an optimizer. In this case, since the analysis tool depends on the metric to optimize, to create the prototype it is only necessary to choose an AD tool and implement the optimizer.

The chosen AD tool was Rosetta, which was discussed in sections 1.1 and 1.2. This tool was selected for several reasons. Since the tool itself does not contain any type of optimization plugins or modules, the creation of an optimizer which interacts with it would give Rosetta's users the ability to perform optimization. Also, since the tool follows the ADA approach, the optimizer could be used for many different types of optimization, e.g., lighting, acoustic, or structural optimization. Another important factor is that the tool follows the textual programming paradigm, which simplifies the communication between it and other modules. Specifically, instead of using a visual interface, the optimizer can be directly invoked using a programming language, and its interface is actually an API, i.e., a set of functions or procedures.

Regarding the optimizer, the first choice is the programming language. The chosen language was

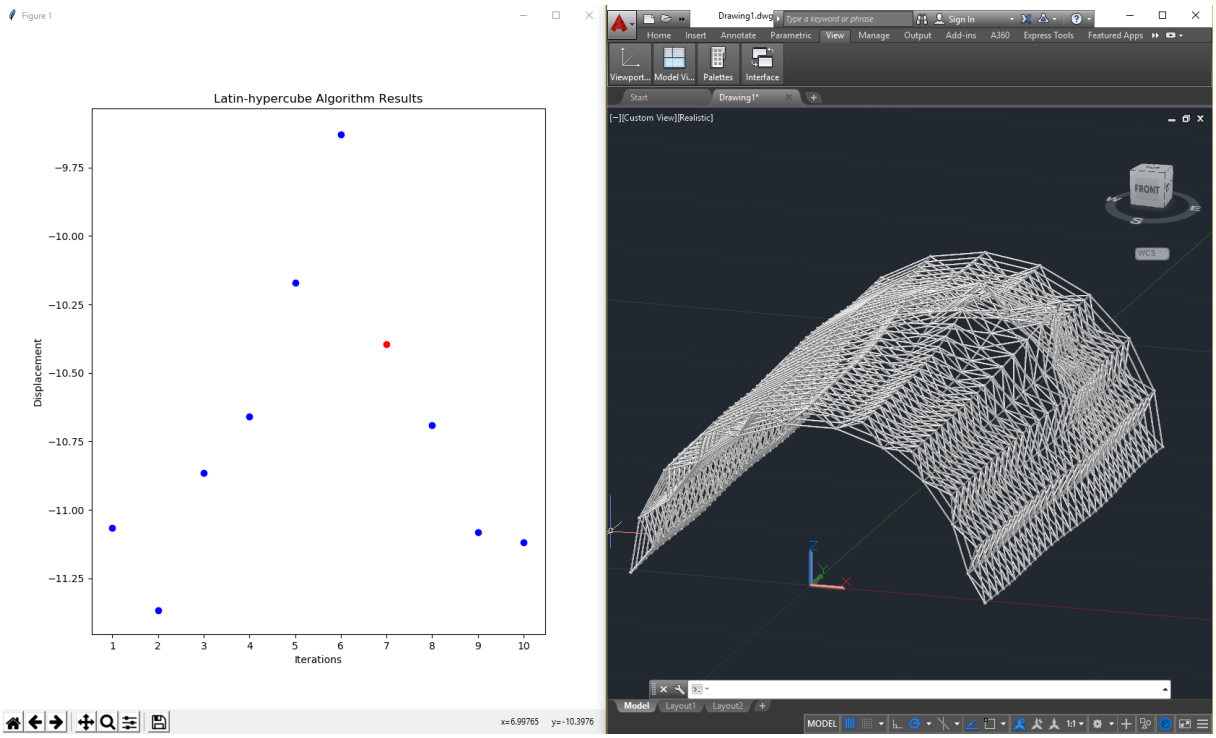

Figure 3.2: An example of the interactive plot feature, where an architect selected the seventh evaluation, the red dot on the left plot, and the values of the parameters in that evaluation were used to generate a 3D model using a CAD tool on the right.

Python, mainly because it already contains a very large and diverse set of optimization libraries. Since these libraries are already implemented, tested, and independently validated, it is best to use them instead of attempting to implement the algorithms again. This language was also chosen because it has a very simple syntax and is one of the languages for which Rosetta is available, i.e., its users can use Python to create their designs.

The second choice is concerned with which of the existing optimization libraries should be used. Table 3.1 shows for each of the 25 available algorithms the libraries used to implement them. In total, 5 libraries were used. The NLopt library[1] was chosen because it has a very diverse set of algorithms from the different categories, and because it has been used in Grasshopper plugins and other studies in architectural optimization, as discussed in section 2.2. The DEAP library [Fortin et al., 2012] was selected because it implements important Metaheuristics and offers several options for each of them. The pySOT library [2] was chosen because it implements several Model-based algorithms specifically for the optimization of costly functions. For the implementation of the sampling algorithms and the simple surrogate algorithms, the NumPy and Scikit-learn scientific computation libraries [Oliphant, 2006, Pedregosa et al., 2012] were used.

[1] http://ab-initio.mit.edu/nlopt, accessed 5 March 2018.
[2] https://pysot.readthedocs.io/en/latest/index.html, accessed 29 May 2018.

Table 3.1: A table specifying for each implemented algorithm in the optimizer, which library was used to implement it.

Class	Global	Algorithm	Library
Sampling	✓	Grid	NumPy
	✓	Latin Hypercube	NumPy
	✓	Random	NumPy
Direct-search	✓	DIRECT	NLopt
	✓	DIRECT-L	NLopt
	✗	Nelder-Mead	NLopt
	✗	PRAXIS	NLopt
	✗	Subplex	NLopt
Metaheuristic	✓	CMA-ES	DEAP
	✓	CRS2	NLopt
	✓	ESCH	NLopt
	✓	GA	DEAP
	✓	ISRES	NLopt
	✓	PSO	DEAP
Model-based	✗	BOBYQA	NLopt
	✗	COBYLA	NLopt
	✓	Forest	Scikit-learn
	✓	GP	pySOT
	✓	Linear	Scikit-learn
	✓	MLP	Scikit-learn
	✓	Polynomial	Scikit-learn
	✓	RBF-Cubic	pySOT
	✓	RBF-Linear	pySOT
	✓	SVMR	Scikit-learn
	✓	Tree	Scikit-learn

The final step of creating the optimizer is the definition of its API. As described in section 1.3, the definition of an optimization process includes the variables or parameters, the function or metric, and, if required, the constraints. Ideally, defining these elements should be an easy process. Also, it should be difficult or impossible to make simple mistakes, such as spelling mistakes, because this could have very high costs, e.g., if an optimization process that should run several algorithms overnight stops because the name of one algorithm is incorrectly spelled. Besides these definitions, it should also be possible to define a set of options, to control the details of the process. Finally, the entire API needs to be simple enough for the user to optimize his designs, and also for developers to include more algorithms from other libraries. The difficulty of creating such an optimizer is that the API of each optimization library is different, and a single unified API is required. To achieve this, the optimizer has a single API and, depending on which algorithm to run, it correctly converts the information obtained from its API to the API of the library to invoke. Therefore, the process of using the optimizer remains simple, and the

process of extending it with more algorithms from other libraries requires only the implementation of a connector between the different APIs.

The optimizer currently includes all of the previously mentioned features. Listing 3.1 shows its API with one simple usage example. In this example, a two parameter function is minimized in 100 evaluations with the DIRECT, GA, and RBF-Cubic algorithms, using very few lines of code. In the first lines, the user imports the optimize function, and all functions and constants used to define parameters, functions, options, constraints, and algorithms, i.e., the functions of the optimizer's API. Usually, the user already has a definition of the function to optimize (fitness function), in this case shown in lines 5 and 6. In the case of architectural optimization, this function may be calling another process, i.e., the Rosetta tool, but in this simple case it is just a function of two variables. Then, the steps to use the optimizer are the following:

1. Define the parameters of the function (lines 9-11). To do this, the user must call the `define_param` function of the API. This function is used to specify the name of each parameter, its type (discrete or continuous), and its lower and upper bounds.

2. Define the fitness function (line 14). Although a definition of the function already exists (lines 5 and 6), it is still required to invoke the `define_function` procedure, to verify the arguments.

3. Define which algorithms to use (line 17). In this case, instead of using strings to specify the algorithms, each algorithm is defined as a constant in the imported prototype definitions (line 2), so the user needs to use a list of constants. This helps prevent possible errors, since the optimization cannot start if the constant name is incorrectly typed.

4. If required, define extra options (line 20). Extra options can be defined using the `define_options` function. Options can be used for defining additional configurations, such as setting termination conditions, setting the name of results files, activating the interactive plot, or setting the population size for population-based algorithms. All options have default values, so users only need to specify the ones they wish to change. Also, options are specified through keywords, instead of strings, to avoid misspelling errors.

5. The final step is to start the optimization (line 22). To do this, the user simply needs to call the `optimize` function, and give the function, parameters, algorithms, and options that were previously defined. This function maps each algorithm to the specific library implementation and invokes it. Each algorithm is sequentially executed, and the result of each evaluation, as well as the final result and termination cause, are returned to the user.

Listing 3.1: A simple example of the optimizer's API

```
1  from prototype.optimization import optimize
2  from prototype.definitions import *
3
4
5  def fitness_function(x, y):
6      return (1 - x)**2 + 100 * (y - x**2)**2
7
8  # Define parameters
9  x = define_param(name="x", type_=float, lower_bound=-40, upper_bound=40)
10 y = define_param(name="y", type_=float, lower_bound=-40, upper_bound=40)
11 parameters = [x, y]
12
13 # Define fitness function
14 f = define_function(parameters, fitness_function, maximize=False)
15
16 # Define the algorithms to use
17 algorithms = [DIRECT, GA, RBF_Cubic]
18
19 # Define options
20 options = define_options(eval_limit=100)
21
22 optimize(f, parameters, algorithms, options)
```

With these simple steps, it is possible to optimize a function with several algorithms. However, more complex optimization processes might require additional details, which have not yet been discussed. Listing 3.2 shows a more complex example, which includes an inequality constraint. Similarly to the fitness function, the constraint is initially defined as a normal Python function (line 11) and, afterwards, it is specified using the `define_constraint` function (line 23). In this example, the inequality constraint is $x + y - 2 \leq 0$, therefore the value returned by the constraint function should be less than or equal to zero if the parameters satisfy the constraint. However, in the case of an equality constraint, such as $x + y - 2 = 0$, then an acceptable value of the parameters would cause the function to return 0.

Listing 3.2: A more complex usage example, which includes constraints

```
1  from prototype.optimization import optimize
2  from prototype.definitions import *
3
```

```
4
5  # Define the fitness function
6  def fitness_function(x, y):
7      return (1 - x)**2 + 100 * (y - x**2)**2
8
9
10 def constraint(x, y):
11     return x + y - 2
12
13 # Define Parameters
14 x = define_param(name="x", type_=float, lower_bound=-40, upper_bound=40)
15 y = define_param(name="y", type_=float, lower_bound=-40, upper_bound=40)
16 parameters = [x, y]
17
18 # Define fitness function
19 f = define_function(parameters, fitness_function, maximize=False)
20
21 # Define constraints
22 c = define_constraint(parameters, constraint, inequality=True)
23 constraints =[c]
24
25 # Define the algorithms to use
26 algorithms = [DIRECT, GA, RBF_Cubic]
27
28 # Define options
29 options = define_options(eval_limit=100)
30
31 optimize(f, parameters, algorithms, options, constraints)
```

The previous examples are sufficient to show the prototype's ability to perform complex optimization processes with a few simple steps. In summary, the main components of the public API are the `define_param`, `define_function`, `define_constraint`, `define_options`, and `optimize` functions.

The `define_param` function takes as arguments the name of the parameter, the type of the parameter (int or float), and the lower and upper bounds of the parameter. The function then returns a `Parameter` object, which encodes all of the information about the parameter for future use.

The `define_function` function receives a list of `Parameter` objects, a standard Python function which returns the performance value, and a boolean value indicating if the function should be maximized or minimized. This function then returns a `Function` object encoding this information, which can later be

given to the `optimize` function.

The `define_constraint` function takes a list of `Parameter` objects, a Python function which specifies the constraint, and a boolean value indicating if it is an inequality or equality constraint. This function returns a `Constraint` object with this information.

The `define_options` function can be used to define several options for the optimization process. If no arguments are given, then default options are generated. This function returns an `Options` object with all options. Some of the available options are:

1. `crossover_prob`: The probability of the crossover step in a GA
2. `eval_limit`: The maximum number of evaluations for each algorithm
3. `population_size`: The population size used for the CMA-ES, GA, and PSO algorithms
4. `surrogate_opt_alg`: The optimization algorithm used to optimize the surrogate in the simple surrogate algorithm
5. `time_limit`: The maximum time limit for each algorithm

The `optimize` function takes a `Function` object, a list of `Parameter` objects, a list of `Algorithms`, an `Options` object, and a list of `Constraint` objects. This is the main function which executes each algorithm and returns for each one a `Result` object indicating the best solution, termination condition, and other relevant information. The algorithms themselves are constants defined in the definitions module.

4

Evaluation

Contents

This chapter focuses on the evaluation of both the proposed framework and the algorithms that have been implemented in the current prototype. In section 4.1, three known test functions are used to assess the effectiveness of the algorithms. Since these are known test functions, it is possible to analyze if the algorithms converged to an optimal solution. In section 4.2, two structural optimization case studies are used to evaluate both the framework and a subset of algorithms. These case studies use the Rosetta tool and the optimizer, following the framework proposed in chapter 3, including the design, analysis, and optimization stages.

Overall, the goal of this chapter is to answer the following questions:

- Is there a single algorithm that can consistently, across all case studies, reach the best solution?
- Can algorithms of a specific category or with a specific property, achieve better results than all others?
- How does the best result of each algorithm vary across several test runs?
- How do the results found by the algorithms evolve as the number of function evaluations increases?
- How much time does each algorithm take to reach its best solution?

By answering these questions, it will be possible to understand if, as discussed in sections 1.4 and 2.2, the current popular method of selecting only a GA for architectural optimization is or is not the best choice, and if there exists an algorithm or category of algorithms which excels when compared to others.

4.1 Test Functions

To evaluate the effectiveness of the algorithms, three known test functions are used. In order to analyze the capabilities of each algorithm in different situations, the properties of each function, i.e., the number of variables, the shape, and the number of global and local minima, are different.

In all three functions, all algorithms, except the sampling algorithms, are tested with an evaluation limit of 1000 and a stopping value of 1E-10. The stopping value is used to avoid possible numerical stability problems due to floating point precision and because all functions have at least one global optimum at 0.

For population-based algorithms, after experimenting different values for the initial population, the formula that produced the best results is $20(n + 1)$, where n is the number of parameters. Regarding the time taken by each algorithm, it is important to notice that it depends on the processing power of the machine where tests are made. In order to remove that dependency, the number of evaluations of each algorithm is measured instead, as this is proportional to the actual time spent but is independent

of the processing power and of the complexity of the evaluation function. Additionally, as discussed in section 2.1, stochastic algorithms and algorithms which depend on starting points may achieve different values if tested twice under the same conditions. To mitigate this issue, and to provide a statistically fair analysis, each algorithm is tested 10 times, and both the standard deviation and the mean of the results are used to draw conclusions.

Finally, when comparing the different algorithms, besides the categories and properties described in section 2.1, the algorithms may also be classified into the following groups:

- `G_Direct-search`: Global direct-search algorithms, i.e., DIRECT and DIRECT-L
- `L_Direct-search`: Local direct-search algorithms, i.e., Nelder-mead, PRAXIS, and Subplex
- `G_Model-based`: Global Model-based algorithms, except the simple surrogate algorithms, i.e., GP, RBF-Cubic, and RBF-Linear
- `L_Model-based`: Local model-based algorithms, i.e., BOBYQA and COBYLA
- `S_Model-based`: The simple surrogate algorithms, i.e., Forest, Linear, MLP, Polynomial, SVMR, and Tree

These settings and properties apply to all of the three test functions.

4.1.1 Himmelblau Function

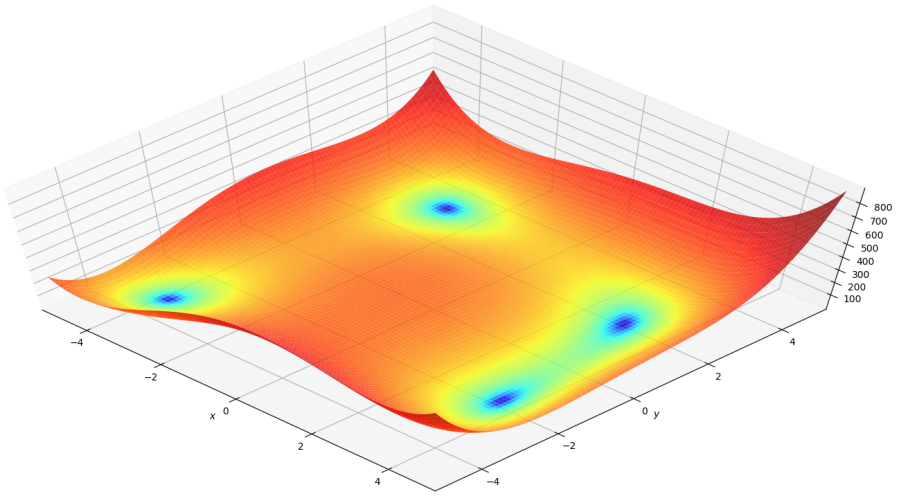

Figure 4.1: A plot of the Himmelblau test function, showing its four global minima.

The Himmelblau function is an optimization test function with two variables, a simple shape, and four different optimal solutions, which are also the only existing local minima, as displayed in figure 4.1 and described by the following equations [Himmelblau, 1972]:

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

$$f(3,2) = f(-2.805118, 3.131312) = f(-3.779310, -3.283186) = f(3.584428, -1.848126) = 0$$

$$x \in [-5,5]$$

$$y \in [-5,5]$$

As described in the beginning of section 4.1, each algorithm was executed 10 times with a 1000 evaluations limit. By observing the results in table 4.1, it is immediately clear that, despite the simplicity of the function and the large limit of evaluations, only direct-search algorithms, the CMA-ES algorithm, and the two local model-based algorithms were able to consistently reach a global minimum.

Table 4.1: A table for the Himmelblau test function, showing the average result and standard deviation of the result of each algorithm. It also shows the average evaluation where the result was found.

Class	Global	Algorithm	Avg Evaluation	Avg Result	Std Dev Result
Direct-search	✔	DIRECT	607.0	0.00E+00	0.00E+00
Direct-search	✔	DIRECT-L	400.0	0.00E+00	0.00E+00
Direct-search	✘	Nelder-Mead	86.5	0.00E+00	0.00E+00
Direct-search	✘	PRAXIS	52.0	0.00E+00	0.00E+00
Direct-search	✘	Subplex	132.8	0.00E+00	0.00E+00
Metaheuristic	✔	CMA-ES	347.2	0.00E+00	0.00E+00
Metaheuristic	✔	CRS2	935.1	1.46E-05	2.13E-05
Metaheuristic	✔	ESCH	544.9	3.18E-01	3.00E-01
Metaheuristic	✔	GA	946.0	2.75E-03	4.19E-03
Metaheuristic	✔	ISRES	385.0	1.36E-01	1.63E-01
Metaheuristic	✔	PSO	472.5	5.50E-01	5.13E-01
Model-based	✘	BOBYQA	44.3	0.00E+00	0.00E+00
Model-based	✘	COBYLA	188.2	0.00E+00	0.00E+00
Model-based	✔	Forest	819.8	2.02E+00	1.62E+00
Model-based	✔	GP	531.9	9.74E-02	7.12E-02
Model-based	✔	Linear	874.6	5.55E+00	2.11E+00
Model-based	✔	MLP	900.0	3.81E+00	2.68E+00
Model-based	✔	Polynomial	892.4	2.96E+00	2.55E+00
Model-based	✔	RBF-Cubic	463.7	9.69E-02	7.99E-02
Model-based	✔	RBF-Linear	315.2	2.30E-01	2.61E-01
Model-based	✔	SVMR	924.3	1.61E+00	1.73E+00
Model-based	✔	Tree	888.6	1.52E+00	1.34E+00

Overall, the best performing algorithms were the local algorithms, with all of them being able to reach a global minimum before the 200th evaluation, and with the BOBYQA algorithm on average reaching its best solution in 44.3 evaluations. In this function, since all local minima are also global minima, it was expected that local algorithms would excel.

Regarding metaheuristics, besides CMA-ES, no other algorithm was able to reach a global minimum. Both the GA and CRS2 algorithms were able to, on average, obtain values that were close to a global minimum, while the PSO, ISRES, and ESCH algorithms were the worst-performing metaheuristics. Besides not reaching a global minimum, these three algorithms also found their best result around the 500th evaluation, which means that for the last 500 iterations these algorithms were unable to improve, i.e., they stagnated. However, if this was a costly function, this type of behavior would be good, since the algorithms found a reasonably good result faster than others. Also, despite the stochastic behavior of metaheuristics, the standard deviation of their best result over the ten runs was negligible.

Of the model-based algorithms, the simple surrogate algorithms (Forest, Linear, MLP, Polynomial, SVMR, and Tree) were the ones which on average obtained the worst results. However, all of them have a large standard deviation, since on some runs they performed much better than on others. On the contrary, the other global surrogate algorithms (GP, RBF-Cubic, and RBF-Linear), were close to a global minimum before their 500th evaluation, but were unable to obtain it, while the local model-based algorithms (BOBYQA and COBYLA) both achieved a global minimum.

Figure 4.2 shows the average result of each group of algorithms across the 1000 evaluations. The results were normalized into a 0-1 scale, and logarithmic scaling was applied to the plot's y-axis, to help understand how the results changed while the evaluations increased. As shown by the plot, the local direct-search methods were the best group. The global direct-search and local model-based methods also achieved good results, while the global model-based, metaheuristics, and simple surrogate algorithms all showed little improvement after the 500th evaluation. Therefore, it appears that the best overall category for this function is the direct-search methods.

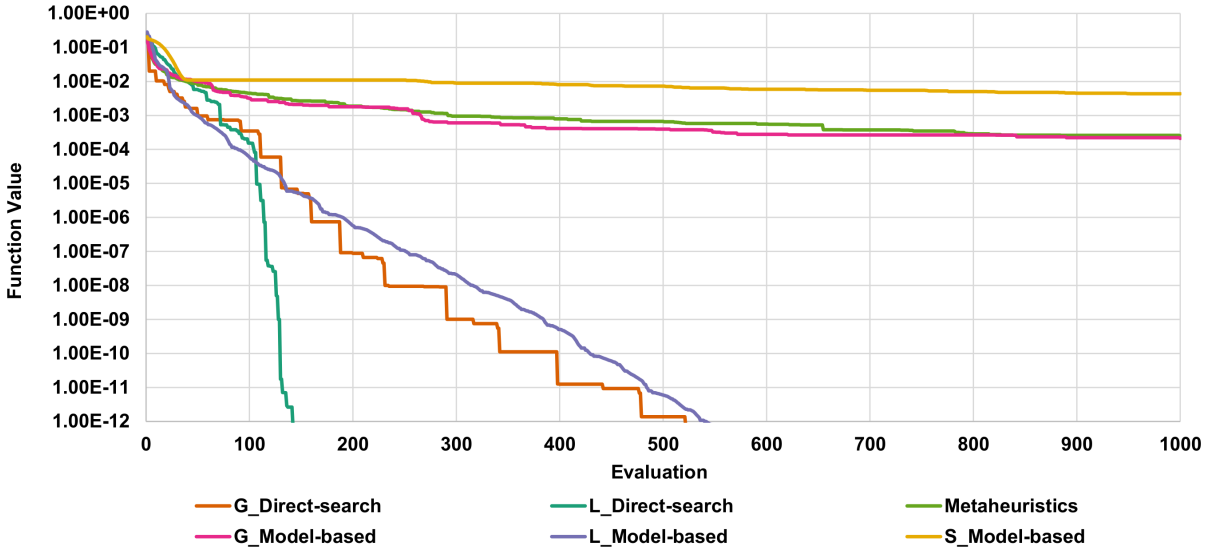

Figure 4.2: A plot of the average best result of each group of algorithms on the Himmelblau test function. The results were scaled into a 0-1 scale, and logarithmic scaling was used on the y-axis.

4.1.2 Levy Number 13 Function

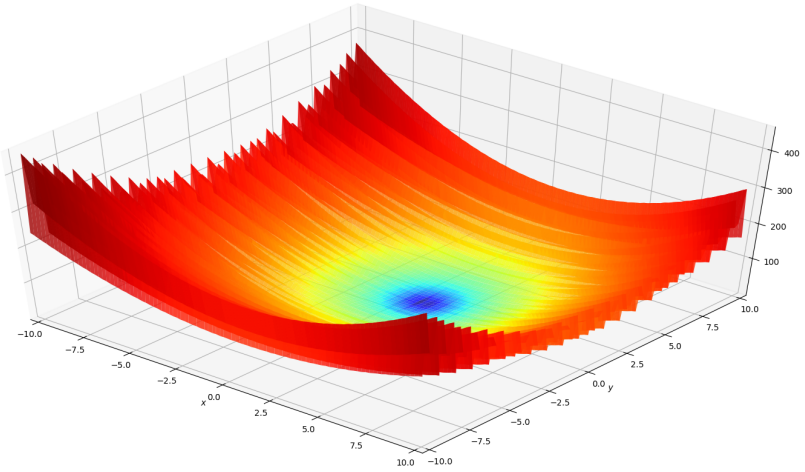

Figure 4.3: A plot of the Levy number 13 test function, which has only one global minimum.

The Levy number 13 function is an optimization test function with two variables, a single global minimum, several local minima, and an irregular shape filled with peaks, as displayed in figure 4.3 and described by the following equations:

$$f(x,y) = sin(3\pi x)^2 + (x-1)^2 * (1 + sin(3\pi y)^2) + (y-1)^2 * (1 + sin(2\pi y)^2),$$
$$f(1,1) = 0$$
$$x \in [-10, 10]$$
$$y \in [-10, 10]$$

Unlike the Himmelblau function of the previous section, the large amount of peaks and local minima make the Levy function a much harder function to optimize. The results in Table 4.2 show this difficulty, since only two algorithms were able to reach the global minimum with a 1000 evaluation limit.

Only DIRECT and DIRECT-L achieve the global optimum. An explanation for their great results is related to how both algorithms explore the function. As discussed in section 2.1, these algorithms divide the search space equally, and begin by sampling the center point. In this case, the center point (0, 0) is close to the actual global optimum (1, 1) and is a good result compared to points further from the optimum. Due to the nature of these algorithms, they will tend to search around this point for the global optimum, instead of exploring further away. Therefore, these great results occur due to the algorithm's sampling policy and the fact that the first point gives them a good head start.

Table 4.2: A table for the Levy number 13 test function, showing the average result and standard deviation of the result of each algorithm. It also shows the average evaluation where the result was found.

Class	Global	Algorithm	Avg Evaluation	Avg Result	Std Dev Result
Direct-search	✓	DIRECT	465.0	0.00E+00	0.00E+00
Direct-search	✓	DIRECT-L	335.0	0.00E+00	0.00E+00
Direct-search	✗	Nelder-Mead	104.0	3.06E+01	4.35E+01
Direct-search	✗	PRAXIS	55.1	2.98E+01	3.39E+01
Direct-search	✗	Subplex	170.8	2.45E+01	3.83E+01
Metaheuristic	✓	CMA-ES	410.9	2.53E+00	7.48E+00
Metaheuristic	✓	CRS2	916.5	1.11E-05	8.07E-06
Metaheuristic	✓	ESCH	781.6	3.45E-02	3.70E-02
Metaheuristic	✓	GA	673.4	3.63E-02	4.02E-02
Metaheuristic	✓	ISRES	529.3	4.53E-01	2.54E-01
Metaheuristic	✓	PSO	581.8	1.07E-01	1.01E-01
Model-based	✗	BOBYQA	37.0	2.81E+01	3.65E+01
Model-based	✗	COBYLA	466.9	3.34E+01	3.99E+01
Model-based	✓	Forest	619.7	6.52E-03	5.27E-03
Model-based	✓	GP	554.9	4.33E-01	2.60E-01
Model-based	✓	Linear	313.8	7.92E-03	8.73E-03
Model-based	✓	MLP	638.9	1.15E-02	1.02E-02
Model-based	✓	Polynomial	264.1	8.81E-03	1.09E-02
Model-based	✓	RBF-Cubic	493.3	1.79E-01	1.14E-01
Model-based	✓	RBF-Linear	559.6	4.97E-01	2.14E-01
Model-based	✓	SVMR	495.4	2.12E-02	2.73E-02
Model-based	✓	Tree	201.0	1.84E-02	1.71E-02

Regarding the local algorithms, as expected, they converge to local minima quickly. However, unlike the Himmelblau function, local minima in this function have much higher values than the global minimum, therefore they are not a good solution. Besides converging to local minima, these algorithms also display a large standard deviation of the results between runs. The high standard deviation comes from the fact that the initial point given to all these algorithms is randomized on each run, and the algorithms tend to always find the closest minimum to that initial point. These issues are the main problems of local algorithms, as discussed in section 2.1, and they were expected to occur in difficult functions such as this one.

The metaheuristics also under-performed. Even the CMA-ES algorithm, one of the best for the Himmelblau function, achieved bad results and was even the worst metaheuristic both on average and on standard deviation. Of the other metaheuristics, CRS2 stands out, since it was the best algorithm besides DIRECT and DIRECT-L. The remaining ones obtain good results, taking into account that this function has a highly irregular shape.

Finally, regarding the global model-based algorithms and the simple surrogates, they all present good results. The Forest, Linear, and Polynomial algorithms, three simple surrogates, were the best-performing model-based algorithms. By observing the plot of each group of algorithms in figure 4.4, it is possible to observe that the simple surrogate algorithms standout as the best algorithms, after

the global direct-search algorithms. The other global model-based algorithms achieved slightly worse results, similarly to the metaheuristics.

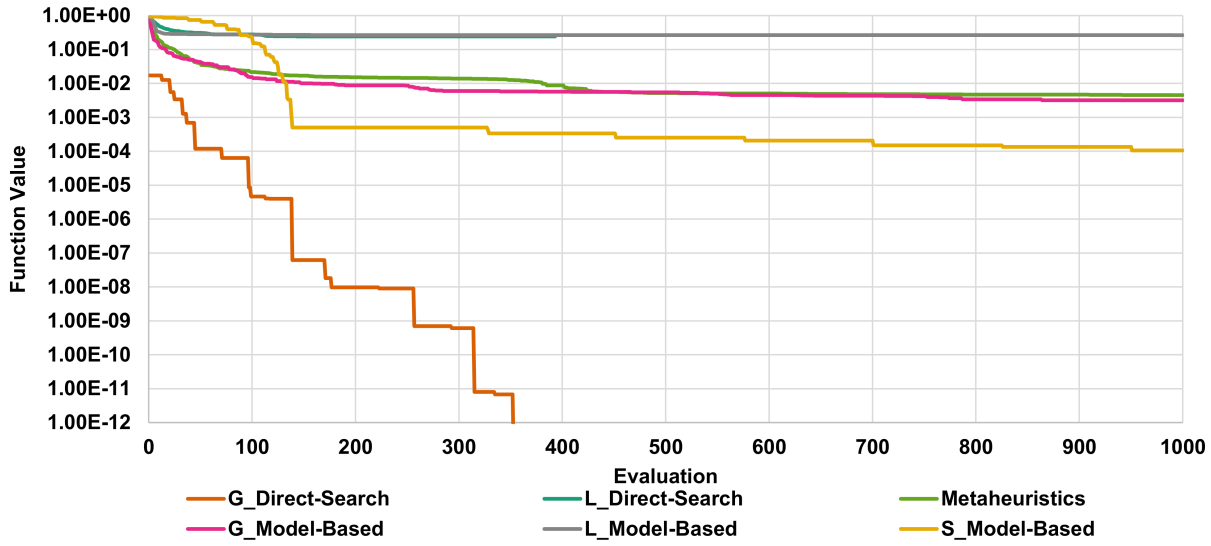

Figure 4.4: A plot of the average best result of each group of algorithms on the Levy number 13 test function. The results were scaled into a 0-1 scale, and logarithmic scaling was used on the y-axis.

4.1.3 Rastringin Function

The Rastringin function, similarly to the Levy number 13, is a complex function with several local minima. Unlike the previously presented functions, it can be generalized to arbitrary dimensions. Figure 4.5 displays the function with two variables, while the following equations describe the function for n dimensions:

$$f(x_1, ..., x_n) = 10n + \sum_{i=1}^{n} x_i^2 - 10cos(2\pi x_i)$$
$$f(0, ..., 0) = 0$$
$$x_i \in [-5.12, 5.12]$$

For the purposes of this study, the number of dimensions, n, was set to 10. Effectively, by increasing the number of dimensions while maintaining the evaluation limit, algorithms which attempt to explore the function by following a path, i.e., direct-search and local algorithms, should have more problems than before. These types of high dimensional problems are where metaheuristics tend to perform well, however the shape of the function and large amount of local minima still make it a difficult case.

Similarly to the Levy number 13 function, by observing the results over 10 runs in table 4.3, it is possible to observe that DIRECT and DIRECT-L were the best algorithms, achieving the global optimum

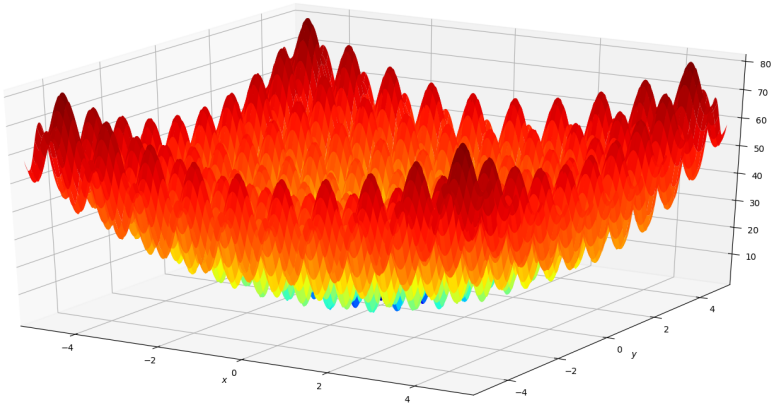

Figure 4.5: A plot of the 2-variable Rastringin test function, with its global minimum in the center.

on the first result. This occurs because the algorithms always sample the center point of the solution space first, which in this case is the optimal solution.

Regarding the simple surrogate algorithms, they are the only ones to exhibit good results with a small standard deviation. However, all of them return very similar results obtained on the 126th evaluation. The fact that all of them obtain the best result on the same evaluation suggests that the different surrogates are yielding similar values, and that the common parts of the algorithms, such as the sampling strategy, are responsible for this.

As shown in figure 4.6, metaheuristics, global model-based, and local algorithms, all exhibit similar results. Overall, it appears that none of these categories can beat the simple surrogates on this complex function, even though metaheuristics are known for their performance on high dimensional optimization problems. The global direct-search category is not represented, since it immediately found the optimum solution.

Table 4.3: A table for the Rastringin test function, showing the average result and standard deviation of the result of each algorithm. It also shows the average evaluation where the result was found.

Class	Global	Algorithm	Avg Evaluation	Avg Result	Std Dev Result
Direct-search	✓	DIRECT	1.0	0.00E+00	0.00E+00
Direct-search	✓	DIRECT-L	1.0	0.00E+00	0.00E+00
Direct-search	✗	Nelder-Mead	976.0	8.97E+01	3.04E+01
Direct-search	✗	PRAXIS	255.2	9.25E+01	1.94E+01
Direct-search	✗	Subplex	699.9	6.30E+01	2.37E+01
Metaheuristic	✓	CMA-ES	933.9	2.81E+01	1.37E+01
Metaheuristic	✓	CRS2	789.5	4.76E+01	8.20E+00
Metaheuristic	✓	ESCH	858.4	3.11E+01	4.31E+00
Metaheuristic	✓	GA	737.4	3.35E+01	1.01E+01
Metaheuristic	✓	ISRES	342.4	6.96E+01	5.49E+00
Metaheuristic	✓	PSO	647.7	7.00E+01	8.14E+00
Model-based	✗	BOBYQA	457.4	8.77E+01	3.37E+01
Model-based	✗	COBYLA	883.2	7.45E+01	1.77E+01
Model-based	✓	Forest	126.0	2.59E-01	7.14E-02
Model-based	✓	GP	516.0	8.90E+01	7.89E+00
Model-based	✓	Linear	126.0	2.65E-01	9.48E-02
Model-based	✓	MLP	126.0	3.20E-01	6.23E-02
Model-based	✓	Polynomial	126.0	2.60E-01	6.28E-02
Model-based	✓	RBF-Cubic	452.8	6.98E+01	6.74E+00
Model-based	✓	RBF-Linear	535.6	8.24E+01	8.76E+00
Model-based	✓	SVMR	126.0	2.68E-01	6.71E-02
Model-based	✓	Tree	126.0	2.83E-01	7.47E-02

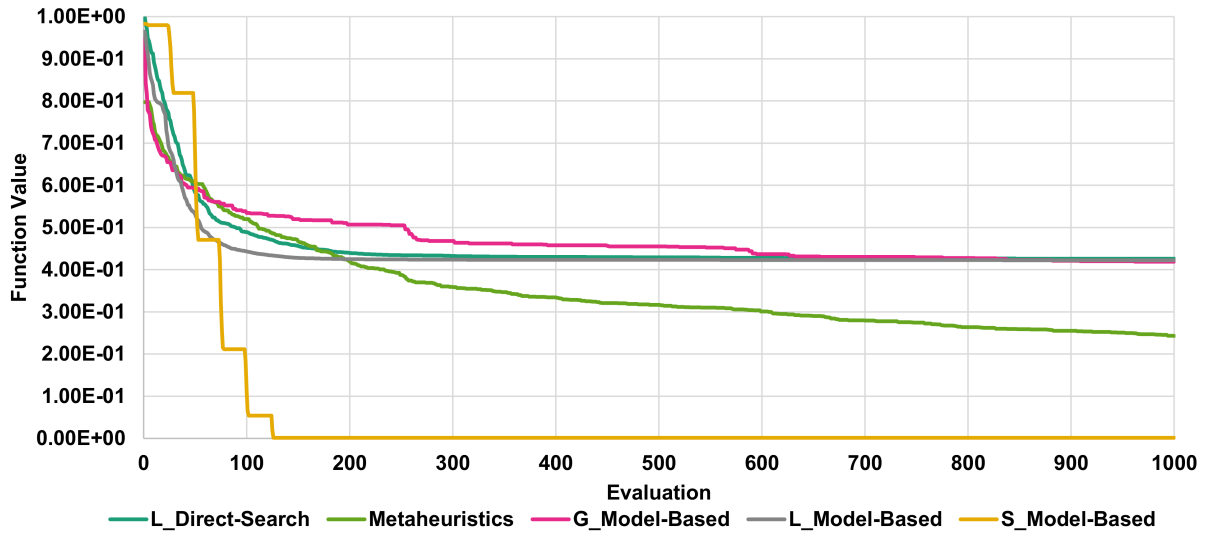

Figure 4.6: A plot of the average best result of each group of algorithms on the Rastringin test function. The results were scaled into a 0-1 scale, not including the global direct-search algorithms, and logarithmic scaling was used on the y-axis.

4.2 Case Studies

In the previous section, optimization test functions were used to test the effectiveness of different algorithms and the underlying prototype optimizer. The goal of this section still includes testing the algorithms, but it also focuses on the entire architectural optimization framework from start to finish. To do this, two architectural case studies are used with the optimization framework described in chapter 3.

4.2.1 An Urban Museum

This case study addresses a real two floor service building, located in an urban context, intended to work as a museum [Caetano et al., 2018]. The case study study itself was created in collaboration with an architectural studio. The goal of the architects was to create a truss structure to support the entire span of the building's roof slab. Initially, the problem was to find the height of the structure which minimized the maximum vertical displacement. The architect's previous knowledge, obtained from years of experience, was not enough to determine this value. In general, it is known that a very small height would give extremely high displacement, and that increasing the height would improve the result up to a certain point, after which the result would become worse. Also, taking into account aesthetics, the architects preferred the smallest height truss which complied with European regulation for maximum displacement - 2.56 centimeters in this case. Therefore, the goal was to find the smallest height which returned 2.56 centimeters.

In this stage of the design process the architects still had a wide range of options regarding the truss design itself. They considered two types of trusses - Warren with Verticals and Pratt - and considered variations of those types, by varying the number of modules between supports - 6, 8, and 10 modules. To perform the optimization, the actual height of the truss was the only parameter, and it was bounded between 0.8 and 2.1. Each different type of truss was optimized separately.

The different truss types were all defined using Rosetta, and the Robot analysis tool was used to calculate their performance. Regarding the optimization, since this problem only contained a single parameter, as an initial approach, Latin Hypercube sampling was used. This entire process utilized the framework described in chapter 3. Figure 4.7 shows one line fitted to each type of truss, using the samples returned by the optimization algorithm.

By observing the results of simply fitting a line to the samples, it became immediately apparent that no more algorithms were required. For the specific conditions of this case study, the fitted lines were simple and accurate. With these results, the architects were able to make informed decisions regarding the feasibility of certain types of trusses, and also to understand which type was best for this problem.

Although in this case study only one algorithm was used, it was possible to test the effectiveness of the architectural optimization framework. After defining the algorithm using Rosetta, the architect simply

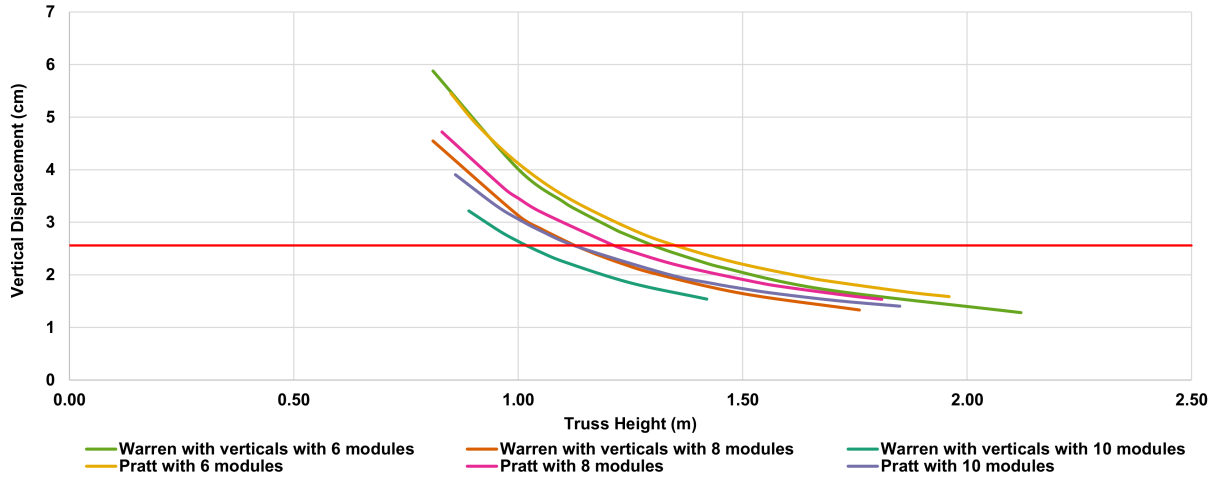

Figure 4.7: Results of using sampling methods with different types of trusses for the museum case study. The red line indicates the 2.56cm preferred value.

specified the analysis tool and the algorithm, using the API described in section 3.2, and was able to obtain previously unknown insights, which impacted the quality of the final design.

4.2.2 Space Frame Optimization

Besides the real-world case study, another architectural case study was created. This artificial case study was a space frame deformed by three attractor points intended to give the structure a non-uniform shape. In this case, the parameters are the position of the attractor points, and the goal is to minimize the maximum vertical displacement, which is provided by the Robot analysis tool. Figure 4.8 illustrates two examples of the structure.

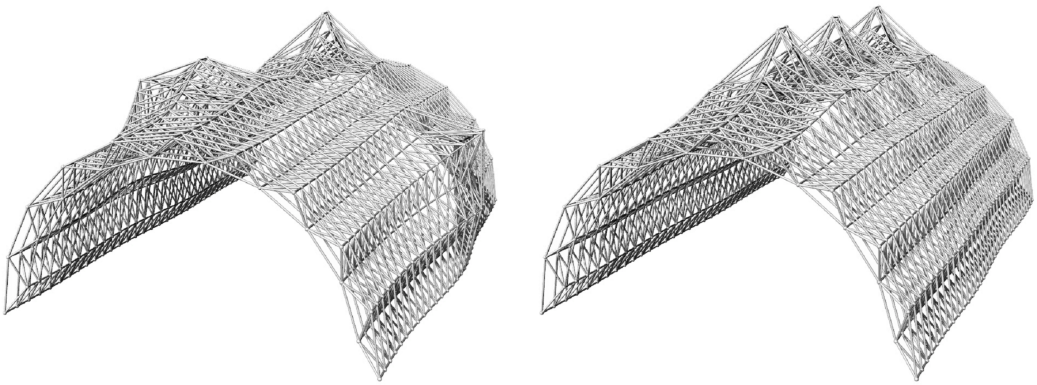

Figure 4.8: Two variations of a space frame with three attractor points, which give it a non-uniform shape.

At the time of performing these tests, the prototype optimizer contained all algorithms except the

CMA-ES, PSO, sampling, and simple surrogate algorithms. For testing the remaining algorithms, the process was similar to the one described in section 4.1, but, due to how time-consuming each evaluation was, instead of an evaluation limit of 1000 and 10 test runs, the limit was 100 evaluations and 3 test runs.

Figure 4.9 illustrates the mean best result, as a function of the number of evaluations, for the groups defined in section 4.1. According to the results, the best algorithms were the global direct-search and global model-based methods. In the early stages of the optimization process, the global model-based algorithms obtained the best results, but after the 35th evaluation, the global direct-search algorithms performed better. After the 30th evaluation, both categories obtained better values than all other categories.

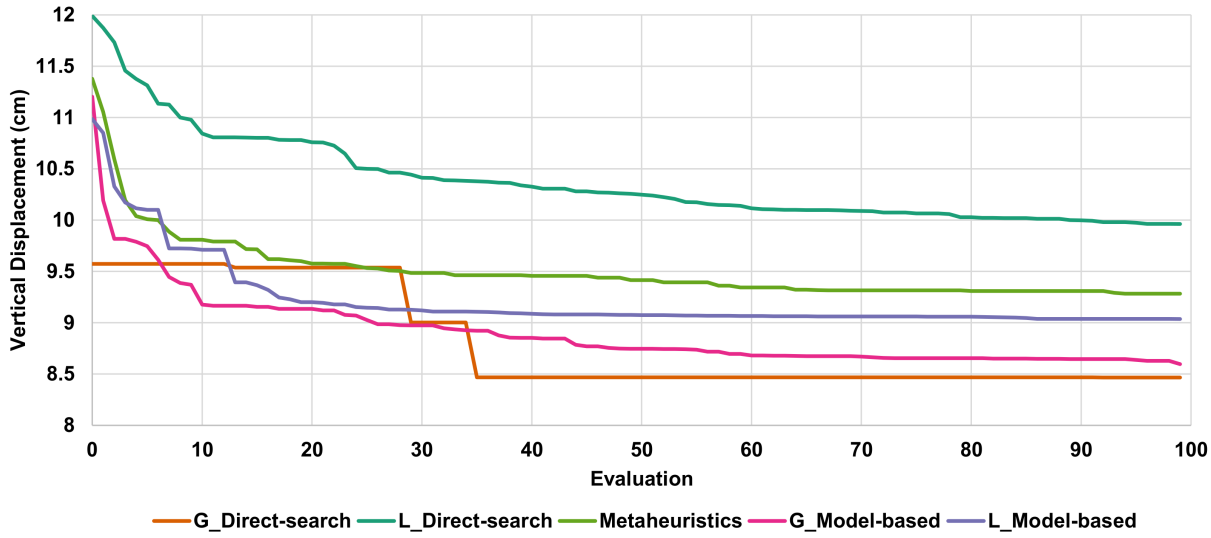

Figure 4.9: Results of applying several optimization algorithms to the space frame case study.

Regarding the result of each individual algorithm, table 4.4 shows the best result of each algorithm and the evaluation where the best result was obtained. The DIRECT and DIRECT-L algorithms almost achieved the best overall result and converged quickly. The other direct-search methods, Nelder-Mead, PRAXIS, and Subplex, performed much worse, and are within the four worst algorithms. For metaheuristics, the common GA was the second worst algorithm, even though it found its best result on the 78th evaluation. ISRES was the best metaheuristic, achieving the 6th best result, even though it found that result very early on. CRS2 and ESCH only achieved the 9th and 10th best result. Finally, the best performing algorithms were model-based. COBYLA and BOBYQA, the local methods, achieved the 7th and 8th best result. The global algorithms, RBF-Linear, RBF-Cubic, and GP, were some of the best algorithms, only matched by DIRECT and DIRECT-L. RBF-Linear achieved the best overall result.

In this case, it appears that, yet again, metaheuristics are not the best type of algorithms to solve the problem, as they are beaten by direct-search and model-based algorithms.

Table 4.4: A table showing the average result of each algorithm, over 3 runs with a 100 evaluation limit. It also shows the average evaluation where each algorithm found its best result.

Class	Global	Algorithm	Avg Evaluation	Avg Result	Std Dev Result
Direct-search	✓	DIRECT	30.00	8.47E+00	0.00E+00
Direct-search	✓	DIRECT-L	36.00	8.47E+00	0.00E+00
Direct-search	✗	Nelder-Mead	84.67	9.32E+00	8.52E-01
Direct-search	✗	PRAXIS	63.00	1.09E+01	2.38E+00
Direct-search	✗	Subplex	93.33	9.53E+00	5.00E-01
Metaheuristic	✓	CRS2	47.67	9.11E+00	1.27E-01
Metaheuristic	✓	ESCH	70.00	9.15E+00	9.98E-02
Metaheuristic	✓	GA	78.33	9.95E+00	1.90E-01
Metaheuristic	✓	ISRES	36.33	8.93E+00	2.17E-01
Model-based	✗	BOBYQA	50.67	8.99E+00	6.75E-02
Model-based	✗	COBYLA	68.00	9.08E+00	2.60E-01
Model-based	✓	GP	61.67	8.61E+00	2.13E-01
Model-based	✓	RBF-Cubic	49.67	8.75E+00	2.24E-01
Model-based	✓	RBF-Linear	68.00	8.43E+00	2.13E-01

5

Discussion

Contents

In this chapter, the conclusions obtained thus far, from the introduction (chapter 1), algorithms review (chapter 2), proposed framework (chapter 3), and tests (chapter 4), are discussed, and guidelines for future work are introduced.

5.1 Conclusions

The increase in processing power of computers has changed the architectural world. Nowadays, the typical architectural process involves designing with CAD or BIM tools, which enables the nearly automated production of several required documents and views, but does not give the architects a chance to try and visualize several different design variations, or to attempt to find the best performing one. The growing concerns of sustainability, paired with regulations and other restrictions, have driven the development of the PBD approach, where performance is a primary issue. With PBD, the entire design process is driven by performance, and changes are made to increase said performance. However, this leads to a greater workload for the architect, since complex design changes are time-consuming, and architectural projects frequently have strict deadlines which are hard to fulfill. To enable complex changes to be made effectively, new paradigms emerged, such as AD. With these new paradigms and tools, architects have the ability to visualize variations of their design with little effort. Besides design creation, some of these tools also allow taking advantage of other advances in engineering, namely analysis tools. By using either plugins or the ADA approach, architects can truly follow a PBD approach, since they can automatically generate and evaluate several designs. However, although this greatly simplifies the workload, architects are still required to analyze the results of each evaluation to decide which design should be evaluated next. Since the evaluation process can be very time-consuming, to relieve the architect from this burden, optimization algorithms can be used to select the next variation and guide the entire process.

The field of mathematical optimization has produced countless optimization algorithms, making the task of choosing an algorithm rather difficult. Moreover, as proven by the NFL theorems, "for any algorithm, any elevated performance over one class of problems is offset by performance over another class" [Wolpert and Macready, 1997], i.e., there is no algorithm which is the best in every single problem. However, according to recent studies discussed in section 2.2, metaheuristics, specifically GAs, are usually the first and only choice for architectural design optimization. This is surprising, because, as discussed in section 2.1, there is a large set of interesting algorithms which can be used, and some of them are perhaps better suited for the time-consuming optimization problems that are typical in architectural optimization. Besides this, the actual GA has several problems, namely the algorithm is highly dependent on the choice of hyperparameters and is deeply stochastic in nature, i.e., it is unstable and can return very different results for the same problem.

In order to explore these algorithms in practice, several tests were conducted. In general, across all conducted tests, metaheuristics were never the best category of algorithms. Other categories, namely, global direct-search and model-based algorithms, provided much better results. Several factors could change this ranking, such as a different set of hyperparameters for the algorithms, a lucky random start for a local algorithm, or a lucky random step taken by a metaheuristic. Therefore, the conclusion should be that it is preferable to try different algorithms during the optimization process. Still, it is important to consider that algorithms whose quality is highly dependent on luck should not be the first choice. Besides the choice of algorithm, the evaluation or time limit is also relevant. According to the tests of chapter 4, running just one algorithm for a long time appears to be wasteful, since after a certain point no major leaps in solution quality are made. Instead, allowing two or three different algorithms to run for a small amount of time has a higher chance of achieving a good result.

Based on the conclusions regarding algorithmic efficiency, the next logical step for architects would be to use several optimization algorithms in their workflow. As previously discussed, optimization can be used to guide the generation process according to the analysis results, following the PBD approach. However, although several optimization plugins exist for popular AD tools, the tools themselves only present a small subset of algorithms. Moreover, the interface of each tool is different, forcing architects to adapt in order to use several algorithms, yet again increasing their workload. To tackle this issue, this dissertation proposed an architectural design optimization framework in chapter 3. The proposed framework allows for the fully automated optimization of an architectural design, based on an algorithmic description provided by the architect, using several optimization algorithms and following the conclusions taken from the theoretical study of optimization. Besides the proposal of the framework, a prototype was implemented and tested, proving the real world capabilities of the framework. Also, some of the early theoretical conclusions were proved using the prototype and test cases.

Overall, as discussed in this dissertation, optimization can be a very useful technique for modern architecture, but existing tools make this a difficult and drawn-out process, which can lead to bad results if performed incorrectly. Also, the theoretical discussion of the GA and the obtained results indicate that GAs should not be the default choice. In fact, other interesting algorithms were identified and studied in the context of this dissertation. Finally, this dissertation shows that the combination of AD, ADA, and optimization, enables the automation of the architectural design optimization process. The framework of chapter 3 fulfills the final goal of this dissertation, by enabling a completely automated design optimization process, and providing a prototype which can be used in real world applications.

5.2 Future Work

In this dissertation, an automated framework for architectural optimization was proposed and tested using different case studies. However, there are still other interesting research paths that could be pursued in the future.

Regarding the prototype implementation of the framework, even though tests were performed, other relevant tests could be made using other performance metrics, e.g., lighting, cost, or acoustic metrics. Another type of interesting test would be comparing the prototype with other popular architectural optimization approaches. Such tests would make it possible to study the difference between designing and optimizing with several algorithms using different approaches.

Another interesting future research path includes exploring the idea of explainability, which already exists in the field of ML. Explainable ML approaches attempt to provide the user a way of understanding the results by returning not only the result, but also an explanation of it. These approaches would be easier to apply to surrogate-based optimization, because the surrogate model itself could be used to understand the relationship between the parameters and the results.

Finally, another possible research path involves providing even further automation. Currently, the proposed framework automates the generation and evaluation of designs according to optimization algorithms, but it depends on an initial design. In the future, the creation of this initial design could also be automated, e.g., through a set of predefined algorithmic template designs which could be adapted depending on the current situation. To do this, it would be necessary to create a sufficiently large set of adaptable algorithmic templates, either manually, or by using an ML algorithm.

Bibliography

[Aguiar et al., 2017] Aguiar, R., Cardoso, C., and Leitão, A. (2017). Algorithmic Design and Analysis Fusing Disciplines. In *Disciplines and Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, pages 28–37, Cambridge, USA.

[Branco and Leitão, 2017] Branco, R. C. and Leitão, A. (2017). Integrated Algorithmic Design: A single-script approach for multiple design tasks. In *ShoCK! - Sharing Computational Knowledge!: Proceedings of the 35th Education and Research in Computer Aided Architectural Design in Europe Conference (eCAADe)*, pages 729–738, Rome, Italy.

[Brent, 1973] Brent, R. P. (1973). *Algorithms for Minimization Without Derivatives*. Dover Publications.

[Caetano et al., 2018] 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. In *Learning, Adapting and Prototyping: Proceedings of the 23rd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, pages 111–120, Beijing, China.

[Christensen et al., 2014] Christensen, J. T., Parigi, D., and Kirkegaard, P. H. (2014). Interactive tool that empowers structural understanding and enables FEM analysis in a parametric design environment. In *Shells, Membranes and Spatial Structures: Proceedings of the 2014 Annual Symposium International Association for Shell and Spatial Structures (IASS)*, pages 1–8, Brasilia, Brasil.

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

[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. In *Future Trajectories of Computation in Design: Proceedings of the 17th Computer-Aided Architectural Design Futures Conference (CAAD Futures)*, pages 151–169, Istanbul, Turkey.

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

[Costa and Nannicini, 2018] Costa, A. and Nannicini, G. (2018). RBFOpt: an open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation*, 10(4):597–629.

[Eastman et al., 2008] Eastman, C., Liston, K., Sacks, R., and Liston, K. (2008). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Wiley Publishing.

[Fortin et al., 2012] Fortin, F.-A., De Rainvile, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175.

[Gablonsky and Kelley, 2001] Gablonsky, J. M. and Kelley, C. T. (2001). A Locally-Biased form of the DIRECT Algorithm. *Journal of Global Optimization*, 21(1):27–37.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary computation*, 9(2):159–195.

[Hare et al., 2013] Hare, W., Nutini, J., and Tesfamariam, S. (2013). A survey of non-gradient optimization methods in structural engineering. *Advances in Engineering Software*, 59:19–28.

[Himmelblau, 1972] Himmelblau, D. M. (1972). *Applied nonlinear programming*. McGraw-Hill.

[Ilunga and Leitão, 2018] Ilunga, G. and Leitão, A. (2018). Derivative-free Methods for Structural Optimization. In *Computing for a better tomorrow: Proceedings of the 36th Education and Research in Computer Aided Architectural Design in Europe Conference (eCAADe)*, pages 179–186, Lodz, Poland.

[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.

[Kalay and Mitchell, 2014] Kalay, Y. E. and Mitchell, W. J. (2014). *Architecture's New Media: Principles, Theories, and Methods of Computer-Aided Design*. The MIT Press.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

[Koziel and Yang, 2011] Koziel, S. and Yang, X.-S. (2011). *Computational Optimization, Methods and Algorithms*. Springer Berlin Heidelberg.

[Lopes and Leitão, 2011] Lopes, J. and Leitão, A. (2011). Portable Generative Design for CAD Applications. In *Integration through Computation: Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, pages 196–203, Banff, Canada.

[Marler and Arora, 2009] Marler, R. T. and Arora, J. S. (2009). The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, 41(6):853–862.

[Marsden et al., 2004] Marsden, A. L., Wang, M., Dennis, Jr., J. E., and Moin, P. (2004). Optimal Aeroacoustic Shape Design Using the Surrogate Management Framework. *Optimization and Engineering*, 5(2):235–262.

[McKay et al., 1979] McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245.

[Miller, 2011] Miller, N. (2011). The Hangzhou Tennis Center: A Case Study in Integrated Parametric Design. In *Parametrism: Proceedings of the 2011 Regional Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, pages 141–148, Lincoln, USA.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

[Oliphant, 2006] Oliphant, T. E. (2006). *A guide to NumPy*. Trelgol Publishing.

[Oxman, 2006] Oxman, R. (2006). Theory and design in the first digital age. *Design Studies*, 27(3):229–265.

[Pedregosa et al., 2012] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Powell, 1994] Powell, M. J. D. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, 275:51–67.

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

[Preisinger and Heimrath, 2014] Preisinger, C. and Heimrath, M. (2014). Karamba - A Toolkit for Parametric Structural Design. *Structural Engineering International*, 24(2):217–221.

[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.

[Roudsari and Pak, 2013] Roudsari, M. S. and Pak, M. (2013). Ladybug: A Parametric Environmental Plugin for Grasshopper To Help Designers Create an Environmentally-Conscious Design. In *Proceedings of the 13th Conference of International building Performance Simulation Association*, pages 3129 – 3135, Chambéry, France.

[Rowan, 1990] Rowan, T. (1990). *Functional stability analysis of numerical algorithms*. Ph.d. thesis, University of Texas at Austin, USA.

[Runarsson and Yao, 2000] Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294.

[Runarsson and Yao, 2005] Runarsson, T. P. and Yao, X. (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 35(2):233–243.

[Santos, 2010] Santos, C. d. S. (2010). *Computação bio-inspirada e paralela para a analise de estruturas metamateriais em microondas e fotonica*. Ph.d. thesis, University of Campinas, Brasil.

[Sidelko, 2013] Sidelko, J. (2013). *Museo Soumaya: Facade Design to Fabrication*. Primedia E-launch LLC.

[Terzidis, 2006] Terzidis, K. (2006). *Algorithmic Architecture*. Routledge.

[Ward, 1994] Ward, G. J. (1994). The RADIANCE lighting simulation and rendering system. In *Proceedings of the 21st Annual Conference on Computer Graphics and iIteractive Techniques (SIGGRAPH)*, pages 459–472, Orlando, USA.

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

[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. In *Living Systems and Micro-Utopias - Towards Continuous Designing: Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, pages 177–186, Melbourne, Australia.

[Wortmann and Nannicini, 2017] Wortmann, T. and Nannicini, G. (2017). Introduction to Architectural Design Optimization. In *City Networks - Planning for Health and Sustainability*, volume 128, chapter 14. Springer International Publishing.

[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? In *Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD)*, pages 51–58, Toronto, Canada.

[Zavala et al., 2014] Zavala, G. R., Nebro, A. J., Luna, F., and Coello Coello, C. A. (2014). A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, 49(4):537–558.