More is More: The No Free Lunch Theorem in Architecture

Inês Pereira[1] and António Leitão[2]
[1, 2] *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal*
{ines.pereira[1], antonio.menezes.leitao[2]}@tecnico.ulisboa.pt

Abstract: The growing interest in sustainability and environmental design promoted the development of tools that respond to the architects' demand for efficient ways to evaluate building performance and optimize their designs. To optimize a design, a parametric model of that design is iteratively instantiated and evaluated by an optimization algorithm that searches for the solutions that best fulfil a set of performance objectives. However, according to Wolpert´s No-Free-Lunch Theorem, different optimization problems are best served by different optimization algorithms and, thus, architects should test multiple ones. Unfortunately, this is not a straightforward task for the currently available algorithmic design tools because different optimization algorithms have different requirements, forcing architects to spend considerable time and effort to adapt their parametric models and simulation tools, to configure the optimization algorithms, and to visualize the optimization results. This paper addresses the beforementioned problems by presenting the integration of a wide range of optimization within the same algorithmic design tool. We discuss the use of this tool in a case study that demonstrates the usefulness of multiple optimization algorithms to avoid the potential non-optimality that emerges from using just one of them.

Keywords: Architectural optimization, No free lunch theorem, Parallelization

1. Introduction

The adoption of building optimization strategies has been growing as a result of the increasing interest in sustainability and environmental design, promoting the development of tools that respond to the architects' demand for efficient ways to evaluate performance and optimize designs (Touloupaki and Theodosiou, 2017). To this end, architects resort to Algorithmic Design (AD) tools to create the necessary parametric models and then evaluate their performance using analysis tools. This can then be coupled with an optimizer to iteratively create and evaluate design variations in the search for a better- performing alternative. However, there are many optimizations algorithms available and it is difficult to know a priori which algorithm is the most adequate for solving a particular design problem. In this context, a wide range of algorithms should be tested, especially at initial design stages, to allow architects to understand which one is more suitable for their problem, and confidently resort to that one during the remaining stages of the project development.

Imaginable Futures: Design Thinking, and the Scientific Method. 54th International Conference of the Architectural Science Association 2020, Ali Ghaffarianhoseini, et al (eds), pp. 765–774.

Architectural optimization is usually simulation-based, meaning the objective functions are computed through time-intensive analysis, resulting in highly time-consuming and often time-unfeasible processes. For this reason, many architects refrain from using algorithmic optimization in their daily practice. Moreover, this becomes much more problematic when they not only need to test multiple optimization algorithms but also these are scattered across different optimization tools. This implies that changes to the optimization setup, even if small, must be made to adapt to the requirements of the new tool.

In this paper, we proposed an approach that addresses these problems by integrating a wide range of optimization algorithms in the same AD environment and allowing for the simultaneous testing of multiple optimization processes. The proposed approach reduces the time needed to test different optimization algorithms, increasing the confidence in the results, and motivating architects to continually employ such methodologies in their daily practices.

2. Architectural optimization

Architectural optimization problems tend to have multiple objectives that often conflict with each other, meaning the best solution for one objective might have poor performance in the others. Additionally, architectural optimization is frequently criticized for not considering all aspects of the design, focusing on a few selected objectives, and not giving sufficient control to the architect that, ultimately, is responsible for the design decisions. In this view, one of the main goals of architectural optimization is to find a way to efficiently reconcile performance and design freedom.

Pareto optimization provides an attractive approach to achieve that goal. This is a Multi-Objective Optimization (MOO) approach that finds a set of solutions where none of the objectives can be improved without degrading some of the others (Khazaii, 2016). This set of optimal solutions represent the trade-offs found by the optimization algorithm and embody the so-called Pareto front. By producing not just one but a possibly large set of optimal solutions, Pareto optimization allows the architect to select from this set what he considers the best solution, satisfying not only the measurable objectives used during the optimization but also personal ones.

3. No Free Lunch Theorem

Architectural optimization problems are often simulation-based, meaning the objective functions do not have a known mathematical expression and, instead, are calculated through analysis tools. Since it is not possible to extract information from the objective functions to guide the optimization process, black-box optimization algorithms are more suitable to address architectural optimization problems (Wortmann and Nannicini, 2016). These can be sub-divided into three main classes: **metaheuristic algorithms,** which are inspired by natural phenomena and biological analogies, for example, evolution by natural selection or swarm behaviours; **direct-search algorithms**, which sequentially evaluate design solution through deterministic methods; and **model-based algorithms**, which create an approximation of the black-box objective functions and then iteratively refine it as the optimization progresses.

Interestingly, according to Wolpert's No Free Lunch Theorem, no optimization algorithm constantly outperforms all others in all problems (Wolpert and Macready, 1997). Therefore, knowing which optimization algorithms are most suitable for a specific problem requires testing a heterogeneous set of algorithms. Moreover, each algorithm makes different assumptions regarding the problem they aim to solve, thus testing multiple ones increases the chances of finding the best match for a specific problem.

This, in turn, can result in large optimization gains. Yet, testing different optimization algorithms is time-consuming, especially in building performance optimization, where complex simulations are necessary.

In general, optimization requires a large number of evaluations of the objective functions. However, in simulation-based optimization, the evaluations are computationally expensive and, thus, it is necessary to use optimization algorithms capable of produce satisfactory results with a small sample of the design space (Wortmann, 2019).

Model-based algorithms are faster at calculating objective functions, since the time-consuming simulations are replaced by an approximation, and have been showing promising results, particularly in Single-Objective Optimization (SOO) (Wortmann et al., 2017). However, to produce better predictions these algorithms need to sample, using simulation-based evaluations, a sufficiently large fraction of the design space, which may hinder the speedup gained by using the approximation. Additionally, the architectural field seems to prefer a sub-class of the metaheuristics – the evolutionary algorithms. However, these may not be the best option as they usually need a high number of evaluations to achieve the same results as other classes of algorithms.

4. Architectural optimization tools

The interest in architectural design optimization has been increasing over the last two decades, as is visible in the growing number of optimization tools available. These tools are usually developed as plugins for AD tools, such as Grasshopper or Dynamo, allowing architects to easily couple parametric modelling and building performance simulations with optimization strategies. The following sections present an overview of the most well-known optimization tools for architecture, along with the challenges architects may face when they want to test multiple optimization algorithms.

Despite the variety of optimization tools currently available, they significantly differ in the type of problems they can solve (e.g., single or multiple objective) and in the number and class of optimization algorithms they offer.

4.1. Grasshopper

Grasshopper for Rhino is a popular AD tool where architects can develop their designs by dragging and connecting graphical components. The following paragraphs present some of the optimization plugins that were developed specifically for Grasshopper.

Galapagos (Rutten, 2013) is a SOO plugin that uses two generic solvers: a genetic algorithm and a simulated annealing algorithm. It offers a simple, well-organized, and user-friendly interface where all the necessary options for the optimization have pre-defined defaults. On the one hand, its simplicity encourages designers with little experience in optimization to start integrating optimization strategies in their workflow. On the other hand, more experienced designers may feel limited when using it.

Wallacei (Makki et al., 2020) supports MOO, allowing users to run the well-known evolutionary optimization algorithm NSGA-II (Deb et al., 2002). It also provides detailed analytic tools coupled with a wide range of visualization options to assist users in understanding the results produced and, therefore, making more informed decisions.

Opossum (Wortmann, 2017) supports SOO and MOO, and includes metaheuristic algorithms and model-based ones for both optimization problems. For SOO it provides the model-based RBFOpt (Costa and Nannicini, 2018) and the evolutionary-based CMA-ES (Hansen et al., 2006), and for MOO, the

model-based RBFMOpt, the evolutionary-based MOEA/D (Rubio-Largo et al., 2014) and NSGA-II, the ant-colony-based MACO (Sun et al., 2010), and, finally, the particle swarm-based NSPSO (Liu, 2008). Additionally, Opossum addresses different levels of expertise, has a user-friendly and ready-to-use interface, with pre-defined default values for each algorithm, which can be re-configured by more experienced users.

Octopus (Vierlinger, 2020) is based on the MOO evolutionary algorithms SPEA2 (Zitzler et al., 2001) and HypE (Bader and Zitzler, 2011), and already includes some model-based algorithms. It works in a similar way to Galapagos but includes the concept of Pareto optimization. Octopus' interface is as user-friendly as the one of Galapagos, generating multiple visualizations that help the user to better understand the optimization results.

Goat (Rechenraum, 2016) uses the NLopt library (Johnson, 2008) to provide SOO algorithms from the black-box-optimization classes mentioned in Section 3. It has the metaheuristic algorithms CRS2, the direct-search algorithms DIRECT and SUBPLEX, and the model-based algorithms COBYLA and BOBYQA. Moreover, Goat is deterministic, i.e., running the same algorithm with the same parameters will always yield the same result. This feature allows for the reproducibility of results, a valuable quality for scientific work. On the other hand, it only returns a single optimal solution, which is not appreciated by most architects.

Silvereye (Cichocka et al., 2017) is a SOO plugin based on the PSO algorithms. The main goal of Silvereye is to help inexperienced users solve complex real-world optimization problems. It has a simple and user-friendly interface, with default values for the algorithms' configuration. These can be updated by the user, and the tool will automatically save the new values to be used as the default configuration the next time the plugin is loaded.

4.2. Dynamo

Dynamo is another well-known AD tool. It is also based on a visual programming language, but differently from Grasshopper, it is integrated with the BIM tool Revit. Although in smaller numbers, there are also optimization plugins developed specifically for Dynamo. Here, we present the one that is most used by practitioners of this AD tool.

Optimo (Asl et al., 2015) uses NSGA-II to solve both SOO and MOO problems. It does not provide users with a user-friendly graphical interface, meaning even post-processing and visualizations must be done through components and are showed as background of the Dynamo environment. This may keep the less experienced users away from using the tool.

4.3. Comparison

Table 1 summarizes the features we consider more relevant for this research from each one of the plugins presented previously. From all the tools we analysed, Opossum is the tool with a more diverse offer of optimization algorithms and the only Grasshopper plugin that includes options for both SOO and MOO. Octopus and Goat also have a fair number of algorithms and, similarly to Opossum, provide model-based algorithms. Moreover, Goat is the only one that provides direct-search algorithms. Finally, current optimization plugins already provide all classes of black-box optimization for both SOO and MOO, except direct-search for the later, a combination that is still being researched.

Table 1: Design optimization plugins and their corresponding optimization algorithms.

AD Tool	Optimization Tool	Objectives	Number of Algorithms	Optimization Algorithms			
				Metaheuristics		Direct-Search	Model-Based
				Evolutionary	Others		
Grasshopper	**Galapagos**	SOO	2	**x**	-	-	-
	Wallacei	MOO	1	**x**	-	-	-
	Opossum	SOO	2	**x**	-	-	**x**
		MOO	5	**x**	**x**	-	**x**
	Octopus	MOO	**5**	**x**	-	-	**x**
	Goat	SOO	5	**x**	-	**x**	**x**
	Silvereye	SOO	1	-	**x**	-	-
Dynamo	**Optimo**	SOO/MOO	1	**x**	-	-	-

5. Discussion

To address the challenges mentioned previously, we coupled a textual-based AD tool with a set of simulation-based evaluation tools and with a parallelized optimization framework that relies on a unified description of the optimization problem. This combination (1) provides a large variety of optimization algorithms, (2) parallelizes the optimization processes, and, finally, (3) supports post- processing and visualizations techniques to compare optimization results more easily.

5.1. Optimization problem

To test the proposed approach, we developed a case study that consists of a skylight truss. The skylight is divided into multiple truss pyramids where the height of each one changes randomly within a pre- defined range. Additionally, the number of pyramids can also vary. Figure 1 shows some variations regarding the number of pyramids (a to c, and b to d), and the maximum height (a to b, and c to d).

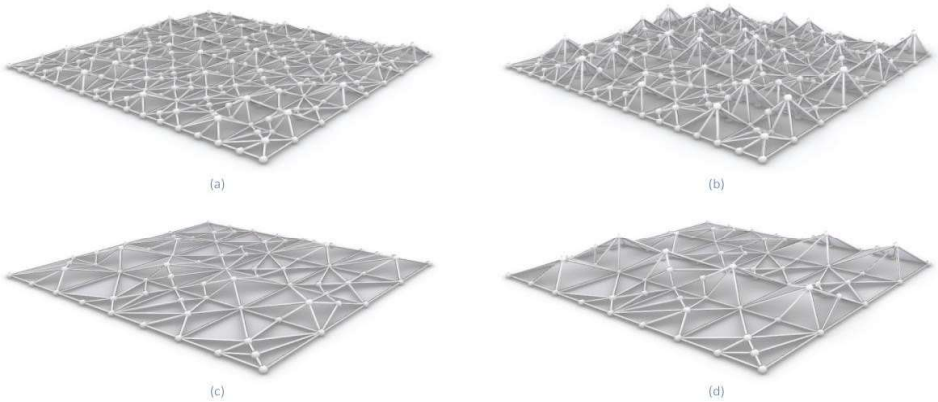

Figure 1: Variations of the truss skylight.

The number of pyramids (*n*) and the maximum height of the pyramids (*h*), as well as the material of the structural bars (*mat*) and its radius(*r*), were considered as the variables of our optimization problem. As objectives, we were interested in improving the structural behaviour of the skylight, measured in terms of the structure maximum displacement, while reducing its cost, measured in terms of the structure's volume and the materials cost per m^3, as described in Equations 1 and 2. These are conflicting objectives because a material with better structural performance is also more expensive. Thus, the goal for the optimization is to reconcile these objectives by finding the set of design solutions for the truss skylight that represent the best trade-off between structural performance and cost.

$$\boldsymbol{min}\ f(n,h,mat,r) = max\_displacement(n,h,mat,r) \quad (1)$$

$$\boldsymbol{min}\ h(n,h,mat,r) = total\_volume(n,h,mat,r) \times cost(mat) \quad (2)$$

To test the parallelization approach and to find out which optimization algorithm yields the best results for this case study, we tested nine different ones: six metaheuristics and three model-based algorithms. To allow a fair comparison with the previously presented optimization plugins, we excluded the direct-search class. As for the metaheuristic class, we tested the particle swarm-based SMPSO and OMPSO, and the evolutionary-based SPEA2, NSGA-II, MOEA/D, and PESA2 (Corne et al., 2001). Concerning the model-based class, to create the approximation of the objective functions we tested the Random Forest Regressor (RFR), the Extra Tree Regressor (ETR), and the Gaussian Process Regressor (GPR) (with a radial basis function as kernel), and we combined them with a metaheuristic solver (SPEA2).

Additionally, to allow a fair comparison between the different algorithms, they were limited to the same maximum number of evaluations: 600. This value corresponds to a very small percentage of this problem's design space, more precisely to 4.3%. Moreover, all of the metaheuristic algorithms, which in our case are population-based, were given the same initial population, computed through the Latin hypercube algorithm, and the same number of iterations (20), resulting in a population size of 30

elements. The model-based algorithms work in a different way than the beforementioned class, thus, were defined in a more suitable way. To construct the approximations, we initially produced 420 random evaluations, corresponding to 70% of the maximum possible evaluations, and then, leave the remaining 30% to be evaluated during the optimization. We performed three runs to minimize the randomization factor inherent to each algorithm and to have more informed conclusions about the performance of each one, resulting in 27 optimization processes.

5.2. Optimization results

Once we had the results of all three runs of the optimization algorithms tested, we produced the corresponding Pareto front to compare the algorithm's performance. The results are illustrated in Figure 2, where one Pareto front contains all of the optimal solutions found during the multiple runs.

From this figure, we can clearly observe that the particle swarm algorithms were the ones that showed the worst performance in this problem, particularly, OMOPSO was the algorithm with the poorest performance. The evolutionary algorithms and model-based ones exhibited similar performance, with the former being slightly worse than the latter. This shows one of the main drawbacks of using Pareto fronts to compare the performance of algorithms and reinforces the need to use other metrics to evaluate the performance of the optimization algorithms.

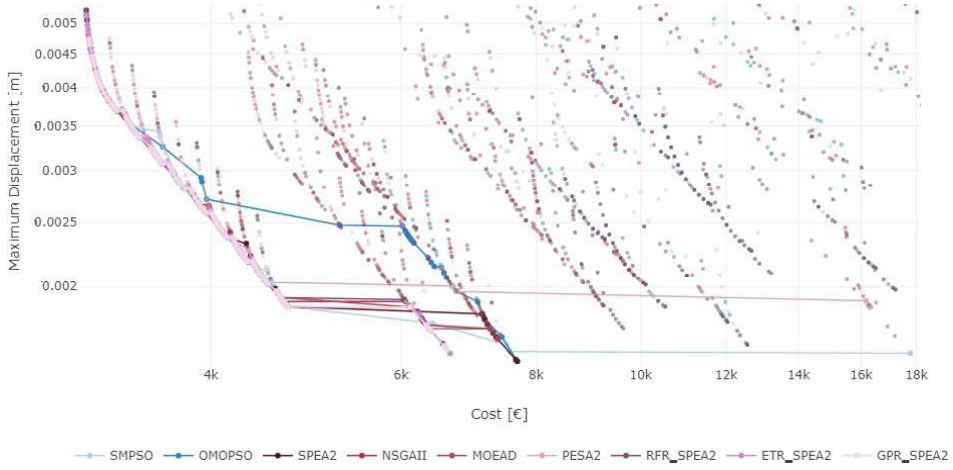

Figure 2: Pareto fronts for the optimization algorithms tested.

To have more in-depth performance evaluation and to determine which of the tested algorithms is the more adequate for our optimization problem, we decided to use the hypervolume and the Overall Non-dominated Vector Generation (ONVG) indicators. The former measures the volume of the dominated space and the latter measures the number of optimal solutions. Moreover, to understand how the optimizations evolve over time, we measure these indicators per iteration, as illustrated in Figure 3. It is noteworthy that for the model-based algorithms, only the solutions found through the

approximations are represented. From the information in this figure, we can take more concrete conclusions about the performance of the algorithms. Particularly, the model-based RFR_SPEA2 and ETR_SPEA2 were not only the algorithms that more quickly achieved a higher value of hypervolume, but also the ones that found more optimal solutions.

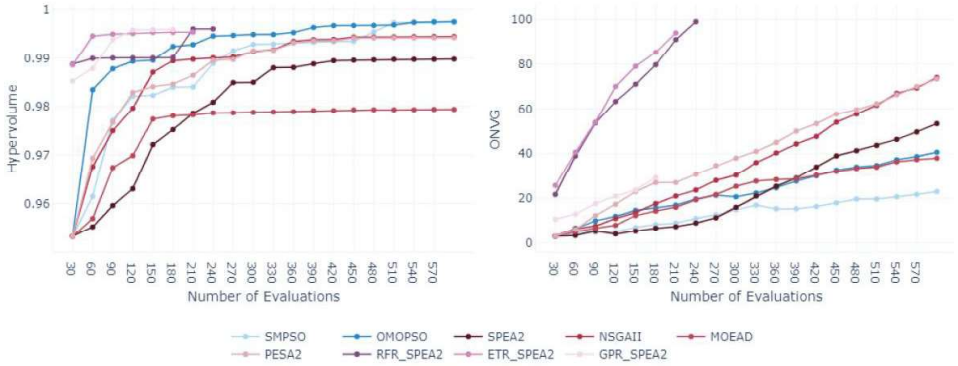

Figure 3: Hypervolume and ONVG indicators per evaluation.

5.3. Parallelization results

When a set of tasks can be executed in parallel, the time needed to complete all of them depends only on the longest one. This means that the maximization of the parallelization speedup requires that all tasks take approximately the same time to complete. However, in our case study, we observed that the SMPSO algorithm took, on average, twice as much time to complete as the second slowest one, which caused a decline in the speedup factor. Given that our approach supports selective cancelling of the optimization processes that are running in parallel, it is up to the architect to decide if the solutions produced by other optimizations are good enough to allow for early termination. As a result, to better evaluate the parallelization effort, we divided it into two parts: one where the outlier algorithm is not considered, and another where it is. In the first case, assuming that we have as many processing cores as needed and taking into account the actual time need for each optimization the theoretical speedup was 13.3. However, as the number of tasks (21) exceeded the capabilities of the available hardware (only eight cores with hyper-threading), the observed speedup was 6.7. In the second case, as we increased the number of tasks (27) to include SMPSO's runs, the theoretical speedup decreases to 8.6 but the actual one was 2.9.

Despite being considerably below the theoretical speedups, the results represent a significant improvement in the time needed to test multiple optimization algorithms, and more so for the first case. This means that the parallelization has the potential to motivate architects already interested in optimization to test a wider range of algorithms, while simultaneously appealing to those interested in environmental design, but afraid of the time-consuming aspect of optimization. Finally, results show that even in machines with limited resources, parallelization approaches can considerably reduce the computational time needed for experimenting with the different optimization algorithms. In fact, in

other case studies, tested in more powerful hardware, we have experienced larger speedups that reach one order of magnitude.

6. Conclusions and Future work

Architectural optimization can help architects achieve more performative and sustainable designs. To this end, architects must use optimization algorithms that are adequate for their design problem. However, in general, this adequacy is not known beforehand. Therefore, for each optimization problem, it is important to test a wide range of optimization algorithms. Unfortunately, the currently typical optimization workflow is not practical, forcing architects to (1) switch between tools and (2) test algorithms sequentially, considerably increasing the time spent in what is already a highly time- consuming process.

To address these problems, we proposed a framework that combines an algorithm design tool with simulation and optimization tools, allowing architects to use a unified description of the optimization problem to test multiple optimization algorithms. Moreover, we parallelized it to run multiple optimization processes simultaneously, thus, speeding up the identification of the best-performing optimization algorithm. This allows architects to worry less about the details of the optimization and, instead, direct their attention towards the creative process of design development, while still ensuring a high-performing final design.

One of the limitations of this research is that, differently from what the architectural field prefers, it uses a textual-programming language. Textual-programming languages have a higher learning curve when compared with visual ones. However, they scale better, meaning algorithmic descriptions of complex designs become easier to understand and, therefore, to optimize. Another limitation to consider is the hardware resources needed for an effective parallelization. Although the evaluation was performed on an off-the-shelf laptop, the results show that even in this case the speedup obtained justifies the adoption of parallelization approaches in optimization problems. Moreover, given that the current hardware evolution points in the direction of an increasingly larger number of processing units, it is expected that the parallelization speedups will improve in the future.

As future work, we will explore additional parallelization approaches to further increase the speedup. In particular, we plan to also parallelize the evaluation of individuals in population-based optimization algorithms, as well as the evaluation of a single individual for embarrassingly parallel objective functions, such as those based on lighting analysis or acoustic analysis.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and PTDC/ART-DAQ/31061/ 2017.

References

Asl, M., Stoupine, A., Zarrinmehr, S., et al. (2015) “Optimo: A BIM-based Multi-objective Optimization Tool Utilizing Visual Programming for High Performance Building Design.” In *Proceedings of the 34th Education and Research in Computer Aided Architectural Design in Europe (eCAADe) Conference*. 2015. pp. 673–682.

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

Cichocka, J., Migalsk, A., Browne, W., et al. (2017) “SILVEREYE – The Implementation of Particle Swarm Optimization

Algorithm in a Design Optimization Tool." In *Future Trajectories: Proceedings of the 17th International Conference on Computer-Aided Architectural Design Futures (CAAD Futures)*. 2017. pp. 151–169.

Corne, D., Jerram, N., Knowles, J., et al. (2001) "PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization." In *Proceedings of the Genetic and Evolutionary Computation Conference*. 2001. pp. 283–290.

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.

Deb, K., Pratap, A., Agarwal, S., et al. (2002) A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2): 182–197.

Hansen, N., Müller, S.D. and Koumoutsakos, P. (2006) Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11 (1):1–18.

Johnson, S.G. (2008) *The NLopt nonlinear-optimization package*. Available at: http://github.com/stevengj/nlopt (Accessed: 16 August 2020).

Khazaii, J. (2016) "chapter 11: Pareto-Based Optimization." In *Advanced Decision Making for HVAC Engineers: Creating Energy Efficient Smart Buildings*. Springer. pp. 99–115.

Liu, Y. (2008) "A fast and elitist multi-objective particle swarm algorithm: NSPSO." In *Proceeding of the 2008 IEEE International Conference on Granular Computing*. 2008. pp. 470–475.

Makki, M., Showkatbakhsh, M. and Song, Y. (2020) *An Evolutionary Multi-Objective Optimization and Analytic Engine for Grasshopper 3D*. Available at: https://www.wallacei.com/ (Accessed: 11 August 2020).

Rechenraum (2016) *goat*. Available at: https://www.rechenraum.com/en/goat.html (Accessed: 15 August 2020).

Rubio-Largo, Á., Zhang, Q. and Vega-Rodríguez, M.A. (2014) MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *Information Sciences*, 289 (1): 91–116.

Rutten, D. (2013) Galapagos: On the logic and limitations of generic solvers. *Architectural Design*, 83 (2): 132–135.

Sun, X., You, X. and Liu, S. (2010) "Multi-objective Ant Colony Optimization Algorithm for Shortest Route Problem." In *Proceeding of the 2010 International Conference on Machine Vision and Human-machine Interface*. 2010. pp. 796–798.

Touloupaki, E. and Theodosiou, T. (2017) Performance simulation integrated in parametric 3D modeling as a method for early stage design optimization - A review. *Energies*, 10 (5): 637–655.

Vierlinger, R. (2020) *Octopus*. Available at: https://www.food4rhino.com/app/octopus (Accessed: 14 August 2020).

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. (2017) "Opossum: Introducing and Evaluating a Model-based Optimization Tool for Grasshopper." In Janssen, P., Loh, P., Raonic, A., et al. (eds.). *Protocols, Flows and Glitches: Proceedings of the 22nd d International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2017*. Suzhou, China, 2017. pp. 283–292.

Wortmann, T. (2019) Genetic evolution vs. function approximation: Benchmarking algorithms for architectural design optimization. *Journal of Computational Design and Engineering*, 6 (3): 414–428.

Wortmann, T. and Nannicini, G. (2016) "Black-box optimisation 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)*. 2016. pp. 177–186.

Wortmann, T., Waibel, C., Nannicini, G., et al. (2017) "Are genetic algorithms really the best choice for building energy optimization?" In *SIMAUD '17: Proceedings of the Symposium on Simulation for Architecture and Urban Design*. 2017. pp. 41–48.

Zitzler, E., Laumanns, M. and Thiele, L. (2001) *SPEA2: Improving the strength pareto evolutionary algorithm*.