2. Wind Turbine Placement Optimization

Intention:
Explain the Optimization Problem.

In this chapter, the concept of the turbine placement optimization problem is introduced. The basic problem is easy to describe: Multiple turbines i with the location $\mathbf{t}_i$ for $i = 1, \ldots, N_t$ should be placed on a map with the size $m = m_x \cdot m_y$. It is not about finding areas where wind turbines may be placed. The objective is to determine locations of multiple turbines in an area. Thereby, one type of wind turbine is employed. The locations of the turbines are described in a solution $\mathbf{x}$ and this solution can be rated by an objective function $f(\mathbf{x})$. Depending on the objective function, it is a maximization or minimization problem, e.g., the maximization of the power output or the minimization of the costs. As in this thesis, the power output should be maximized; the task is to find the optimal solution $\mathbf{x}^*$ for which it applies $f(\mathbf{x}^*) > f(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{X}$, with solution space $\mathcal{X}$ in which all solutions are located. The wind model employed to compute the objective function f is introduced in Chapter 3. Besides the objective function f, constraints play an

D. Lückehe, *Evolutionary Wind Turbine Placement Optimization with Geographical Constraints*, DOI 10.1007/978-3-658-18465-0_2

important role for the optimization problem. A solution $\mathbf{x}$ that can be realized in real-world terms has to fulfill all constraints. In the turbine placement optimization problem, there are multiple constraints that have to be taken into account, e.g., minimum distances to objects on the map and between the turbines. Especially in this field, the existing approaches do not use acceptable simplifications – see Section 1.2. In this thesis, real-world data from a map service are used to model constraints on the map. The constraints are proposed in Chapter 4.

This chapter is structured as follows. First, optimization problems in general are introduced. Then, different techniques to solve various types of optimization problems are shown. In the next step, a deeper look is taken at heuristic black-box optimization, including the introduction of popular approaches to solve heuristic black-box problems and an explanation of the meaning of solution spaces. It is followed by a section on the representation of solutions. In this section, a binary and a real-valued representation are proposed. At the end of this chapter concluding remarks are made.

2.1. Optimization Problem

The structure of an optimization problem is shown in Figure 2.1. The figure is based on [Eiben and Smith, 2007]. In an optimization problem, the input vector $\mathbf{x}$ is missing and the task is to find an input vector $\mathbf{x}$ that matches an objective in the context of model f and the output value y. If model f, instead of the input vector $\mathbf{x}$, is missing, the task is a modeling problem and if the output value y is missing, the challenge is a simulation problem. In case of a modeling and simulation problem, the output value y can be an output vector $\mathbf{y}$. As in an optimization problem, the input vector $\mathbf{x}$ is missing; the input vector is called solution $\mathbf{x}$ and the model f is the objective function.

The task in a general optimization problem is to find a solution $\mathbf{x}$ from solution space $\mathcal{X}$ that minimizes or maximizes an objective function f. Depending on the co-domain of solution space $\mathcal{X}$, there are discrete and continuous optimization problems. If the number

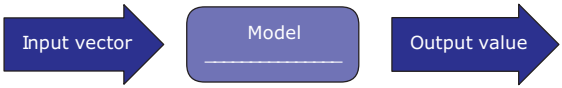


Figure 2.1.: Structure of an optimization problem.

of possible solutions is finite in a discrete optimization problem, it is a combinatorial optimization problem. A famous example for this kind of optimization problems is the traveling salesman problem [Rosenkrantz et al., 1977]. In the theoretical context, a minimization problem is often chosen – this is based on convention. By replacing f through $-f$, it is possible to switch between minimization and maximization problems. Thus, optimization algorithms to minimize can easily be used to maximize an objective function f. As in this thesis, the power output of wind turbines should be maximized, the description of a maximization problem is chosen:

$$\max_{\mathbf{x}} f(\mathbf{x}). \tag{2.1}$$

For the optimal solution $\mathbf{x}^*$ applies $f(\mathbf{x}^*) \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. In real-world optimization problems, there are often a lot of constraints to consider. To take these constraints into account, Equation 2.1 is applied with the subject to:

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, n_1 \tag{2.2}$$

$$h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, n_2 \tag{2.3}$$

where $g_i(\mathbf{x})$ represents the constraint described by inequalities and $h_j(\mathbf{x})$ are the equality constraints. The equations $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ can be used to specify a degree of constraint violation $G(\mathbf{x})$ [Coello, 2002]:

$$G(\mathbf{x}) = \sum_{i=1}^{n_1} \max(0, g_i(\mathbf{x}))^{\beta_G} + \sum_{j=1}^{n_2} |h_j(\mathbf{x})|^{\gamma_G} \tag{2.4}$$

where β_G and γ_G are parameters to weigh the constraints. During the optimization process, a solution $\mathbf{x}_1$ with $G(\mathbf{x}_1) \neq 0$ can help to

lead the process into a worthwhile direction. But at the end of the optimization process, a solution $\mathbf{x}_2$ with a high objective function value $f(\mathbf{x}_2)$ is required and the solution $\mathbf{x}_2$ has to fulfill $G(\mathbf{x}_2) = 0$.

2.2. Solving Optimization Problems

There are a lot of techniques to solve optimization problems. An overview can be found in [Boyd and Vandenberghe, 2004], [Maringer, 2005], [Rothlauf, 2011], [Avriel, 2003], and [Luenberger and Ye, 2008]. Determining which method is the best choice depends on the characteristics of the optimization problem, e.g., it is not necessary to use a highly advanced method to solve a simple optimization problem but, on the other hand, a very complex problem is not solvable with a simple approach.

An overview on the most important approaches is shown in the following. When the task is to compute an optimization problem consisting of linear functions, it is related to the field of linear programming. It can be solved by using the Simplex Algorithm [Dantzig, 1990]. The Simplex Algorithm transforms inequations into equations by applying slack variables [Boyd and Vandenberghe, 2004] and then includes and excludes base variables until the optimum is found [Rothlauf, 2011]. When the optimization problem includes quadratic functions, the problem lies in the field of concave programming. Solving convex optimization problems is more computational complex than linear optimization problems but there are approaches to solve them [Hiriart-Urruty and Lemarechal, 1996]. Optimization problems that handle non-convex functions are related to the field of nonlinear programming. When handling nonlinear optimization problems, "there are no effective methods for solving the general nonlinear programming problem" [Boyd and Vandenberghe, 2004]. Thus, to solve a nonlinear optimization problem, specially adapted algorithms are needed to create optimal solutions in an effective way. For these algorithms often applies: "The computational algorithms of nonlinear programming are typically iterative in nature, often

characterized as search algorithms" [Luenberger and Ye, 2008]. An example are Newtons method based search algorithms. Applied to a one-dimensional problem, the next solution x_{i+1} in the iteration process is computed as presented in Equation 2.5 by the Newtons method [Hiriart-Urruty and Lemarechal, 1996]:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.5}$$

Equation 2.5 shows that the new solution is based on the current solution x_i and the derivation of the objective function $f'(x)$ is needed to compute the new solution x_{i+1}. For highly complex and multi-dimensional problems, it is possible that the determination of the derivation can not be computable or can cause very high computational costs.

In a real-world context, "many practical problems are NP-complete" [Rothlauf, 2011]. The class NP-complete describes the complexity of a problem. An NP-complete problem is assigned to the class of NP problems and NP-hard problems. Solutions of problems that lie in the complexity class NP can be verified in polynomial time. If a problem can be solved in polynomial time it is assigned to the P complexity class. The question if P is equal to NP is an unsolved question in theoretical computer science [Fortnow, 2009]. If P is equal to NP then NP-complete problems can be solved in polynomial time. But as "most scientists conjecture that NP $\neq$ P" [Arora and Barak, 2009] and NP-complete problems are also NP-hard, they cannot be solved in polynomial time. The definition of NP-hard can be found in [Arora and Barak, 2009]. To solve an optimization problem that is not solvable in polynomial time, heuristic optimization approaches are a good choice [Rothlauf, 2011]. As the verification of a solution $\mathbf{x}$ can be done in polynomial time, heuristic optimization approaches treat the objective function $f(\mathbf{x})$ as a black-box to evaluate created solutions. These solutions are iteratively changed by specialized operators to generate optimal solutions.

2.3. Heuristic Black-Box Optimization

The word heuristic comes from the Ancient Greek word

εὑρίσκω

which stands for find or discover [Abbass, 2001]. It describes the art of finding reasonably good solutions under the constraints of bounded time and limited knowledge [Gigerenzer et al., 1999]. According to the origin of the Ancient Greek word, heuristic optimization algorithms discover solution space $\mathcal{X}$ and try to use the knowledge from the previous observations to direct the discovery process into the right direction. The algorithms perform randomized. They can be split into two groups: construction heuristics and improvement heuristics. Construction heuristics start with an empty solution and terminate when the solution is completed. There are no improvement steps. These algorithms do not usually perform as well as other heuristics but are able to create reasonably good solutions. Typically, they can determine solutions faster than other approaches [Schneider and Kirkpatrick, 2006]. Improvement heuristics start with complete solutions and improve them iteratively [Rothlauf, 2011]. The initial solutions can be created by a random method like a construction heuristic or by the usage of problem specific knowledge. Improvement heuristics create iteratively new solutions and evaluates them [Maringer, 2005]. Thereby, the solutions indicated to direct the optimization process into a worthwhile direction are picked. There are a lot of different applications in engineering where randomized approaches like construction heuristics and improvement heuristics are applied [Marti, 2005].

2.3.1. Popular Approaches

In this section, two popular improvement heuristics are introduced: Evolutionary Algorithms and Simulated Annealing. Like every improvement heuristic, they start with one or multiple initial solutions and then iteratively create new solutions. The approaches can be primarily distinguished by the number of solutions that are considered

simultaneous, the operators that vary the solutions, and the way the solutions are selected for the next iteration step.

Evolutionary Algorithms

Evolutionary Algorithms (singular: EA, plural: EAs) are very famous in the field of improvement heuristic. An overview of different types and techniques is given in [Neumann and Witt, 2010], [Eiben and Smith, 2007], [Beyer, 2001], [Beyer and Schwefel, 2002] and [Ashlock, 2005]. The approach is inspired by the optimization in nature that is called evolution. It consists of three main operators. For variation of the solutions there are recombination and mutation operators. They ensure the exploration of solution space $\mathcal{X}$. The selection is responsible for choosing solutions and giving the optimization process a direction. EAs were developed independently in Germany by Rechenberg [Rechenberg, 1973] and Schwefel [Schwefel, 1977], and in the United States by Holland [Holland, 1975].

The basic method is shown in Algorithm 1. In the first step, the initial population $\mathcal{P}^0$ is created. Like mentioned in the introduction of improvement heuristics, an initial solution $\mathbf{x}^0$ can be created by a randomized method like a construction heuristics, or by the usage of problem specific knowledge. The initial population $\mathcal{P}^0$ becomes the current population $\mathcal{P}$ and the first generation starts. As preparatory work for the recombination, ρ solutions from $\mathcal{P}$ are selected. The recombination creates a new solution $\mathbf{x}$ by combining the parameters of the parents. There are numerous ways to accomplish this, e.g., for rational numbers, the mean value can be calculated [Bäck, 1996]. In the next step, the new solution $\mathbf{x}$ is mutated. The mutation has to fulfill the properties of reachability and scalability, and should not drift [Beyer, 2001]. For rational numbers, a common choice is the Gaussian mutation where it applies for the one-dimensional case: $x' = x + \sigma \cdot \mathcal{N}(0, 1)$. While $\mathcal{N}$ is the Gaussian distribution and σ is the step size to scale the mutation strength [Eiben and Smith, 2007]. The mutated solution $\mathbf{x}'$ is added to the new population $\mathcal{P}'$. The recombination and mutation are repeated λ times. Thus, λ is

Algorithm 1 General EA.

1: create initial population $\mathcal{P}^0 \leftarrow \{\mathbf{x}_0^0, \mathbf{x}_1^0, \ldots, \mathbf{x}_\mu^0\}$
2: initialize population $\mathcal{P} \leftarrow \mathcal{P}^0$
3: **while** no termination condition **do**
4: empty new population $\mathcal{P}' \leftarrow \{\}$
5: **while** λ times **do**
6: parents $\leftarrow$ select ρ solutions from $\mathcal{P}$
7: $\mathbf{x} \leftarrow$ recombine parents
8: $\mathbf{x}' \leftarrow$ mutate $\mathbf{x}$
9: add $\mathbf{x}'$ to $\mathcal{P}'$
10: **end while**
11: evaluate: compute $f(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{P}'$
12: $\mathcal{P} \leftarrow$ select μ solutions from $\mathcal{P}'$ or $\mathcal{P} \cup \mathcal{P}'$
13: **end while**
14: **return** best $\mathbf{x} \in \mathcal{P}$

the size of the new population $\mathcal{P}'$. Before the algorithm can select solutions for the next generation, it has to evaluate the new solutions. It uses the objective function f which is called the fitness function in the context of EAs [Neumann and Witt, 2010] and computes $f(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{P}'$. In the field of complex real-world optimization problems, the evaluation is usually the most expensive step. If the fitness function f is highly expensive, it can consume more than 99% of the computation time. After the evaluation, the selection chooses μ solutions. There are two main categories for the approaches. In the plus-selection, the solutions can be selected from $\mathcal{P} \cup \mathcal{P}'$ and in the comma-selection, only solutions from $\mathcal{P}'$ can be selected [Kinnear et al., 1999]. There are different approaches to determine which solutions should be selected, ending with very complex methods from the field of population management and beginning with the easiest method that just selects the solutions with the best fitness function

values. This means for a maximization problem, the solutions with the highest fitness function values are chosen [Eiben and Smith, 2007]. In constrained optimization problems, the selection is particularly interesting as there are feasible and infeasible solutions. The selected solutions are the population $\mathcal{P}$ for the next generation. The steps of creating a new population $\mathcal{P}'$ and selecting solutions are called a generation. The production of generations is repeated until a termination condition is fulfilled. For example, this could be a fixed number of generations that is defined before the optimization runs. Also, an adaptive method can be applied that is able to detect that no further improvements are possible.

When talking about an implemented EA, the notation $(\mu + \lambda)$-EA or (μ, λ)-EA is often used [Bäck, 1996]. With this notation, the reader can immediately recognize the significant parameters: the number of created solutions per generation λ, the number of selected solutions μ, and, if a plus- or comma-selection is applied. As already mentioned, the objective function is called the fitness function in the context of EAs and the current iteration is called the generation.

A very famous and state-of-the-art algorithm is the CMA-ES [Hansen, 2006]. The CMA-ES creates new solutions like most other EAs based on the Gaussian distribution. Thereby, the covariance matrix is employed. It is updated by the covariance matrix adaptation and represents the pairwise dependencies between the variables. Wagner *et al.* used the CMA-ES to optimize the locations of wind turbines on an empty map [Wagner et al., 2011]. Their experiments show that the optimization results of the CMA-ES outperform the optimization module of the industry tool OpenWind [AWS Truepower, 2008].

Simulated Annealing

The approach of Simulated Annealing (SA) is based on a cooling down process of metallic elements that reduce defects in crystal structures. It was proposed by [Kirkpatrick et al., 1983] and [Černý, 1985] as an optimization method that is well suited to finding the global

optimum in a solution space $\mathcal{X}$ with multiple local optima. The basic approach employs discrete solution spaces but there are extensions that work with continuous solution spaces [Bertsimas and Tsitsiklis, 1993] like [Jeng and Woods, 1990]. The special idea of the approach of SA is the acceptance of worse solutions depending on a temperature T. The higher temperature T is, the higher the chance is of accepting worse solutions as a new base for the optimization process. Like in the cooling process of metallic elements, the temperature T is very high at the beginning and is reduced during the optimization process.

Algorithm 2 SA Algorithm.

1: $\mathbf{x} \leftarrow \mathbf{x}^0$
2: **while** no termination condition **do**
3: create neighbor $\mathbf{x}'$ from $\mathbf{x}$
4: **if** $f(\mathbf{x}') > f(\mathbf{x})$ **then**
5: $\mathbf{x} \leftarrow \mathbf{x}'$
6: **else if** $\mathcal{U}[0,1] < e^{-\frac{f(\mathbf{x})-f(\mathbf{x}')}{T}}$ **then**
7: $\mathbf{x} \leftarrow \mathbf{x}'$
8: **end if**
9: $T \leftarrow \tau_T \cdot T$
10: **end while**
11: **return** $\mathbf{x}$

SA – shown in Algorithm 2 – works similar to an $(1+1)$-EA. So, in every iteration, one new solution is created and the algorithm selects one solution. The optimization starts with an initial solution $\mathbf{x}^0$ that is assigned to the working solution $\mathbf{x}$. Then, until a termination condition is reached, the optimization algorithm creates a neighbor $\mathbf{x}'$ of the current solution $\mathbf{x}$. The neighboring solution $\mathbf{x}'$ is analyzed w. r. t. objective function f. If the solution $\mathbf{x}'$ improves the objective function value, i.e., in a maximization problem $f(\mathbf{x}') > f(\mathbf{x})$, the

solution $\mathbf{x}'$ replaces $\mathbf{x}$. For a better chance to leave local optima, SA can also accept worse solutions. The probability M that describes the chance to accept a worse solution depends on the difference between the objective function values $f(\mathbf{x})$ and $f(\mathbf{x}')$ and on the temperature T, as shown in Equation 2.6:

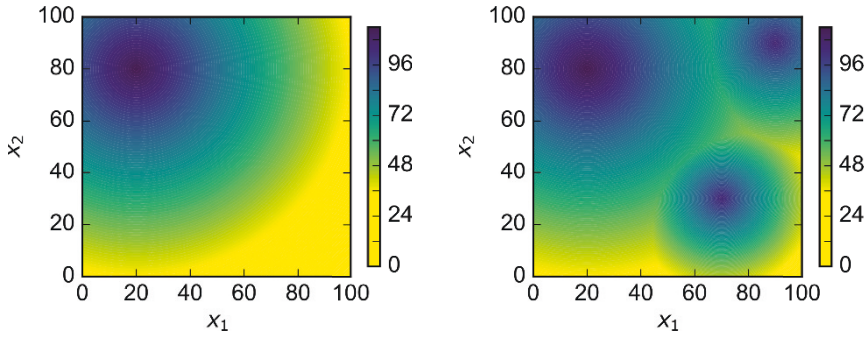
$$M = e^{-\frac{f(\mathbf{x})-f(\mathbf{x}')}{T}} \tag{2.6}$$

The result of the equation is compared to $\mathcal{U}[0,1]$ which is a uniform distributed value between 0 and 1. If the temperature T goes to infinity, Equation 2.6 goes to $e^0 = 1$. As this is nearly always higher than $\mathcal{U}[0,1]$, each solution will be accepted. If the temperature T is $T = 0$, Equation 2.6 goes to $e^{-\infty} = 0$. Therefore, no worse solutions will be chosen. At the end of the current iteration, T is updated. As it applies: $0 < \tau_T < 1$, the temperature T is decreased.

2.3.2. Solution Space

Like already mentioned, solution space $\mathcal{X}$ describes the space in which all solutions are located. So in a continuous solution space, it applies for every solution $\mathbf{x} \in \mathbb{R}^d$ that $\mathbf{x} \in \mathcal{X}$ and therefore $\mathcal{X} \subseteq \mathbb{R}^d$. In discrete and combinatorial optimization problems, there is a discrete solution space. The structure of solution space $\mathcal{X}$ has a large impact on the optimization process. In many cases, a visualization of solution space $\mathcal{X}$ can be very helpful to better understand the problem and to adapt the optimization algorithms. Unfortunately, solution spaces $\mathcal{X} \subseteq \mathbb{R}^d$ for $d > 3$ cannot be visualized without losing information and a lot of the existing real-world problems are more dimensional. Besides the better understanding of the problem, a visualization of solution space $\mathcal{X}$ also helps to get an impression from the behavior of optimization algorithms and situations in the optimization process that leads to problems. A common issue are local optima. An example is given in the following.

Figure 2.2 shows two examples of a two-dimensional solution space $\mathcal{X}$. On the x-axis, the first dimension x_1 is plotted and



(a) With one optimum. **(b)** With three optima.

Figure 2.2.: Two solution space visualizations with $\mathcal{X} \subseteq \mathbb{R}^2$.

on the y-axis, the second dimension x_2 is visualized. In case of a maximization problem, the determination of the best solution is easier in the example of Figure 2.2a because there is only one optimum. This should be demonstrated with an example. The optimal solution $\mathbf{x}^*$ in both solution spaces is $\mathbf{x}^* = (20, 80)^T$ and in this example, improvement heuristics as an optimization algorithm starts with the initial solution $\mathbf{x}^0 = (80, 20)^T$. The heuristic discovers solution space $\mathcal{X}$ in the optimization process. Employing solution space $\mathcal{X}$ shown in Figure 2.2a, the values are monotonous decreasing on the way to the global optimum which makes it easy for the heuristic. Working with the solution space $\mathcal{X}$ shown in Figure 2.2b, the heuristic requires the ability to overcome the local optimum in the lower right part of the solution space $\mathcal{X}$ to find the global optimum. This is quite a harder task than finding the global optimum in the example presented in Figure 2.2a.

It can be seen that the visualization of solution space $\mathcal{X}$ helps to specify requirements for the heuristic, i.e., the optimization algorithm. But to make this clear, the creation of a visualization is very expensive. Thus, a visualization of solution space $\mathcal{X}$ is not available for an efficient algorithm during the optimization process. For example, to create the

visualizations in Figure 2.2, a grid with 100×100 points was laid over solution space $\mathcal{X}$ and for every grid point, the objective function f was computed. An efficient optimization algorithm is able to find the optimum with a comparable accuracy to the visualizations with significantly less objection function calls.

2.4. Representation

To solve a real-world problem, a representation of solutions is needed that can be understood by an optimization algorithm. Possible solutions in the real-world are called candidate solutions. In the context of EAs, it is referenced with the term phenotypes. Genotypes are the solutions for the model. Thus, representation is the process of mapping the phenotypes onto the genotypes [Gorunescu, 2011].

In this thesis, two types of representation are proposed. The inspiration for the binary representation is the interesting property of the turbine placement optimization problem that, independently from the dimensionality of a solution $\mathbf{x}$ and the way the genotypes describe the phenotypes, the solution $\mathbf{x}$ can be visualized on a two-dimensional map. The binary representation uses this two-dimensional perspective for the problem and places grid points on the two-dimensional map. The real-valued representation describes the properties of every turbine as a vector of real-valued numbers. This representation leads to interesting reflections in solution space $\mathcal{X}$. These reflections are shown in Section 2.4.2 and are used to employ powerful variation operators for the heuristic optimization algorithms.

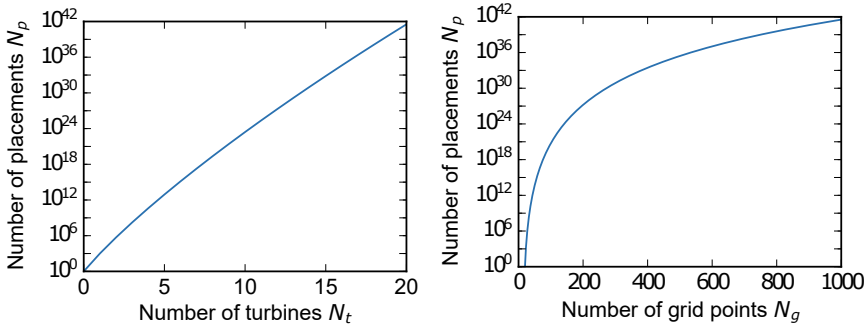
2.4.1. Binary Representation

In a binary representation, the genotype consists of a string of binary digits [Eiben and Smith, 2007]. As binary representations are used for discrete optimization problems [Rao, 1996], the usage leads to a discretization. The basic idea in this approach is a grid that is laid over the map. Every grid point is represented by a bit. If a wind turbine is placed on the grid position, the value of the bit is 1 and

if the grid position is empty, the value is 0. Thus, the turbines may only be placed at certain positions on the grid. For example, the solution $\mathbf{x}^g = [0, 0, 0, 1, 1, 0, 0, 0, 0]$ represents a grid with nine possible locations $N_g = 9$ and a turbine placed at the fourth and fifth location. As there is one value for placed wind turbines, only one type of turbine can be modeled. This matches the requirements of this thesis but the representation could be extended easily. The finite number of grid positions N_g leads to a limited number of possible placements N_p. Theoretically, a brute force approach could compute all placements and choose the best. But in practice, the number of placements N_p is so large that it is not possible to compute the fitness function value for all solutions. Equation 2.7 shows how N_p is calculated:

$$N_p = \frac{N_g!}{(N_g - N_t)! \cdot N_t!} \tag{2.7}$$

where N_t is the number of turbines that should be placed on the grid. The number of possible free locations on the grid is represented by $N_g!/(N_g - N_t)!$ and the term $N_t!$ implements the property that the order of the turbines does not matter. In Figure 2.3a, the number of possible placements N_p depending on an increasing number of



(a) Depending on the number of turbines N_t.

(b) Depending on the number of grid points N_g.

Figure 2.3.: The number of possible placements N_p.

turbines N_t are shown. The number of grid points N_g is fixed to $N_g = 1000$. Figure 2.3b visualizes N_p depending on N_g with $N_t = 20$. In both plots shown in Figure 2.3, the scale of the y-axis is logarithmic and they visualize the behavior of Equation 2.7 for the case $N_g \gg N_t$. As it makes sense to consider the behavior for situations with many more grid points than wind turbines to place. It can be seen that N_p increases approximately exponential with increasing N_t. If N_p depends on N_g with fixed N_t, it is shown that $N_p = 0$ applies for $N_g < N_t$. With increasing N_g, N_p starts to grow very rapidly and then slows down the increasing rate. If $N_p \ggg N_t$, Equation 2.7 can be simplified to Equation 2.8:

$$N_p = (N_g)^{N_t}/N_t! \tag{2.8}$$

With this description, shown in Equation 2.8, it can be easily seen that N_p increases polynomial with N_g for $N_g \ggg N_t$. Thus, the gradient in Figure 2.3b corresponds to a polynomial function if the x-axis goes to infinity.

Table 2.1.: Examples for the number of possible placements N_p.

	Possible locations N_g		
Number of turbines N_t	100	1000	10000
1	1.000×10^2	1.000×10^3	1.000×10^4
3	1.617×10^5	1.662×10^8	1.666×10^{11}
5	7.529×10^7	8.250×10^{12}	8.325×10^{17}
10	1.731×10^{13}	2.634×10^{23}	2.743×10^{33}
20	5.360×10^{20}	3.395×10^{41}	4.033×10^{61}
30	2.937×10^{25}	2.430×10^{57}	3.609×10^{87}

In Table 2.1, examples for the number of possible placements N_p are given. When using a model that is able to compute the fitness function in 100 ms, the computation of 1×10^{10} solutions takes more than 30 years. If there is a map with a size of 5 km $\times$ 5 km and a distance of 100 m between the grid points is used, this grid consists of 2500 grid points. With a feasible rate of 40%, only 1000 points are

possible locations. Table 2.1 illustrates that even at this imprecise scale and with so few grid points to consider, it is not possible to compute all N_p placements for multiple turbines, e.g., 10 turbines would lead to 2.634×10^{23} solutions to verify. Therefore, a brute force approach would not be successful.

2.4.2. Real-Valued Representation

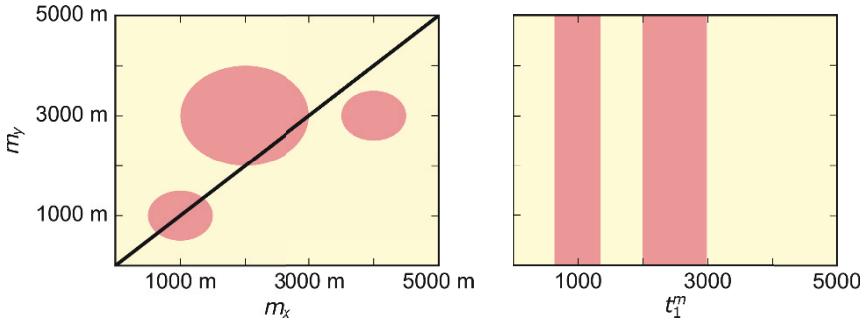
In the real-valued representation, the genotype is described by continuous values. For the optimization process, an advantage of real-valued representations is the easier definition of problem-specific operators compared to other representations [Lee and El-Sharkawi, 2008]. Every property of each turbine is used as a continuous dimension. This leads to the advantage that no discretization on the candidate solutions is needed to describe a solution $\mathbf{x}$. In this thesis, it is defined that each turbine has two dimensions to describe the location of a turbine. So, each turbine has an x- and y-coordinate. For future works, it is possible to extend this definition by addition parameters like the turbine height or the type of turbine. By employing the definition for an optimization problem with N_t turbines, the following applies:

$$\mathbf{x} = [\mathbf{t}_i]_{i=1}^{N_t} \in \mathbb{R}^{2 \times N_t} \tag{2.9}$$

with $\mathbf{t}_i = (x_i^t, y_i^t)$. Equation 2.9 describes the turbine-oriented view of solution $\mathbf{x}$. It is possible to treat this $2 \times N_t$-dimensional optimization problem as a typical N-dimensional problem with $N = 1 \times 2 \cdot N_t$. Then, the solution vector $\mathbf{x}$ is described as follows:

$$\mathbf{x} = [x_i]_{i=1}^{N} \in \mathbb{R}^N. \tag{2.10}$$

An interesting property of the real-valued representation is $f(\mathbf{x}_1) = f(\mathbf{x}_2)$ for $\mathbf{x}_1 = (\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{N_t})^T$ and every $\mathbf{x}_2 = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{N_t})^T$ with $\mathbf{r}_{1,\ldots,N_t} \in \mathcal{T}$, $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{N_t}\}$, and $\{\mathbf{r}_k\} \cap \{\mathbf{r}_l\} = \emptyset$ for every $k, l \in [N_t]$, and $k \neq l$. This property also employs for the constraints and leads to reflections in solution space $\mathcal{X}$; it is caused by the characteristic that the order of the turbines in a solution $\mathbf{x}$ does not matter, e.g., $f((\mathbf{t}_1, \mathbf{t}_2)^T) = f((\mathbf{t}_2, \mathbf{t}_1)^T)$.



(a) A simple map.

(b) One-dimensional solution space for one turbine.

Figure 2.4.: Example of a simple map and the matching one-dimensional solution space for one turbine.

To visualize the reflections in solution space $\mathcal{X}$, Figure 2.4a shows an example of a simple map with the m_x-coordinate on the x-axis and the m_y-coordinate on the y-axis. In this example only the constraints are taken into account. Three circles represent three constraints on the map, e.g., caused by minimum distances to objects in the center of the circles. For this map, solution space $\mathcal{X}$ should be visualized. For a meaningful solution space $\mathcal{X}$, at least two turbines have to be visualized. As there are two-dimensions per turbine, this leads to a four-dimensional plot which is not realizable. Thus, in this example each turbine is placed on one meta-dimension t^m which is projected to the two-dimensional map. The black diagonal line from the lower left to the upper right represents the development of the meta-dimension t^m. It applies:

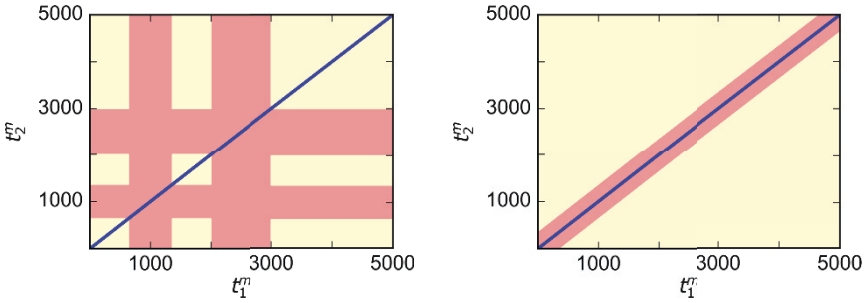
$$\mathbf{t} = (t^m, t^m) \text{ and } t^m = \frac{[\mathbf{t}]_1 + [\mathbf{t}]_2}{2} \text{ for } [\mathbf{t}]_1 = [\mathbf{t}]_2 \tag{2.11}$$

The matching solution space for only one turbine is shown in Figure 2.4b. The scale on the x-axis corresponds to Equation 2.11 and the black diagonal line from the example map. Thus, $x = 1000$ represents the location $m_x = 1000$, $m_y = 1000$ on the map. Again, the red areas

represent non feasible parts of solution space $\mathcal{X}$, while the light yellow regions visualize feasible areas.

Figure 2.5 visualizes the solution space $\mathcal{X}$ for two turbines. The first turbine t_1^m is located at the x-axis and the second turbine t_2^m is represented by the y-axis, e.g., the point $t_1^m = 1000$, $t_2^m = 4000$ represents a solution $\mathbf{x}$ consisting of two turbines $\mathbf{t}_1, \mathbf{t}_2$ at the locations $\mathbf{t}_1 = (1000, 1000)$ and $\mathbf{t}_2 = (4000, 4000)$. In Figure 2.5a, the visualized solution space $\mathcal{X}$ represents the example of the simple map including the three constraints. For example, the solution $\mathbf{x}$ visualized by the point $t_1^m = 1000$, $t_2^m = 4000$ is infeasible as the location $m_x = 1000$, $m_y = 1000$ is not feasible in the example map. Therefore, it is plotted in red. Figure 2.5b shows solution space $\mathcal{X}$ only considering the minimum distance between the turbines. As only the structure should be shown in this plot, no absolute values for the minimum distance are mentioned. It can be seen that the solution is non-feasible if the turbines $\mathbf{t}_1, \mathbf{t}_2$ are too close together and it is feasible in every other case. In both plots of Figure 2.5, it can be seen that solution space $\mathcal{X}$ is mirrored at the diagonally from the lower left to the upper right. This diagonal is plotted with a blue line.

Figure 2.6 takes the constraints on the map and the minimum distance between the turbines into account. Even with just two one-dimensional turbines, only taking the constraints into account and a map with three constraints, solution space $\mathcal{X}$ employs interesting structures. Solution space $\mathcal{X}$ is symmetrical diagonally from the lower left to the upper right. This is caused by the property that the order of the turbines does not matter. Also, it can be seen that solution space $\mathcal{X}$ consists of many feasible islands. These properties are much stronger when using multiple two-dimensional turbines. It leads to multiple reflections in solution space $\mathcal{X}$ and much more feasible islands. The knowledge of these effects can be used to employ powerful variation operators for the heuristic optimization algorithms.



(a) Considering the constraints on the map.

(b) Considering the minimum distance between the turbines.

Figure 2.5.: Solution space $\mathcal{X}$ for two turbines.

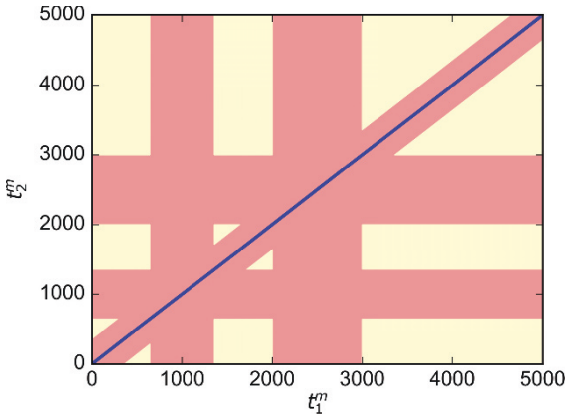


Figure 2.6.: Solution space $\mathcal{X}$ for two turbines including constraints on the map and minimum distance between the turbines.

2.5. Remarks

In an optimization problem, an input vector $\mathbf{x}$ is searched for. By employing a model f, this input vector $\mathbf{x}$ leads to an output value. An optimization approach attempts to find an $\mathbf{x}$ that lead to a specific output value, e.g., a minimum or maximum value. There

are various approaches to solving optimization problems. Which approach is the most suitable depends on the problem. To solve an optimization problem that is not solvable in polynomial time, heuristic optimization approaches are a good choice [Rothlauf, 2011]. This applies for many real-world problems. There are two kinds of optimization heuristics: construction heuristics and improvement heuristics. Improvement heuristics perform usually slower but are able to create better solutions [Schneider and Kirkpatrick, 2006]. They iteratively explore solution space $\mathcal{X}$. Two popular approaches are EAs and SA. Requirements on the variation operators of the approaches depend on the properties of solution space $\mathcal{X}$ which depends on the problem and the representation of candidate solutions. In this thesis, two representations for the turbine placement optimization problem are proposed. In the binary representation, it can be seen that even for simple and imprecise settings, the number of possible placements is huge. Thus, a brute force method would not succeed. The real-valued representation leads to interesting reflections in solution space $\mathcal{X}$. In this thesis, the knowledge of these effects will be used to employ powerful variation operators.

http://www.springer.com/978-3-658-18464-3

Evolutionary Wind Turbine Placement Optimization with Geographical Constraints

Lückehe, D.

2017, XXII, 195 p. 64 illus., 15 illus. in color., Softcover
ISBN: 978-3-658-18464-3