ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Vegetale: Automated Human-In-The-Loop Design of Vertical Gardens Through Black-Box Optimization Methods

Research in Computer Science

Pouya Pourjafar Kolaei

October, 2020

Supervisors: Prof. Dr. Andreas Krause,
Dr. Fernando Perez Cruz,
Dr. Luis Salamanca

Swiss Data Science Center, ETH Zürich

Abstract

The design of architectural structures is a challenging and time consuming task. One major difficulty is satisfying safety and buildability constraints while simultaneously fulfilling the original aesthetic visions of the designer. In this work we tackle the problem of automating mundane and time consuming aspects of architectural design such as approximating desired attributes and satisfying buildability constraints. We formulate the design task as a combinatorial black-box optimization problem and employ a combinatorial Bayesian optimization algorithm to find optimal solutions. We develop an end-to-end human-in-the-loop pipeline that generates and optimizes a diverse set of vertical gardens which allows the human designers more freedom and time to focus on creative aspects of the process.

Contents

Chapter 1

Motivation and Background

1.1 Motivation

The task of designing architectural structures is a complicated and time consuming process. Often times, a skilled expert who has many years of experience needs to redo the same process multiple times to arrive at a satisfying solution. During the design process, the architect wants to fulfill an artistic and creative concept whilst having to simultaneously take care of constraints that are imposed by the physical world and also having to satisfy some attributes which are needed by the client. An example of a constraint could be the buildability of a structure or the amount of material used. For an attribute, one could think of the amount of sun light that comes through the structure during the day. The major difficulties of such problems arise when such features have to be met. Since for a given design concept, no interpretable model exists that returns a structure satisfying the above mentioned conditions, an architect ends up redoing the same process again and again, each time adjusting the design to more rigorously meet the constraint or better approximate the attributes. Facing such hurdles, fulfilling the original vision of the design can become impossible.
In this work, we aim at automating some of the above mentioned challenges enabling the architects to focus more on the creative and aesthetic aspects of the design rather than on the mundane satisfiability constraints. We tackle the specific problem of designing *vertical gardens*, but believe that our developed methodology is well adaptable to other domains as well. First, we give a short overview of the specific design problem which we are concerned with. Next we further motivate the need for our developed methodology by discussing some of the main challenges that an architect faces and how such a tool could be helpful. We then conclude this section by describing the challenges faced in building an algorithmic method for design and the technical background needed to understand our method.

1.2 Vertical Gardens

We tackle the problem of designing a set of vertical gardens composed of five platforms that are placed in 3D space. Depending on the relative positions of the platforms with respect to each other, the entire structure achieves different attributes. Two such attributes are the rain and sun occlusion for example. Further, the position of the platforms with respect to each other dictates the value of the said attributes and whether the entire structure is constructable or not. We make the distinction between *attributes* and *constraints* in that the *attributes* have to be approximated and don't need to be exact while the *constraints* are hard conditions that have to be met exactly, or else the structure would be infeasible. An architect would like to input some desired values to the system and receive a set of constructions as output that fulfill the input values. For example, the architect might start the design process by inputting a total area and surface to the system. This is the total area and surface of all platforms together. The algorithm then has to generate a set of five platforms that approximate the inputted value. In the next step, the architect might want to set a desired rain and sun occlusion for the construction. The algorithm then has to move the previously generated platforms around in such a way that the desired attributes are approximately fulfilled.

1.2.1 Challenges in Design from a Human's Perspective

An architect is faced with a bevy of challenges while tackling a design problem. For one, there are geographical and material constraints that vastly limit the freedom of the designer. Furthermore, one is met with buildability constraints that need to be satisfied and that are specific to the design concept at hand. All of these values need to be calculated and the design needs to be readjusted to that. This can be very time consuming and mundane, so much so that the original creative drive of the designer might be lost and the aesthetics that were the main motivation for the building being severely altered. This begs the implementation of a model that can automate the time consuming processes and calculations while simultaneously allowing the designer creative freedom. Hence we propose the development of a human-in-the-loop AI that can fulfill those needs.

1.2.2 Challenges in Design from an algorithmic Perspective

Devising an algorithmic solution to a design problem entails its own set of unique challenges. One can categorize these challenges into two groups. One being creative challenges that deal with the question of how does one have an algorithm that is able to design something creative and beautiful. The other challenge being how to have a model that is able to approximate

desired values while being interpretable and safe. While the first challenge is of great interest, we only talk about the latter since it is of more importance for our human-in-the-loop method.
With the rise of machine learning methods and big data, an often taken approach to a challenge is to frame it as an optimization problem. Given a large dataset and an optimizer, gradient based methods are then usually used to minimise or maximize a certain objective. For one, in our task we face the problem of having a large enough dataset for such method and the other one, being that our results are obtained from a black box software. Hence using gradient based methods are not feasible. Given a black-box that returns noisy values for given inputs, we can learn to minimize a certain objective of the black-box by using *Bayesian Optimization* techniques which we will review in the next section.

1.3 Theoretical Background

1.3.1 Bayesian Optimization

Many challenges in machine learning are approached as sequential black-box optimization problems. Some examples of such problems include neural architecture search [11, 24], tuning the hyperparameters of deep learning systems [21] and more recently, applications such as food safety control [2, 17]. Bayesian optimization [16] has been one of the most popular and successful approaches to black-box optimization. Some of the features of Bayesian optimization are its sample efficiency, which is important for expensive to evaluate functions and its tolerance to noisy observations, which is present if the objective function of interest is highly non-linear. A surrogate model is used to represent the input output behaviour of a black-box. Given some initial samples, one fits a surrogate models to these. An acquisition function then guides the exploration for collecting more samples such that one arrives at a good solution without sampling many points.

1.3.2 Setting

The following is adapted from [12]. The Bayesian optimization framework consists of two main components. The first is a Bayesian statistical model of the black-box objective function (often a Gaussian Process), and the other is an acquisition function that tells the algorithm where to make new observations. Some initial samples of the objective are usually given or can be simply obtained by sampling uniformly at random to build an initial estimation of the objective. Once the statistical model is fitted to the initial samples, the acquisition function estimates the most informative point for making new observations and new observations are made until the querying budget is depleted. After each new observation, the model posterior is

Algorithm 1: Basic Bayesian Optimization

Result: Return the best point found
$\mathcal{X}$ the design space;
f the black-box objective;
place a GP prior of f;
fit f to the initial samples;
N the total budget;
a the acquisition function;
for $i < N$ **do**
    update the posterior of f;
    $x_i = max_{x \in \mathcal{X}} a(x)$;
    observe $y_i = f(x_i)$;
    update the posterior of f;
    $i = i + 1$
end

updated. Algorithm **??** illustrates this basic framework. We will give a more detailed overview of Gaussian Processes and different acquisition functions in the proceeding sections.

1.3.3 Gaussian Process

The following is adapted from [12]. We will give a very short overview of Gaussian Processes. A more complete discussion can be found in [19]. Gaussian processes are a powerful method for modeling functions from observations while incorporating prior knowledge. The prior used in GP regression is the multivariate normal with a mean vector and a covariance matrix. The mean vector is constructed by evaluating a mean function at each point and the covariance is constructed by choosing a *kernel* and evaluating it at each pair of points. Assume that we have a set of points $\{x_i\}_{i=1}^n$ for which we make observations $\{f(x_i)\}_{i=1}^n$. Picking some mean function μ_0 and a kernel $\mathbf{K}$, the resulting prior distribution on $f(x_{1:n})$ becomes

$$f(x_{1:n}) \sim \mathcal{N}(\mu_0(x_{1:n}), \mathbf{K}(x_{1:n}, x_{1:n})) \tag{1.1}$$

Using bayes rule, we can calculate the posterior for a new point $\tilde{x}$

$$f(\tilde{x})|f(x_{1:n}) \sim \mathcal{N}(\mu_n(\tilde{x}), \sigma_n^2(\tilde{x})) \tag{1.2}$$

$$\mu_n(\tilde{x}) = \mathbf{K}(\tilde{x}, x_{1:n})\mathbf{K}(x_{1:n}, x_{1:n})^{-1}(f(x_{1:n}) - \mu_0(x_{1:n})) + \mu_0(\tilde{x}) \tag{1.3}$$

$$\sigma_n^2(\tilde{x}) = \mathbf{K}(\tilde{x}, \tilde{x}) - \mathbf{K}(\tilde{x}, x_{1:n})\mathbf{K}(x_{1:n}, x_{1:n})^{-1}\mathbf{K}(x_{1:n}, \tilde{x}) \tag{1.4}$$

The above describe how to model a finite set of points. However, we can use the same approach to model over a continuous domain. Using a mean

function μ and a kernel $\mathbf{K}$ we define a probability distribution of the function f such that for a collection of points $x_{1:n}$, the marginal of f at those points is given by the formula 1.1.
A kernel is a way to model similarity between data points. It needs to fulfill the property of being positive semi definite regardless of the input [23]. There are many different choices of kernels depending on the input domain and problem setting. We present two kernels that were also used in the later sections.
The first is the popular and well known RBF kernel which is used extensively in training non-linear SVMs [6], among other applications [10]. For a pair of input points x_i and x_j and a free length scale parameter l, the RBF kernel takes the following form:

$$\mathbf{K}(x_i, x_j) = exp(\frac{-\|x_i - x_j\|^2}{2l^2}) \tag{1.5}$$

Since the lower limit for the values that the kernel can take is zero and for a pair of equal points $x_i = x_j$ the value equals one, the RBF kernel can be interpreted as a measure of similarity.
Kernels can also be defined on discrete input spaces such as graphs [14]. Given a graph $G = (V, E)$ one can first define its graph Laplacian [13] as follows:

$$L_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ -d_i & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{1.6}$$

$$\tag{1.7}$$

Where d_i is the degree of vertex v_i. Given this matrix, one can define the diffusion kernel on the graph G as

$$\mathbf{K}_\beta = \lim_{s \to \infty} (I + \frac{\beta L}{s})^s \tag{1.8}$$

Where I is the identity matrix, β a temperature parameter and $s \in \mathbb{N}$. Such a kernel can be useful when the input domain is a graph and the objects of interests are nodes on this graph. One can model the similarities of different nodes of the graph based on observations by using the graph diffusion kernel.

1.3.4 Acquisition

Having discussed Gaussian Processes and how they can be used with the help of kernels to model the objective functions of interest, the remaining

ingredient is the acquisition function. Acquisition functions guide the exploration in Bayesian optimization and are important to have a sample efficient algorithm that is able to find good results. They try to mitigate the exploration exploitation dilemma [3] by suggesting to make observation at the most informative points. The most common acquisiton function is the expected improvement (EI) function. This function tries to lead the search towards the best possible point f^* by taking into account the magnitude of improvement [4]. The EI function can be defined as:

$$EI_n(x) = \mathbb{E}\left[max(0, f(x) - f^*)\right] \tag{1.9}$$

So the expected improvement algorithm simply takes the next point to evaluate to be the maximum of the above function

$$x_{n+1} = argmax EI_n(x) \tag{1.10}$$

For a more complete derivation and discussion on EI, the interested reader is suggested to read [4, 12].
Another popular approach to deriving acquisition functions is confidence bound based acquisition functions such as the upper confidence bound (UCB) [22]. They cast the Bayesian optimization problem as a multi-armed bandit problem [20] and use the instantaneous regret

$$r(x) = f^* - f(x) \tag{1.11}$$

for the acquisition. The goal is to minimize the total regret over the search process. Using the upper confidence bound selection criterion, they arrive at the acquisition function:

$$GP - UCB(x) = \mu(x) + \sqrt{\nu\tau_t}\sigma(x) \tag{1.12}$$

It can be shown that this method has no regret for appropriately chosen ν and τ. The proof of this is out of the scope for this work. The interested reader is again lead to [4, 22] for a more complete discussion.

1.3.5 Combinatorial Bayesian Optimization

Bayesian optimization is mostly used in the continuous domain where the number of parameters is less than 20. Recently however, there has been a rising interest in adapting Bayesian optimization for discrete [2, 17] or even mixed [7] domains. We review the recent work of [17] in this section, that takes advantage of the graph signal [18] to model the combinatorial search space.
In COMBO [17], the authors define the search space as the graph cartesian product of multiple smaller graphs. More concretely, for each variable x_i a

small graph G_i is defined where each node corresponds to a state that the variable can take. For ordinal variables, G_i is a path graph and for categorical variables, a complete graph is used. The graph cartesian product of the small graphs then has one vertex for each joint assignment of the discrete variables and thus defines the full search space. Now an appropriate kernel has to be used in order to model the similarities between assignments and enable the use of a Gaussian Process for the algorithm. The signal used in COMBO is the graph fourier signal which is the spectral decomposition of its Laplacian. With this, each graph signal can be represented as a combination of the eigenvalues of the Laplacian and allows for the ARD diffusion kernel [14] to be used. Due to the definition of the combinatorial graph through the graph cartesian product, the eigensystem does not have to be computed for the full graph, which would be intractable, rather it is computed for each individual small graph G_i. The diffusion kernel is then defined as

$$\mathbf{K} = \bigotimes_{i=1}^{m} exp(-\beta_i L(\mathcal{G}_i)) \tag{1.13}$$

where $L(\mathcal{G}_i)$ is the laplacian for graph G_i and β is a sparsifying horseshoe prior [5]. The interested reader is referred to the original paper for more details [17]. We present the COMBO algorithm in algorithm 2 for completeness.

Algorithm 2: COMBO

Result: Return the best point found

Input: N combinatorial variables $\{C_i\}_{1,\dots,N}$;

Compute a sub-graph $\mathcal{G}(C_i)$ for each variable C_i;

Compute eigensystem $\{(\lambda_k^{(i)}, u_k^{(i)})\}_{i,k}$ for each sub-graph $\mathcal{G}(C_i)$;

fit f to the initial samples;

N the total budget;

a the acquisition function;

for *evaluation budget* **do**

fit GP using ARD diffusion kernel to $\mathcal{D}$;

Maximize acquisition function:
$v_{next} = argmax_{v_* \in \mathcal{V}} a(\mu(v_*|\mathcal{D}), \sigma^2(v_*|\mathcal{D}))$;

Evaluate $f(v_{next})$ and append $(v_{next}, f(v_{next}))$ to $\mathcal{D}$;

end

Chapter 2

Problem Statement and Dataset

In this section we formalize the problem setting more and describe the available data sources, how they were generated and the resources used to tackle the challenge.

2.1 Data Description

2.1.1 Data Collection

The data was collected through a combination of manual design combined with the use of the *Grasshopper* [8] software for collecting attribute values. The architects provided a library of platforms, which could then be used to build different 5 platform configurations. Grasshopper was then used to query the different attributes for each configuration of platforms.

2.1.2 Grasshopper

Grasshopper [8] is a software used for parametric design. Given a specification of a design, it can return different attributes. We treat this software as a black-box in our project by defining the objective that we are interested in, in terms of outputs returned by Grasshopper. We try to optimize the parameters such that the desired outputs are approximately reached.

2.1.3 Data Format

The platforms can be described either using polar coordinates or cartesian coordinates. Each platform is a collection of points and a spline interpolation is used to connect the points. An example of a configuration of five platforms with randomly placed support poles can be seen in figure 2.1.

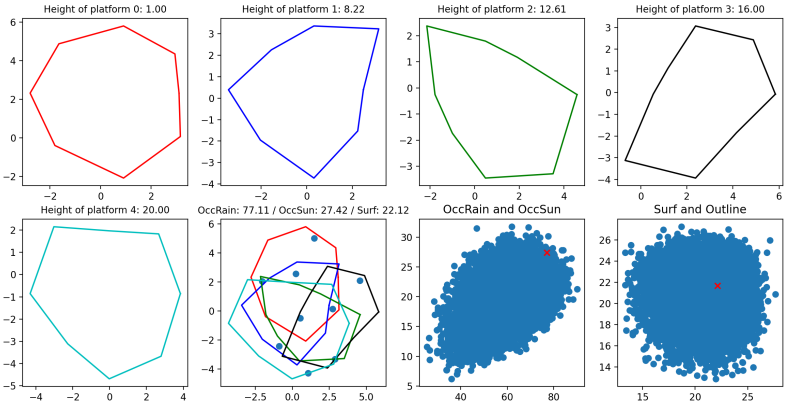


Figure 2.1: An example of five platforms with randomly placed poles.

Cartesian format: we use 8 points per platform initially. Since there are 5 platforms in total and we're using (x, y, z) coordinates, we reach $8 * 5 * 3 = 120$ dimensions using this representation.
Polar format: We can use the polar representation of the geometries by considering 8 radii per platform and the center of each platform in terms of its cartesian coordinate. This leads us to $(8 + 3) * 5 = 55$ dimensions initially. By fixing the height of the first and the last platform, we can further reduce this representation to 53 dimensions.
Constellation based format: The last format used in this project is centered around the constellations. The advantage of this format is that some of the buildability constraints mainly concerned with placing the supports are satisfied a priori. This model assumes first a grid of 11 supports. Each platform can either use one of these supports or not. The data format hence consists of a list of 11 radii per platform, where each entry describes the radius of the platform around each support. For the unused supports, the corresponding entry is simply set to zero. Using this format, each row consists of 55 radii describing the 5 platforms, plus 3 heights for the middle three platforms, resulting in a total of 58 dimensions.

2.1.4 Attributes

A complete list of all the attributes can be found in the source code [9]. We are particularly interested in the sun and rain occlusion attributes and the surface areas and outline lengths. Our goal is to generate a set of platforms given the outline and surface area and then configure them such that the vertical garden approximates the sun and rain occlusion.

2.2 Problem Statement

We now give an overview of the problem statement. Given a vertical garden design problem which requires at most 11 supports given in a predefined grid, the architect gives as an input a total desired area and total desired outline, along with the desired sun and rain occlusion. Given those values, the process should first output 5 platforms that meet the total area and outline and then place them such that the structure is buildable and approximates the desired values. We tackle this problem using both the cartesian format and the SDF format. We will study and elaborate on each approaches advantages and drawbacks in the next chapter.

2.3 Resources

Most of the implementation in this work is using the GPyOpt [1] framework developed by SheffieldML. Also the code from [17] is used and adapted to implement the discrete search. Further scripts for querying Grasshopper and helper functions were provided by Dr.Luis Salamanca and Prof.Matthias Kohler's group at ETH Zurich.

Chapter 3

Methodology

In this section we explain how we tackled the problem. We describe our pipeline step by step. The first step is for the user to input a desired total area and perimeter to the system. We will then use a generative process that will produce 5 platforms that closely match these values. Afterwards the user will input the desired attributes, i.e the desired sun and rain occlusion. The system will then place the generated platforms in such a way as to closely approximate these values while meeting the constraints. For this we describe our two approaches of tackling the problem in the continuous space and the discrete space. We observe that in the continuous space, meeting the constraint requirements becomes intractable for certain points because of the combinatorial nature of the problem and the sparseness of the feasibility. To overcome this, we model the problem in the discrete space where the constraints are met a priori and leverage the COMBO algorithm to optimize our combinatorial black-box.

3.1 Generating the Platforms

The first step in the process of designing a vertical garden is generating the platforms that construct it. We chose 5 platforms for our design but the methodology will also apply to other choices. The user will input to the system a desired total perimeter and total area. We will use the given dataset to simply draw 5 platforms that most closely approach the desired values.

3.1.1 Parametric Approach

We present algorithm 3 as a way to generate platforms. For this, we will use the given database of platforms and store for each platform the ratio between its perimeter squared to the area. We then draw from a Dirichlet a variable s, which describes how the total surface area will be split into

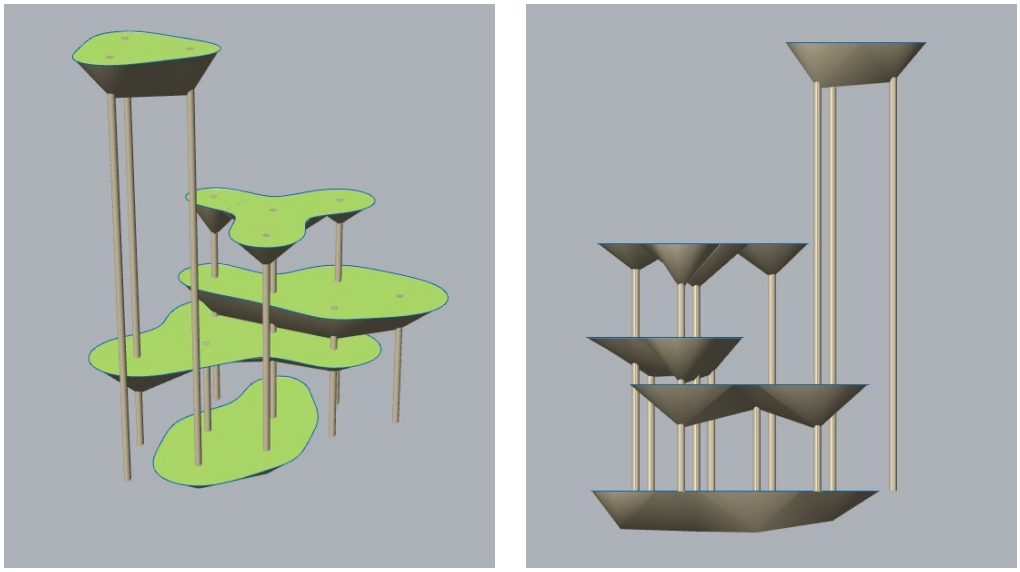


Figure 3.1: Example of generated SDF platforms for area 260.94 and perimeter 123.63. The heights and positions were chosen randomly.

five areas and hence compute the given area to each platform S_i. Then we calculate the residual perimeter and again for each platform, assign a perimeter to it. We then calculate the ration of perimeter squared to the surface area for each computed value and query the closest platforms in the database to that value. We show an example of this for desired area 260.94

Algorithm 3: Platform Generation

Result: 5 Platforms
Describe each platform by $R = \frac{\text{perim}^2}{\text{area}}$;
for *any structure* **do**
  $s = \text{Dirichlet}(10, 10, 10, 10, 10)$;
  $S_i = TS * s_i$;
  $D_{TP} = TP - \sum 2\sqrt{\pi * S_i}$;
  $p = \text{Dirichlet}(10, 10, 10, 10, 10)$;
  $P_i = 2\sqrt{\pi * S_i} + p_i * D_{TP}$;
  $R_i = P_i^2 / S_i$;
  find closest platform in database;
end

and desired perimeter 123.63 in figure 3.1. Please note that this method works for the SDF format as well as for the regular format.

A discussion on the accuracy of the results produced by this method can be

found in chapter 4.

3.2 Bayesian Optimization in Vegetale

In our approach to the problem, we treat Grasshopper as a black-box and define an objective function that describes the closeness between the obtained values and the desired attributes. Thus we formulate the problem as a black-box optimization problem, where our black-box function is the aforementioned objective. We aim to minimize this objective using Bayesian optimization.

3.2.1 Human in the Loop A.I

Human-in-the-loop or HITL refers to a model where the human, in this case the architect, is constantly involved in the process. We propose our solution not as a complete replacement of the expert designers, but mainly as a smart tool that automates the mundane and time consuming tasks and allows the architects to focus on the more creative aspects. After the architect has inputted some initial values, the algorithm can provide multiple solutions that approximate the desired attributes (through consecutive runs). The architect then can select one of these and continue the process to further refine or adjust the design, querying the same algorithm in the process multiple times. We would like to further allude to the fact that the same approach of using Bayesian optimization, could be used for other design problems.

3.2.2 Modeling the Objective

We use the mean squared error as our objective. Given a desired rain occlusion r_{true} and desired sun occlusion s_{true} and obtained occlusion values r, s we define:

$$\mathcal{L}(r,s) = \frac{1}{2}\sqrt{(r_{\text{true}} - r)^2 + (s_{\text{true}} - s)^2} \tag{3.1}$$

We further normalize the values in the interval $[0,1]$ in order to circumvent numerical instabilities due to scale.

3.2.3 Constraint Specification

Our constraints are concerned with supporting the vertical garden through support poles. We are given a list of constraints specified by the architects that determine if a solution is good or not. These constraints are derived such that the physical restrictions of building such a structure are met. We summarize these constraints in the following list.

- Each platform requires to have between 3 to 5 support poles.

- All poles need to have a minimum distance of at least 2 meters between them.
- If a pole falls inside a platform, it needs at least 80 centimeters of separation from the edge of the platform.
- If a pole falls outside of a platform, it needs at least 80 centimeters of separation from the edges of the other platforms.

3.2.4 Search Space

We explore a few approaches of modelling the search space. The main driver, is the satisfiability of our constraints. More precisely, we test a continuous representation of the search space, where the constraints are not satisfied a priori, and a discrete space, where the space is a set of configurations that a priori satisfy the constraints. We see that with careful description of the discrete space, we are able to efficiently set up the problem and arrive at a solution that approximates the attributes and satisfies the constraints.

Continuous

The continuous space is defined on three axis. The objective is to find positions (x, y, z) for each platform such that the attributes are met. We can reduce the number of parameters, by fixing the height of the first and last platform and further fixing the first platform to $(0, 0, 4.5)$ and thus we arrive at 11 parameters. The search space is defined for $x, y \in [-6, 6]$ and $z \in [4.5, 19]$ meters and the RBF kernel is used to model similarity between points. After having either found a good approximation to the desired attributes, or having exhausted the evaluation budget, we attempt at positioning support poles such that they meet the requirements of section 3.2.3. Unfortunately, we found out that although it is possible to find good approximations for the attributes in a few iterations, this usually needs multiple random initializations and only works well for certain values as discussed in section 4.2. Furthermore, finding constraint satisfying positions for the poles is not realistic. The main reason for this shortcoming is the vast search space that is present. Some approaches that were taken to alleviate this problem were:

- Fixing a grid of supports that already satisfy the mutual distance condition between the poles and finding a suitable position for the entire grid through shifts and rotations.
- Reformulating the constraint satisfaction problem as a Bayesian optimization problem, using a method to translate constraints to a continuous loss function [15].
- Computing the overlapping areas in the platforms through geometric algorithms and defining the legal areas for poles to be placed then

using different search methods in the legal areas such as Bayesian optimization.

Although if successful, the above methods had the added benefit of coming up with random looking positions for the poles, which can be aesthetically pleasing, unfortunately, none of the above methods were able to converge to a satisfying solution in the general case.

Discrete

Facing intractability issues for the continuous approach of the task, we use the SDF representation of the data to transform the problem to the discrete domain. For this, each platform is represented as an array of 11 radii. The assumption in this data format is that there exists a given grid of 11 supports and for each entry in the array, they describe whether that support is used for that particular platform. Each value describes the radius that platform has around that support and if it is zero, then the support is not used. A sine distance function is then used to interpolate between the circles and create the platforms, however, this is out of the scope of our discussion. Using this format, we can make sure that the constraints are satisfied a priori. We create a dataset of all possible positions of the platforms, including flips and rotations and transform the problem into a discrete choosing problem, where you pick a joint configuration of 5 platforms such that they match the desired attributes. To this end we use the COMBO [17] algorithm.

3.2.5 COMBO

In order to use COMBO, we will create a categorical variable for each platform. The number of categories that each variable has will depend on the possible number of positions that are pre-computed once the platforms are generated. Having calculated these, we define a complete graph for each individual variable where each vertex of the graph represents one of the possible positions that the current platform can be placed in. Having defined the 5 individual graphs, we call the COMBO algorithm to optimize our objective. A full discussion of the accuracy and results obtained by both the COMBO algorithm and the continuous approach can be found in section 4.2.

Chapter 4

Results

4.1 Generating Platforms

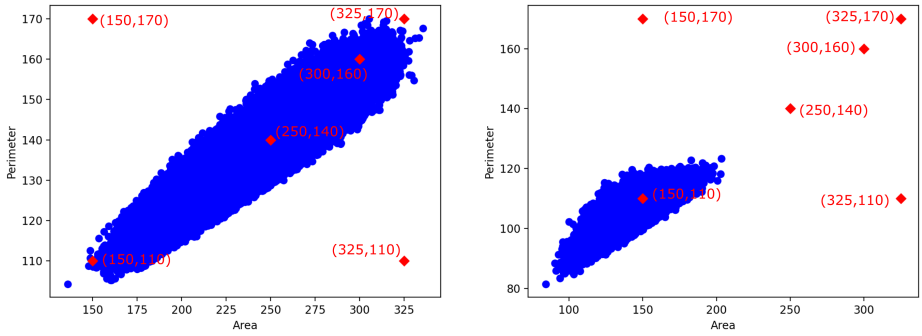


Figure 4.1: Distribution of the total surface and area inside the dataset. The annotated points were chosen for the experiments. The left figure illustrates the SDF distribution and the right figure the XYZ.

We report the accuracy of the generation process presented in algorithm 3. We pick 6 test points such that we cover a wide range of values as illustrated in figure 4.1. As you can see, we have points that are typical for our use case and also some edge cases and out of distribution points. We perform two experiments, in the first one called *Ratio*, the process is identical to algorithm 3 and in the *Surface Only* experiment, we describe each platform only by its surface area and do not take the ratio of $\frac{\text{perimeter}^2}{area}$. We only perform one run per test point since the algorithm always returns the same results. The results are presented in table 4.1. Both experiments are performed using the SDF data format. We observe that in general, using the ratio $\frac{\text{perimeter}^2}{area}$ results in a lower accuracy especially for boundary and out of distribution

values. Only using the surface as a residual parameter gives better results. We believe the reason for that to be that there are not diverse enough values present in the database to give a wide range of ratios, therefore, many different desired attributes will result in the same platform. Using only the surface as a residual parameter, one can overcome this issue.

		Ratio		Surface Only	
Desired Attributes		Abs. Error		Abs. Error	
Area	Perimeter	Area	Perimeter	Area	Perimeter
150	110	100.1	35.24	8.05	2.64
150	170	92.57	18.10	8.05	3.35
250	140	6.13	2.54	1.02	2.61
300	160	56.06	19.80	9.62	7.56
325	110	83.84	20.60	3.50	4.47
325	170	32.95	6.45	3.50	1.52

Table 4.1: Absolute error of the generation process using the SDF data format. The values are picked both from inside and the boundary of the distributions.

For completeness we also report the results of the generation algorithm for the XYZ format in table 4.2. As evident in figure 4.1, the distribution is less varied for this data format. Hence, we have ran the experiments for one point which falls inside the distribution and one point outside of the distribution. The point outside of the distribution has a high absolute deviation for both methods, while the point inside achieves better results. From this we can conclude that the performance of the generation algorithm highly depends on the distribution of the available data set and whether the desired points are well represented inside of it.

		Ratio		Surface Only	
Desired Attributes		Abs. Error		Abs. Error	
Area	Perimeter	Area	Perimeter	Area	Perimeter
150	110	72.6	11.3	40.0	13.4
250	140	170.8	11.4	110.0	41.0

Table 4.2: Absolute error of the generation process using the XYZ data format. The values are picked both from inside and the boundary of the distributions. Only two points are chosen since the distribution is less varied in this format.

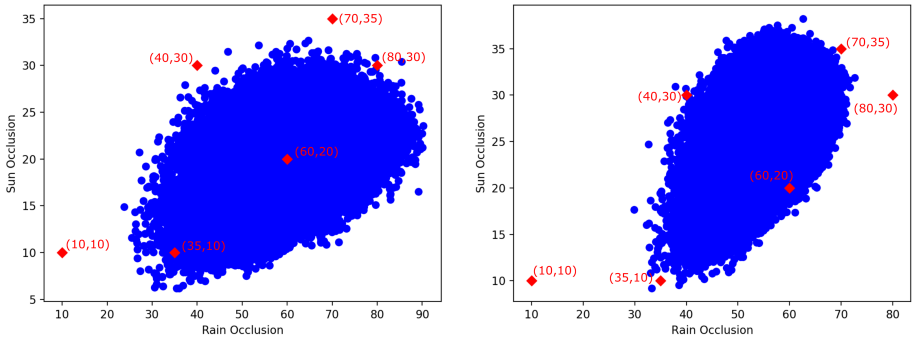


Figure 4.2: Distribution of the rain and sun occlusion inside the dataset. The left figure indicates the XYZ dataset and the right the SDF.

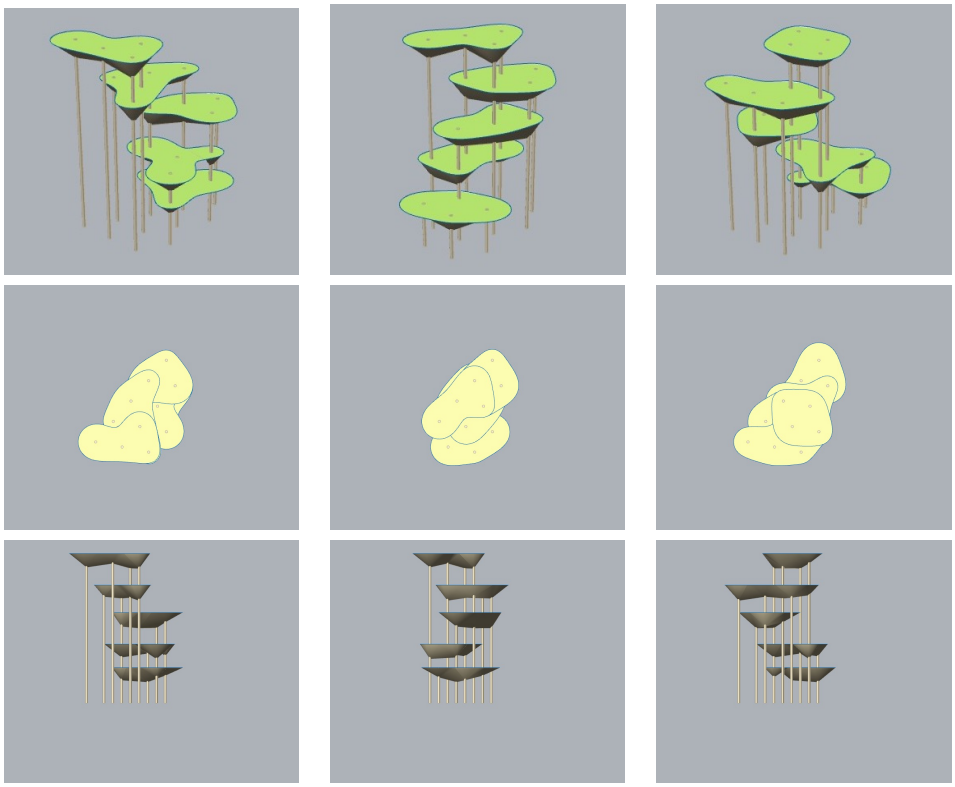
4.2 Finding the Attributes

We now report the accuracy of the different Bayesian optimization methods for approximating the attributes. For that, we draw a set of five platforms with the desired area and the desired perimeter at random. We configure the COMBO algorithm with an $n\_eval = 100$ and 25 initialization samples. The SDF data format is used for COMBO, while the continuous XYZ format had to be used for the continuous Bayesian optimization approach. Note that we are not concerned with meeting the constraints in the continous case and only consider approximating the attributes. The tolerance of the continuous Bayesian optimization is set to 0.009 with a maximum number of iterations of 100. We picked six test points from inside and the boundaries of the typical distribution of values along with some out of distribution points as you can see in figure 4.2. We present the average absolute error of 5 total runs for both methods in table 4.3.

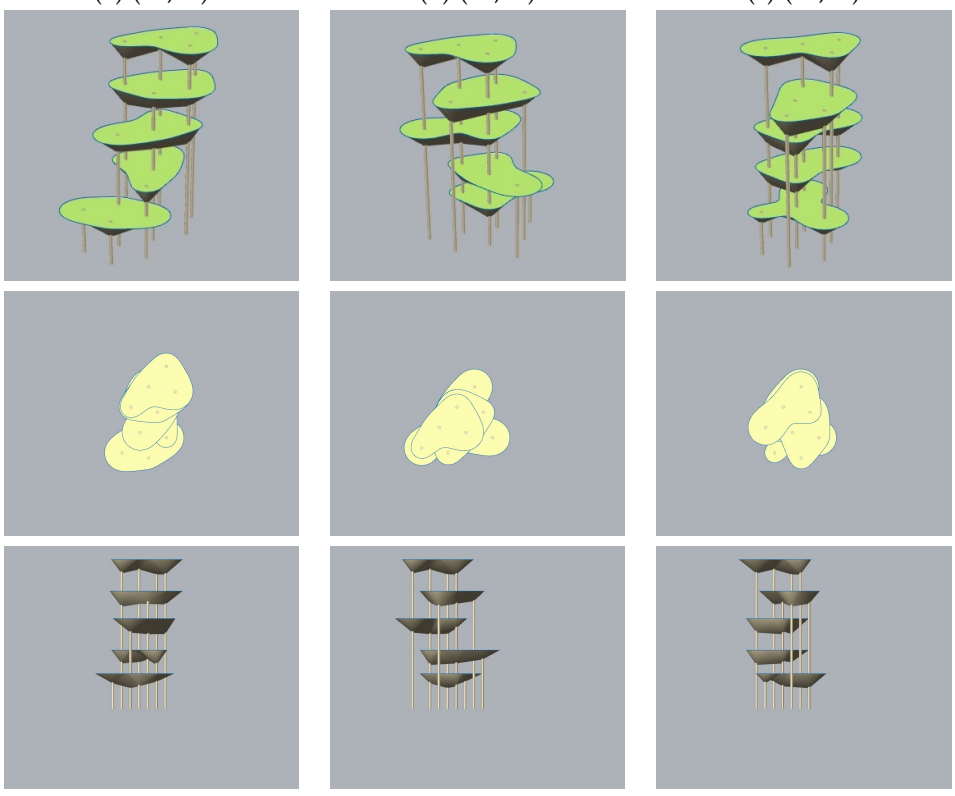
		COMBO		Continous BayOpt	
Desired Occlusion		Avg. Abs. Deviation		Avg. Abs. Deviation	
Rain Occ.	Sun Occ.	Rain Occ.	Sun Occ.	Rain Occ.	Sun Occ.
10	10	41.44 (±1.8)	20.13 (±2.6)	0.76 (±7.7)	0.63 (±7.8)
35	10	21.15 (±5.5)	12.65 (±4.2)	17.47 (±5.8)	5.0 (±1.7)
40	30	13.91 (±3.7)	2.25 (±2.1)	22.98 (±7.8)	17.19 (±6.0)
60	20	6.46 (±5.0)	3.58 (±2.8)	43.86 (±4.7)	14.68 (±1.5)
70	35	15.35 (±1.8)	7.19 (±1.6)	55.85 (±8.0)	27.94 (±4.0)
80	30	22.90 (±6.0)	9.31 (±2.5)	63.41 (±6.3)	23.82 (±2.3)

Table 4.3: Average absolute error of both algorithms for an average of 5 runs.

The continuous approach is in general less robust and highly likely to get stuck in local optima, as the results often don't improve after a few iterations and require multiple initializations. COMBO on the other hand, is able to find acceptable approximations for points around and inside the typical distribution. A full run of COMBO on the problem can take up to 6 minutes on average, while the continuous approach is dependent on the number of total iterations and the tolerance and also the initialization. Rendered images from three different perspectives of the final vertical gardens can be found in figure 4.3 on the next page. As one would expect, lower rain occlusion can result in the platforms being more spread out on the grid as evident from sub-figure 4.3a and 4.3b, while very high rain occlusion results in the platforms being stacked almost directly on top of each other as in sub-figure 4.3d and 4.3e.

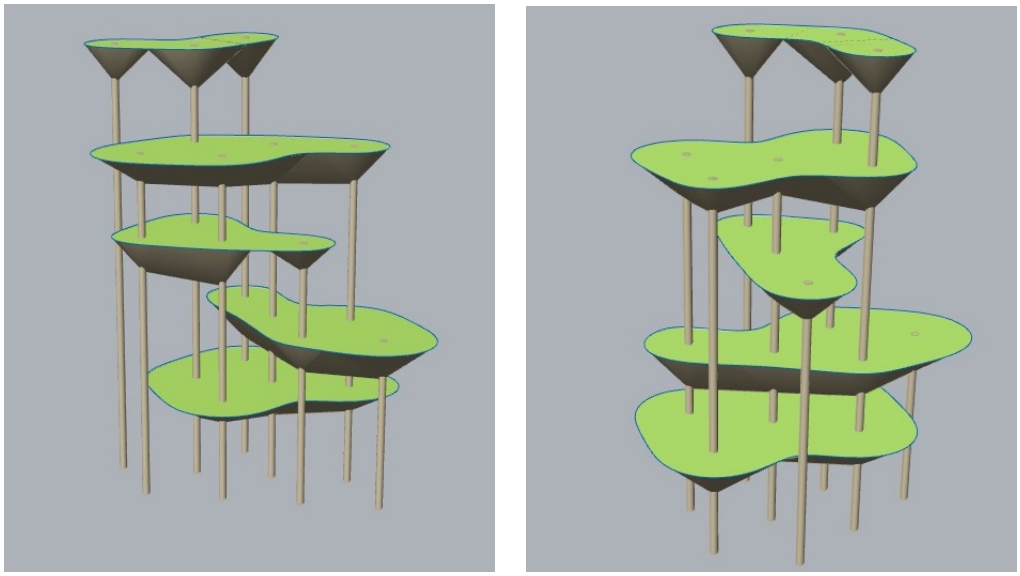


(a) (10,10) (b) (35,10) (c) (40,30)



(d) (60,20) (e) (70,35) (f) (80,30)

Figure 4.3: Example of the configurations found by COMBO. The desired attributes are specified in the sub-captions as (rain occ, sun occ).



(a) Error (8.78,3.15) (b) Error (3.97,7.2)

Figure 4.4: Example of how one can find diverse solutions for the same attributes. The absolute errors of the (rain occ, sun occ) are presented in the sub-captions in that order.

A nice feature of our solution method is that one can take advantage of the high multi-modality of the problem to present a set of diverse solutions that all approximate the same attributes to the client. As an example consult figure 4.4, where both configurations are approximating a rain occlusion of 60 and sun occlusion of 20. Sub-figure 4.4a approximates the rain occlusion with an error of 8.78 and the sun occlusion with an error of 3.15, while sub-figure 4.4b reaches the deviations 3.15 and 7.2 respectively. This fits well within out human-in-the-loop framework, as the architect can then decide which configuration to continue with.

Chapter 5

Conclusion and Future Work

In this work we have explored the field of automated design of architectural structures and have tackled the challenge of designing vertical gardens using machine learning. We have proposed a full end-to-end pipeline that given a set of inputs, first generates a set of suitable platforms and then places them optimally such that they approximate the desired attributes. We have seen some of the main challenges associated with these problems and how one can circumvent these issues with Bayesian optimization. In a real world scenario some problems offer many more infeasible solution points than feasible ones, such that finding a solution that strictly satisfies all of the constraints is intractable, let alone finding the most optimal one. Realizing this we have proposed a way on how one can mitigate this issue by discretizing the search space into choices of constraint satisfying points and how the COMBO algorithm can be employed to find a good solution efficiently. Furthermore we would like to allude to the fact that due to the online nature of our approach and the similarities it has to a human designer, it can be used for collecting a large dataset and is beneficial to use in conjuncture with deep learning based methods.
In the future, we would like to explore how a similar pipeline might be applied to different architecture problems outside of designing vertical gardens. Furthermore, we would like to explore if one can come up with a more general AI based human-in-the-loop method, that allows for architects to freely design structures while not having to worry about time consuming and mundane attributes and constraints.

5.1 Acknowledgement

In conclusion I would like to thank Dr. Fernando Perez Cruz and Dr. Luis Salamanca for allowing me to contribute to this project and for the countless hours of help, guidance and interesting discussions provided by them.

Bibliography

[1] The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

[2] Ricardo Baptista and Matthias Poloczek. Bayesian optimization of combinatorial structures, 2018.

[3] Oded Berger-Tal, Jonathan Nathan, Ehud Meron, and David Saltz. The exploration-exploitation dilemma: A multidisciplinary framework. *PLOS ONE*, 9(4):1–8, 04 2014.

[4] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010.

[5] Carlos M. Carvalho, Nicholas G. Polson, and James G. Scott. Handling sparsity via the horseshoe. volume 5 of *Proceedings of Machine Learning Research*, pages 73–80, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[6] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(48):1471–1490, 2010.

[7] Erik Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. Mixed-variable bayesian optimization. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[8] The Grasshopper3d developers. Grasshopper: algorithmic modeling for rhino. https://www.grasshopper3d.com.

[9] Pouya Pourjafar Dr. Luis Salamanca. Vegetale source code. https://renkulab.io/gitlab/luis.salamanca/vegetale_bayopt, 2020.

[10] Michael Eigensatz and Mark Pauly. Insights into the geometry of the gaussian kernel and an application in geometric modeling. 2006.

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey, 2019.

[12] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[13] William N. Anderson Jr. and Thomas D. Morley. Eigenvalues of the laplacian of a graph. *Linear and Multilinear Algebra*, 18(2):141–145, 1985.

[14] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *in: Proceedings of the 19th International Conference on Machine Learning*, pages 315–322. Morgan Kaufmann, 2002.

[15] Dana Drachsler-Cohen Timon Gehr Ce Zhang Martin Vechev Marc Fischer, Mislav Balunovic. Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*, 2019.

[16] J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pages 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[17] Changyong Oh, Jakub M. Tomczak, Efstratios Gavves, and Max Welling. Combinatorial bayesian optimization using the graph cartesian product, 2019.

[18] Antonio Ortega, Pascal Frossard, Jelena Kovacevic, Jose M.F. Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, May 2018.

[19] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

[20] Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

[21] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th*

International Conference on Neural Information Processing Systems - Volume 2, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[22] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, May 2012.

[23] JP. Vert, K. Tsuda, and B. Schölkopf. *A Primer on Kernel Methods*, pages 35–70. MIT Press, Cambridge, MA, USA, 2004.

[24] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search, 2019.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Vegetale: Automated Human-In-The-Loop Design of Vertical Gardens Through Black-Box Optimization Methods

Authored by (in block letters):

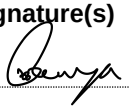
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Pourjafar Kolaei	Pouya

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 16.11.2020	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.