Single-Objective Optimization for Architecture [Extended Abstract]

Guilherme Ilunga
Instituto Superior Técnico, Lisboa
guilherme.ilunga@tecnico.ulisboa.pt

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

INTRODUCTION

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

Usually, architects use Computer-Aided Design (CAD) and Building Information Modelling (BIM) tools to develop the building design, but these tools are unable to properly handle repetitive complex changes, such as those that may be suggested by the engineers. For example, in a design of a space frame which follows a sinusoidal shape, 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 each individual position and size of the bars in the space frame, which is extremely time-consuming. To support these types of changes, new paradigms have been developed, such as the Algorithmic Design (AD) paradigm.

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. 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[1] is a plugin for the Rhinoceros3D CAD tool[2] and is the most popular AD tool among architects [Cichocka

[1] http://www.grasshopper3d.com/, accessed 25 April 2018.
[2] https://www.rhino3d.com/, accessed 25 April 2018.

et al., 2017a]. It follows the visual programming paradigm using a simplistic, intuitive, and easy to use approach, 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. The problem of visual programming tools, such as Grasshopper, is that the visual approach does not scale well. Figure 1 shows one example of an algorithm for a complex design, where 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 and to generate models on several design 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 Application Programming Interface (API) by its users. 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.

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. The problem is which of the newly generated variations should be chosen? Typically, the engineering team's opinion carries a large weight, since they are tasked with ensuring that a building follows regulations and also with evaluating the building's performance regarding different criteria, e.g., structural, lighting, energy consumption, and cost. However, ideally, the architect should be able to automatically infer the performance of the different designs, by using automated tools.

In order to evaluate a building's performance, besides pen-and-paper calculations, numerical analysis and simulation-based tools can be used. These types of tools can, for example, simulate a building's structural behavior given a specific load using finite element analysis, or compute the useful daylight illumination using raytracing techniques. To do this, the tools require a special type of model, different from the geometrical 3D model, called an analytical model.

Analytical models can be generated in BIM tools that have that capability, or they may be modeled directly in the analysis tool, or, when using an AD approach, by accessing the API of the analysis tool. Using the latter method together with a generative process for creating the models makes it possible to guide the generation of the design based on the results of the analysis, following Performance-based Design (PBD) [Oxman, 2006]. A recent approach, called Algorithmic Design and Analysis (ADA), gives the AD tool the ability to generate different analytical and 3D models from the algorithm provided by the architect, enabling the automatic analyses of different variations of the design [Aguiar et al., 2017], making it 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. However, this approach still requires the architect to manually select the values for the different parameters, before setting off the automatic generation-and-evaluation process. Ideally, optimization algorithms could be used at this stage, to automatically decide the values, i.e., guide the process.

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 or a set of functions, depending on whether it is Single-Objective or Multi-Objective optimization, respectively. The goal of the process can be to either obtain the maximum or minimum possible value of the function. To reach the goal, an optimization algorithm changes the parameters, according to a set of constraints. Besides the constraints, the parameters themselves may have bounds, i.e., minimum and maximum possible values.

In order to use the knowledge and ideas from the field of mathematical optimization for the benefit of architectural projects, it is necessary to translate an architectural optimization problem into a mathematical optimization problem, where the metric to optimize is treated as a function. The parameters of the optimization can be the same that were defined in an AD approach. As for the function to optimize, unfortunately, in most cases, it does not have a known mathematical expression. To overcome this problem, analysis tools need to resort to simulation techniques, such as raytracing or finite element analysis, to approximate the function. Additionally, in order to optimize such an unknown function, one needs to use a particular kind of optimization, named black-box optimization, in which the function is treated as a black-box. Unfortunately, the usage of simulation tools for optimization entails a performance penalty. This means that black-box optimization processes need to use a reduced budget of function evaluations. With this approach, existing optimization algorithms can be used directly in architectural design, to obtain the best performing design according to specific criteria.

The goals of this dissertation are to study existing Single-Objective optimization algorithms and apply them to case studies, to evaluate their performance. Also, taking into account the No Free Lunch (NFL) theorems, which state that no algorithm can consistently perform better than all others on all problems [Wolpert and Macready, 1997], this dissertation proposes an automated optimization framework for architecture which allows the usage of several different types of algorithms. 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, specifically the belief that Genetic Algorithms (GAs) are the best algorithm to use.

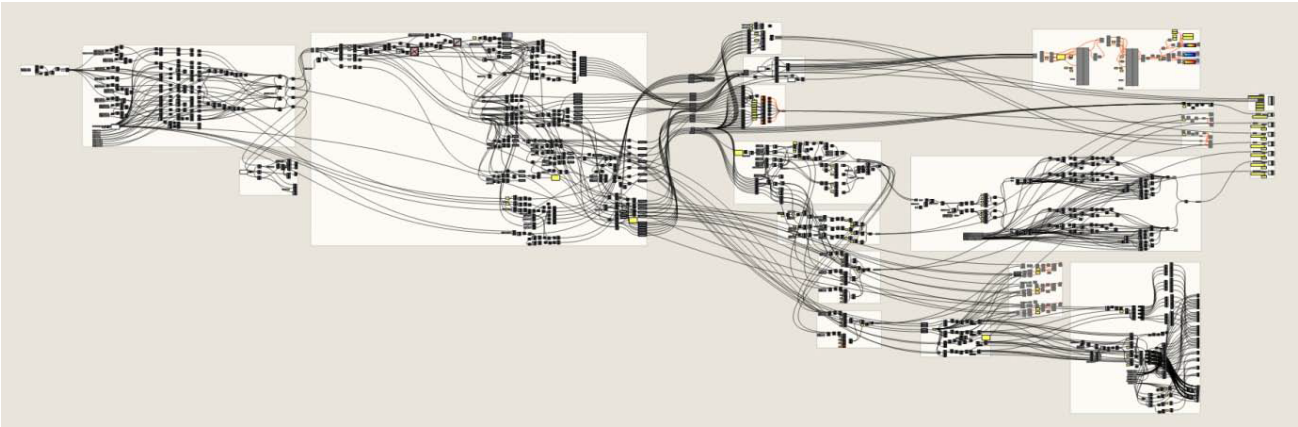

Figure 1: The complete algorithm for the Hangzhou Tennis Center in the Grasshopper tool. Adapted from [Miller, 2011].

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 direct-search, metaheuristics, and model-based algorithms, and also provides examples of specific algorithms.

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 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 algorithms 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.

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 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.

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.

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[3], Goat[4], Silvereye[5] [Cichocka et al., 2017b], and Opossum[6] [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[7], 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

[3] http://www.grasshopper3d.com/profiles/blogs/evolutionary-principles, accessed 15 May 2018.
[4] https://www.rechenraum.com/en/goat.html, accessed 20 May 2018.
[5] https://www.food4rhino.com/app/silvereye-pso-based-solver, accessed 10 February 2019.
[6] https://www.food4rhino.com/app/opossum-optimization-solver-surrogate-models, accessed 10 February 2019.
[7] http://ab-initio.mit.edu/nlopt, accessed 5 March 2018.

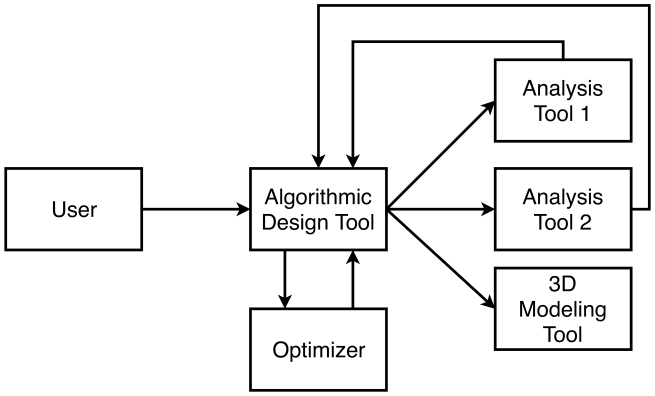

Figure 2: The ADA workflow with an optimizer module. In this workflow, the user only needs to interact with the AD tool and to select the options for the optimizer, enabling the automatic optimization of the design.

and Nannicini, 2017]. Therefore, it is interesting to assess whether GAs can realistically be the first and only choice, or if better options exist.

FRAMEWORK DESCRIPTION

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 the previous section 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, 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 2, 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.

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.

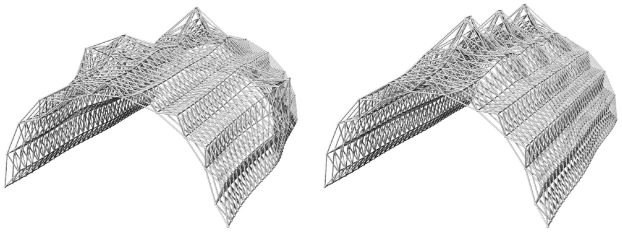

Figure 3: Two variations of a space frame with three attractor points, which give it a non-uniform shape.

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.

CASE STUDY EVALUATION

This section focuses on the evaluation of both the proposed framework and some of the algorithms that have been implemented in the current prototype. To do this, an architectural case study was created. This 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 3 illustrates two examples of the structure.

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,

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 3 times, and both the standard deviation and mean of the results are used to draw conclusions.

Finally, when comparing the different algorithms, besides the categories and properties described in section 2, 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

At the time of performing these tests, the prototype optimizer contained all algorithms except the CMA-ES, PSO, and simple surrogate algorithms. For testing the remaining algorithms, due to how time-consuming each evaluation was, an evaluation limit of 100 evaluations is set.

Figure 4 illustrates the mean best result, as a function of the number of evaluations, for the previously defined groups. 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.

Regarding the result of each individual algorithm, table 1 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 metaheuristics are not the best type of algorithms to solve the problem, as they are beaten by direct-search and model-based algorithms.

Table 1: 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

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 3, metaheuristics, specifically GAs, are usually the first and only choice for architectural design optimization. This is surprising, because, as discussed in section 2, 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

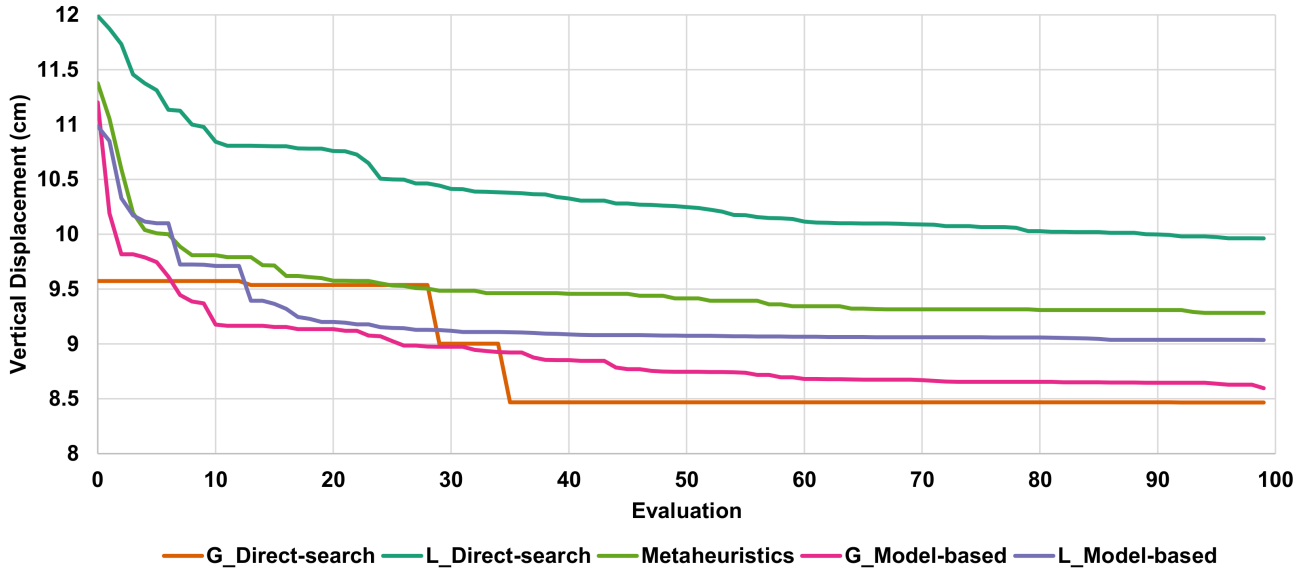

Figure 4: Mean performance results for several optimization algorithms applied to the space frame case study. The global direct-search and global model-based algorithms obtained the best results.

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, proving the previously outlined hypothesis. 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, 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 section 4. 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.

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 section 4 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.

REFERENCES

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

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.

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

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.

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

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

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.

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.

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

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

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.

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.

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

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

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

Price, W. L. (1983). Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40(3):333–348.

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

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

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

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, 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.

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

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

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, 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, 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, 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, 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.