Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet
Theorie der hybriden Systeme

Analyse der Diskretisierung von Genparametern Evolutionärer Algorithmen

Analysis on Discretization of Gene Parameters in Evolutionary Algorithms

Masterarbeit
Informatik

Dezember 2017

Vorgelegt von Presented by	Lukas Netz Lochnerstr. 65 52064 Aachen Matrikelnummer: 297189 lukas.netz@rwth-aachen.de
ErstprüferIn First examiner	Prof. Dr. Erika Ábrahám Lehr- und Forschungsgebiet Theorie der hybriden Systeme RWTH Aachen University
Zweitprüfer Second examiner	Jr.-Prof. Dr. Christina Büsing Lehrstuhl für Mathematik (MathCCES) RWTH Aachen University
Korefferent Co-supervisor	Dr. rer. nat. Pascal Richter Lehrstuhl für Mathematik (MathCCES) RWTH Aachen University

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Aachen, im Dezember 2017

Lukas Netz

Dedicated to my father
Priv. Doz. Dipl.-Phys. Dr. med. Johannes Netz

I wish to express my deepest gratitude to both of my parents. By virtue of their lasting support I could follow my dreams and my curiosity, leading me to this degree. I can not imagine getting greater encouragement.
Thank you.

I also wish to thank my adviser Dr. Pascal Richter and his team of research assistants for their time and patience as well as Prof. Dr. Erika Abraham for enabling me to research this fascinating topic.

Contents

List of Figures

List of Tables

Acronyms

CVC	Continuous Value Chromosome
DVC	Discrete Value Chromosome
GA	Genetic Algorithm[1]
AEP	Annual Energy Production
LCOE	Levelized Cost of Electricity
NPV	Net Present Value
IRR	Internal Rate of Return

List of Symbols

N_{pop}	Number of chromosomes in the population from generation to generation	
N_{gene}	Number of values in a chromosome	
N_{sel}	Number of chromosomes in the selection pool	
N_{elite}	Number of elite chromosomes, that are passed through to the next generation	
$N_{offspring}$	Number of offspring generated each generation	
N_{repl}	Number of chromosomes in the old generation, that get replaced	
N_{rand}	Number of random chromosomes, that are inserted into each new generation	
N_T	Number of turbines	
r	Uniform random number	
μ	Mutation rate	
$n_{\mu_{min}}$	Minimum number of genes to mutate	
$n_{\mu_{max}}$	Maximum number of genes to mutate	
γ	Chromosome	
Γ	Population	
f	Fitness	
α_i	Wind direction sector	
z	Hight of turbine hub (mast hight)	[m]
d	Turbine diameter	
u	Wind Speed	
δu	Wind velocity deficit	
X_{max}	Maximum X-coordinate	[m]
Y_{max}	Maximum Y-coordinate	[m]
R	Radius	[m]
ρ	Discrete mapping resolution	
c	cell index	
p_c	Probability to get chosen	
p_s	Probability to get selected	
D	Domain of a parameter	
A	Algorithm search space	

[1] In the context of this thesis we refer to a 'genetic algorithm' as a synonym of 'evolutionary algorithm'

1. Introduction

"Living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame."[2]

Genetic Algorithms (GAs) mimic the evolutionary process we observe in nature and are robust and powerful optimizers [13][16]. GAs perform very well in a great variety of areas. A promising application of a genetic algorithm was implemented by Richter [24]. The GA was used to improve an offshore wind park by optimizing the turbine layout of the plant.

Although, known for their robustness, GAs are often not applicable in commercial use due to their computational cost of evaluation. The algorithm usually needs to evaluate hundreds or thousands of times per iteration,which is based on simulations that can require a runtime of several minutes up to days. This might result in a runtime of several months. The long runtime of the algorithm might not be an issue if applied in the temporal context of power plant construction, but genetic algorithms have to be tailored to the specific problem which they aim to optimize. Such a long runtime can provide a huge problem in case the algorithm has to be fine tuned or adjusted [3]. Especially considering the overwhelming amount of ways to set up and configure a genetic algorithm. Optimizing the layout of offshore wind parks is becoming more and more important as seen in Figure 1: There are more than 68 wind parks either in design or applied in the German bay alone.

In this work, we analyze whether gene parameter discretization is a viable option to speed up the algorithm in both run time and convergence behavior. We will compare the findings with an algorithm relying on continuous gene parameter values. We implement a dedicated plug-in for an existing genetic algorithm library and evaluate the performance in simulations in relation to the discretization. A simulated offshore wind park as well as several simple objective functions will be used to evaluate the discretization effect in comparison to the usage of continuous values.

1.1. Thesis overview

In this thesis, we first introduce a general background of genetic algorithm based optimization and the current state of the research on involved in the Section 2. In Section 3, we present the model used in the simulations. In Section 4, we illuminate the genetic algorithm and its operators that are implemented. Section 5 contains an analysis on how given operators influence the convergence of the algorithm. In Section 6, we provide experimental results, followed by the evaluation and a conclusion in Section 7.

[2]Prof. John Henry Holland, a pioneer in the field of Genetic Algorithms (John Holland)[1][16]

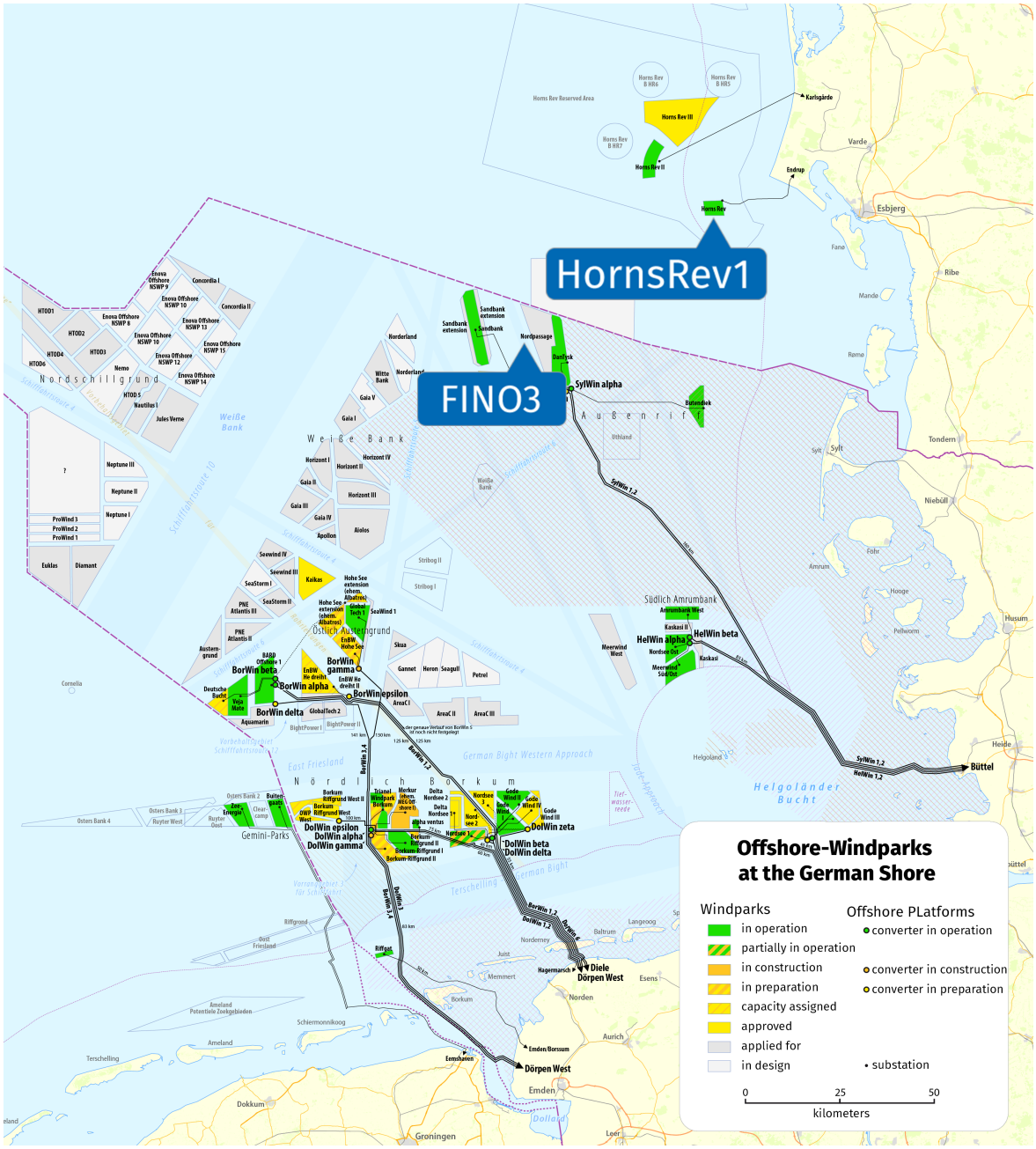


Figure 1: Current Offshore wind parks in the German bay. Green areas indicate offshore wind parks that are in operation. Orange areas are parks, that are in construction. Yellow, and gray areas represent planned wind parks. The wind park we simulate HornsRev1 can be found in the top right on the map. The platform that provided the wind data FINO3 can be found next to it. Map created by Maximilan Dörrbecker (CC-BY-SA 2.0). Translated and edited by Lukas Netz.

2. Background

Genetic algorithms (GAs) have become more popular and efficient in recent years. Progress in the topic has allowed more complex real-world problems in optimization, search and also machine learning, to be solved by GAs. This chapter gives a brief overview of the current state of research on the topics of this thesis. First we take a look at research on improving the efficiency of GAs. Next, we highlight research on methods to find optimal configurations for GAs.

2.1. Literature Review on Efficiency Improvements on Genetic Algorithms

Currently, there exist many ways to improve the optimization process of a genetic algorithm. Ghoshray and Yen succeed in improving accuracy and runtime by implementing a modified genetic algorithm (MGA) by combining the genetic operators such as selection, reproduction, crossover, mutation and simulated annealing [11]. Rivas-Davalos and Irving lowered solution times, by improvements to the genetic operators in order to overcome the problems of low heritability and topological infeasibility [25]. Other attempts were made to enable existing libraries to increase their abilities to run on parallel processes [10][23]. This attempt however is limited by Amdahls law: The speedup $\eta(s)$ of an algorithm is limited by the runtime of the serial program sections, where s is the amount of system resources (e.g. processor cores) and p is the portion of the algorithm that is improvable by the increment of system resources.

$$\eta(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{1}$$

Despite the library we use in this thesis implements parallelization, we will not focus on the resulting benefits in runtime. The configuration for the parallel setup remain identical throughout all experimental set ups and is not part of this thesis.

2.2. Literature Review on Optimal Configurations of Genetic Algorithms

Configuring a genetic algorithm proves to be a difficult task, due to its long runtime and variety of ways to configure it. Ironically, a lot of research on the configuration of such complex optimizers relies itself on genetic algorithms to accomplish this task [3][4]. It would be infeasible to use a genetic algorithm in order to optimize a fitting configuration for our algorithm setup. Kucukkoc et. al. optimize the configuration by using response surface design (RSM) [20]. Eiben and Smit developed a framework to fine tune parameters of evolutionary algorithms [9]. Both approaches optimize the GA as a whole. Eiben et. al. also published research on how to control the parameters of evolutionary algorithms [8]. Although there has been a lot of research on the topic of parameter configuration of genetic algorithms, finding fitting parameters is still a

very time consuming operation. All presented approaches require many iterations of the genetic algorithm in order to optimize the parameters.

2.3. Literature Review on Discretization of Genetic Algorithm parameters

There is not a lot of research done on the difference between a genetic algorithm using continuous values and one using discrete ones, due to the fact that historically, genetic algorithms only use bit strings to encode a solution. More complex encodings such as floating point numbers are still not common. There is a study on discretization scheduling and its benefits on the precision of the optimization [22].

2.4. Summary

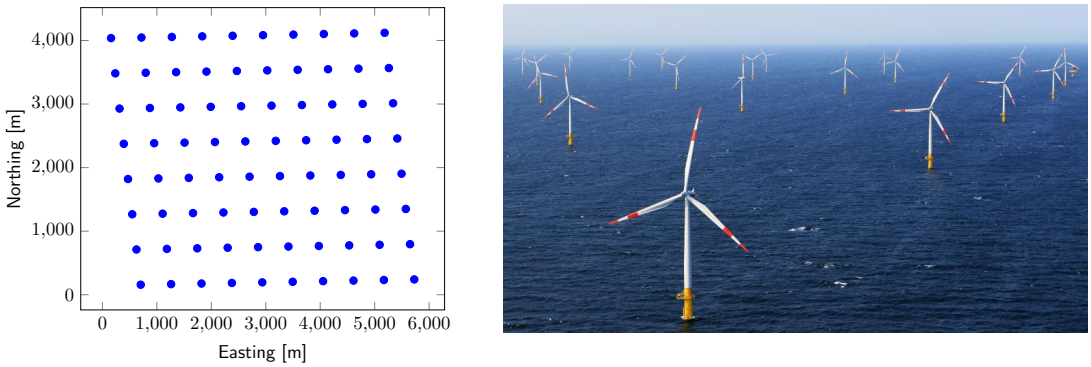
In this chapter we gave an overview of the current state of art on research on genetic algorithms. We highlighted that this thesis is involved in three research areas: *configuration*, *efficiency* and *discretization* of genetic algorithms.

3. Model

The optimization algorithm implemented as part of this thesis is configured to optimize layouts of *Offshore Wind Farms*. In this chapter we will introduce the characteristics and the basic parameters of the model we use to simulate the power plant. We also introduce four objective functions that we use to test our implementations.

3.1. Offshore Wind Farm Model

Wind farms are sets of wind turbines arranged in an efficient layout (Figure 2a). Offshore wind power refers to wind farms that are constructed in bodies of water. Unlike the typical use, in this context the term "offshore" also refers to inshore water areas, such as lakes, fjords and sheltered coastal areas.



(a) Typical layout of an Offshore Wind park (HornsRev1)

(b) Offshore Wind park at the German coast

Figure 2: Wind turbine layout in a typical offshore wind park.

The model we use to simulate an offshore wind park was initially developed by Heiming [15], and can be described by four components (Figure 3). At first the model will convert raw wind data into a *wind model*. Next the wind model is used as an input for the *wake model*, which simulates the interaction of wind turbines in the wind park. The resulting data is used to compute the power generation at each wind turbine (*power generation model*), which gives us the gross annual energy production (AEP). Finally we pass the results to the *cost model* in order to consider different quantities of interests.

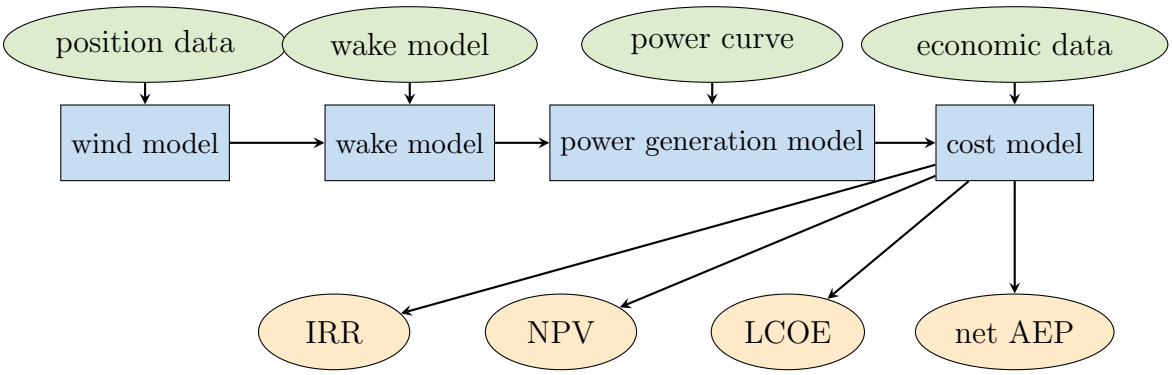


Figure 3: Simplified structure of the offshore model. Rectangles represent the building blocks of the model and represent the four main building blocks of our simulation. Green ellipses represent inputs, red ellipses represent outputs for the different objective functions (see Section 3.5).
Source: Adapted from Cakar [5]

3.2. Wind Model

The wind model we use as an input for our model describes the strength and direction of the wind as a probability distribution. Our model is based on thousands of measurements over seven years at the FINO3[3] research platform.

3.2.1. Wind Direction

FINO3 is located 80 *km* west of Sylt and can be found next to the DanTysk wind park (Figure 1). We can visualize the measured wind direction distribution as shown in Figure 4. The wind directions are divided into 360 sectors, 32 sectors and 12 sectors. The size of each sector translates to the probability that a wind direction from that sector occurs. The distribution of the wind direction is not evenly divided among all sections. A probable wind direction exists (northeast).

[3]http://www.fino3.de/

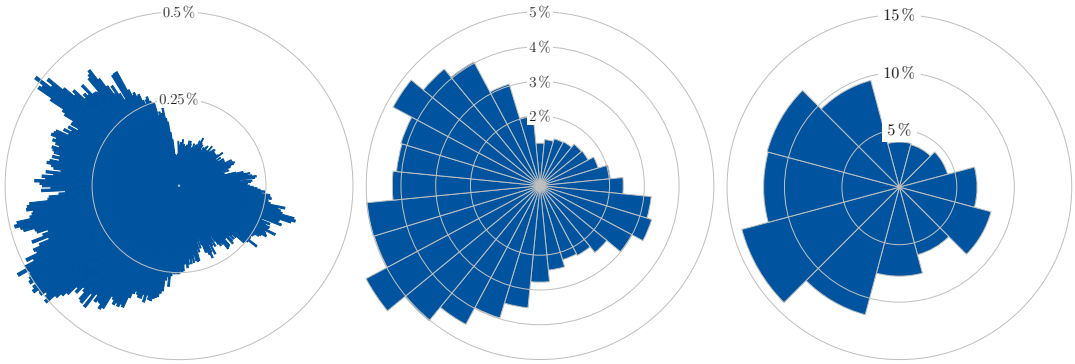


Figure 4: Distributions of wind directions from 2010 to 2017 in three different discretizations. From left to right: 360 sections, 32 sections, 12 sections. The wind direction $\alpha_i = 220$ is among the most likely ones.
Source: Heiming [15]

3.2.2. Wind Speed

The wind model we use employs a Weibull distribution (Equation 2) to approximate the wind speed and direction. Heiming used a maximum likelihood estimation for the parameters of the Wibull distrubution. Each wind section was estimated separately. That means each wind section has its own probability and its own wind speed probability distribution. As examined by Heiming [15], the Weibull distribution provides a good approximation of measured wind speed data. It is described by two parameters: The scaling factor λ and the shape parameter κ.

$$h_{\lambda,\kappa}(x) = \frac{\kappa}{\lambda}\left(\frac{x}{\lambda}\right)^{\kappa-1} \mathrm{e}^{-(x/\lambda)^{\kappa}} \tag{2}$$

A typical distribution of wind speeds is displayed in Figure 5. It is important to note that this set of distributions can only be used to simulate wind in the German bay, as it is based on the measurements from that area. These approximations may not apply to other locations such as offshore wind parks in the Chinese sea or the English Channel.

3.3. Wake Model

Once a wind model is established, we need to approximate the interactions between individual wind turbines. The wind turbine used in our simulation is a HAWT (horizontal axis wind turbine) as depicted in Figure 6. The rotor has the diameter D and can be rotated 360 degrees. The hight of the hub is defined as z. In this chapter we will examine the turbulences caused by HAWT, and how theses turbulences affect near by turbines.

In this work we use the PARK model to simulate the wake effect of wind turbines. The PARK model developed by Jansen [19] approximates the wind speed reduction

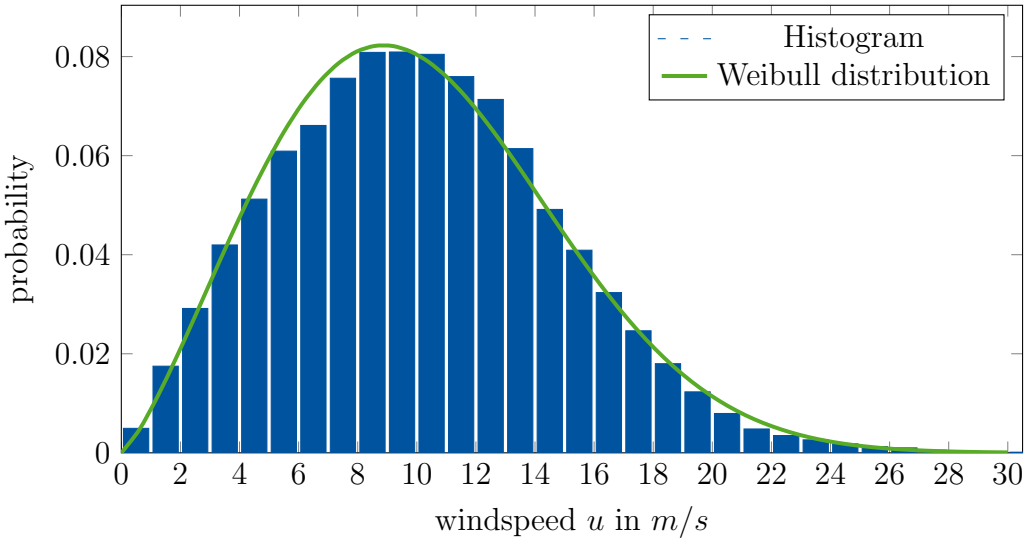
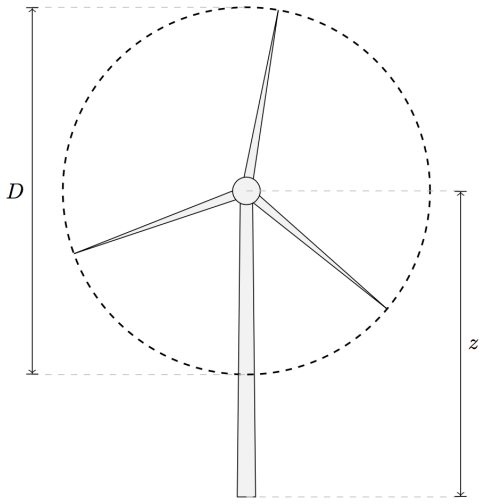


Figure 5: Weibull distribution with approximated values for both λ and k. The plot represents the distribution for a wind direction sector $\alpha_i \in [225^o, 255^o)$ Source: Adapted from Cakar [5].



(a) Horizontal Axis Wind Turbine
Source: Heiming [15]

(b) Windmill CP-A4
Picture by Hans Hillewaert (CC-BY-SA 4.0)

Figure 6: HAWT wind turbines

inside the stream behind a wind turbine. The model does not compute a flow field for the exact wind velocity distribution, thus it ignores e.g. side streams effects. The wake model, as implemented by Heiming, does not account for different wind speeds in different heights. We used wind data that was measured 100 m above sea level. We can assume that small differences in height will have negligible effect on the wind speed. If we look at the wind turbines of HornsRev1 we notice that the height of their mast is 70 m and its wing span is 100 m, hence we can use the measured data. The implemented wake model is designed for long distances between the wind turbines, and requires distances of at least three rotor diameters to be valid. Each turbine creates a wake with a diameter d_w (Figure 7). The diameter of this wake grows linearly by the factor $2k$. The wake decay factor k is defined as follows [27]:

$$k = \frac{1}{2} \cdot ln\frac{z}{z_0} \tag{3}$$

where z represents the height of the turbine mast (Figure 6a) and z_0 is the surface roughness of the site ground. Depending on the weather conditions z_0 can change (e.g. high waves). Heiming however assumes $z_0 = 0.03$ as a constant value, as this is the most common value. The PARK model only accounts for wind velocity decrements inside the wake in stream direction. We define the wind decrementδu by a wind turbine as:

$$\delta u = \frac{u_0 - u_w}{u_0} = 1 - \frac{u_w}{u_0} \tag{4}$$

Where u_w represents the decremented windspeed behind a turbine, and u_0 represents the original wind speed in front of the turbine. As it only depends on the distance in stream direction x, we can write:

$$\delta u = \delta u(x) \tag{5}$$

Heiming computes the velocity deficit over the conservation of momentum. For an isolated system the following has to hold:

$$\sum mass \cdot velocity = \sum density \cdot area \cdot velocity = 0 \tag{6}$$

We approximate by assuming incompressibility of our medium (air) and get:

$$-\rho_{air}\pi \left(\frac{d}{2}\right)^2 u_r - \rho_{air}\pi \left(\left(\frac{d_w}{2}\right)^2 - \left(\frac{d}{2}\right)^2\right) u_0 + \rho_{air}\pi \left(\frac{d_w}{2}\right)^2 u_w = 0 \tag{7}$$

Where u_r represents the wind speed directly behind the rotor. We simplify to:

$$d^2 u_r + (d_w^2 - d^2)u_0 = d_w^2 u_w \tag{8}$$

Next, we compute the wind speed deficit right behind the rotor using Equation 4 and 3:

$$\frac{u_2}{u_0} = 1 - \delta u_r \left(\frac{d}{d + 2kx}\right) \tag{9}$$

Following, we have to replace the δu_r with the relative loss at the turbine, where C_t is the thrust coefficient of the turbine:

$$a(u_0) = 1 - \sqrt{1 - C_t(u_0)} \tag{10}$$

which gives us:

$$\delta u(x) = 1 - \frac{u_w(x)}{u_0} = \frac{1 - \sqrt{1 - C_t(u_0)}}{\left(1 + \frac{2kx}{d}\right)^2} \tag{11}$$

Now that we have derived the wind speed deficit caused by a wind turbine in a free stream, we have to compute how a second wind turbine either completely or in part inside of the generated wake is affected by it. The first turbine that generates the wake is called i and the turbine inside the wake is called j. The wind speed at turbine i is called incedent wind speed $u_{inc,i}$. In case turbine j is completely within the wake of i we simply use the Equation 11. Otherwise we need to compute the part to which j is affected. We do so by introducing the shadowing factor β_k:

$$\beta_k = \frac{A_{int}}{A_{turbine}} \tag{12}$$

where A_{int} is the area of wake that intersects with the area of turbine j. We update our equation to:

$$\delta u(x, u_{inc,i}) = \beta_k \cdot \frac{1 - \sqrt{1 - C_t(u_{inc,i})}}{\left(1 + \frac{2kx}{d}\right)^2} \tag{13}$$

The equation above is dependent on the incident velocity $u_{inc,i}$ of turbine i. We need to generalize it and derive the dependency to the constant u_0.

$$\delta u_{i,j} = 1 - \frac{u_{w,i}}{w_0} = 1 - \frac{u_{inc,j}}{u_0} = \frac{u_0}{u_{inc,i}} \left(\beta_k \cdot \frac{1 - \sqrt{1 - C_t(u_{inc,i})}}{\left(1 + \frac{2kx}{d}\right)^2} \right) \tag{14}$$

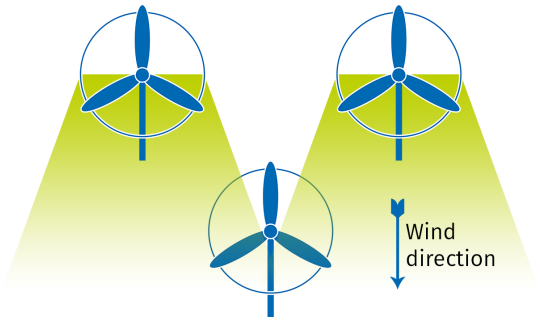
We compute the interaction of wakes as follows:

$$\begin{aligned} \delta u_j^2 &= \delta u_{1,j}^2 + \delta u_{2,j}^2 \\ \Leftrightarrow \left(1 - \frac{u_{inc,j}}{u_0}\right)^2 &= \left(1 - \frac{u_{w,1}}{u_0}\right)^2 + \left(1 - \frac{u_{w,2}}{u_0}\right)^2 \end{aligned} \tag{15}$$

The parameters $u_{w,1}$ and $u_{w,2}$ represent the velocity within the respective wakes. We can generalize to an arbitrary number of N interacting wakes by computing:

$$\delta u_j = \sqrt{\sum_{i=0}^{N} \delta u_{i,j}^2} \tag{16}$$

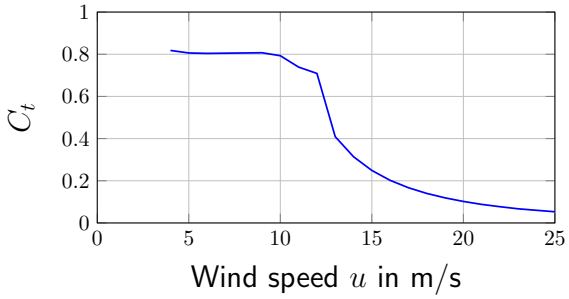
We successfully derived an equation, that approximates how turbines affect the wind in case they are placed in proximity of each other.



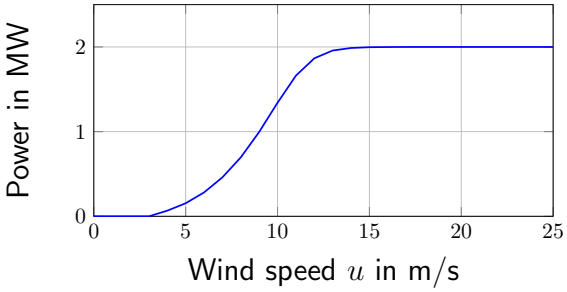
(a) Modeling of a wake behind a wind turbine. The turbulence caused by a turbine influences the ones positioned behind it.

(b) Extraordinary weather conditions resulted in condensation of the very humid air, revealing the turbulence patterns behind the wind turbines [6]
Picture by Christian Steiness (12.02.2008)

Figure 7: Wake turbulence created by a wind turbine



(a) Thrust coefficient C_t.



(b) Power production P

Figure 8: Performance of the Vestas V80 wind turbine with a cut-in speed of 4 m/s and cut-out speed of 25 m/s. Note that the power generation is not linearly dependent on the wind speed.

3.4. Power Generation Model

The horizontal axis wind turbine is an electric power generator. In converts motion power from the rotation of its wings into electric power. Thus creating electric energy from wind. Each wind turbine has a range of wind speeds it operates in. The lower bound of this range is called cut-in speed u_{cutin} and the upper bound is called cut-out speed u_{cutout}. In case the wind speed is lower than the cut-in speed, there is not enough wind to produce energy. In case the wind speed is greater than the cut-out speed, the wind turbine is at risk to get damaged. Typically cut-in speed is at about 3 to 4 m/s (14 km/h) and cut-out speed is about 25 m/s (90 km/h). The power generation is highly dependent on the thrust of the wind. Figure 8 displays the performance of the Vestas V80 wind turbine, which is used both in onshore as well as offshore applications.

3.5. Cost Model

The cost model evaluates different quantities of interests for a given wind farm. Different objective functions are provided, that are based on the *Gross Annual Energy Production* (AEP). We first take a look at gross AEP, and continue presenting the objective functions we employed for the GA.

3.5.1. AEP

The AEP is computed by Heiming as the expected power value of a wind farm for all wind speeds u in one wind direction α_i. We use the Weibull distribution f_{α_i} to compute the wind speed distribution at the wind direction α_i (Section 3.2, Wind Model). Additionally we use the Wake Model (Section 3.3) to calculate wind velocity deficits δu behind turbines. The Power Generation Model (Section 3.4) is used to compute the generated power $P(u)$ at wind speed u. The gross AEP is computed as follows:

$$\begin{aligned} AEP_{gross} &= (8760\ h + 6\ h) \cdot P \\ &\approx (8760\ h + 6\ h) \cdot \sum_{i=1}^{N_{directions}} w_{\alpha_i} \cdot \sum_{j=1}^{N_{speeds}} w_j \cdot W_{\alpha_i}(u_j) \sum_{k=1}^{N_T} P(u_{inc,\alpha_i,j,k}) \end{aligned} \tag{17}$$

Where $u_{inc,\alpha_i,j,k}$ represents the incident velocity at turbine k, with wind speed u_j and wind direction α_i. 8760 h is the time that passes in a year. 6 h are added in order to compensate for leap years. Next, we present the objective functions we use for the genetic algorithm.

3.5.2. Net Annual Energy Production

In order to approximate a real world offshore wind park, we need to assume, that there are energy losses due to technical issues, maintenance or grid downtime over the year. The statistical value can be summarized to plant performance losses p_{loss}. We compute the net AEP as follows:

$$AEP_{net} = AEP_{gross} \cdot (1 - p_{loss}) \tag{18}$$

When used as an objective function, we try to maximize AEP_{net}. The value is normalized against AEP_{max} the maximum AEP possible. AEP_{max} is computed under the assumption that there are no losses such as δu. Each turbine is simulated at its ideal performance.

3.5.3. Levelized Cost of Electricity

The *Levelized Cost of Electricity* (LCOE) is a measurement for the cost per kWh over the entire life cycle of a wind park. LCOE is a good measurement to evaluate

amortization of the power plant. The implemented formula was developed by Lackner and Elkinton [21]:

$$K_{LCOE} = \frac{C_{capital} \cdot \frac{(1+r_{rate})^l \cdot r_{rate}}{(1+r_{rate})^l - 1} + C_{om}}{AEP_{net}} \tag{19}$$

The parameter $C_{capital}$ represents the total cost for turbines, cabling, substations etc.. C_{om} refers to the annual cost of operation and maintenance, r_{rate} is the discount rate including debt, taxes and insurance over an expected life cycle of l years.

3.5.4. Net Present Value

The Net Present Value (NPV) is a measurement for the current value of a project. Gonzales et al. [14] compute the NPV as follows:

$$C_{NPV} = C_{prv} - C_{capital} + \sum_{t=1}^{l} \frac{AEP_{net} \cdot K_{energy} - C_{om}}{(1+r_{rate})^t} \tag{20}$$

The parameter C_{prv} represents the current value of the project after the lifetime l. Similar to LCOE $C_{capital}$ represents the cost of the installation and C_{om} the cost to operate and maintain the project. K_{energy} translates to the current price of energy on the market. The parameters K_{energy}, C_{om} and AEP_{net} are time dependent. For our model we approximated these parameters as constants and replaced them with the respective average value.

3.5.5. Internal Rate of Return

Similar to NPV, the Internal Rate of Return (IRR) is a measure of profitability. The Internal Rate of Return r_{IRR} is defined to be the value of r_{rate} in Equation 20 that results in a NPV of zero.

$$C_{NPV} = C_{prv} - C_{capital} + \sum_{t=1}^{l} \frac{AEP_{net} \cdot K_{energy} - C_{om}}{(1+r_{IRR})^t} \stackrel{!}{=} 0 \tag{21}$$

If r_{IRR} is greater than the real discount rate r_{rate} (and additional risk deficits), then the project is expected to be profitable.

3.6. Alternative Objective Functions

Using a simulation to evaluate a layout is very time consuming and is one of the greatest contributers to the long run time of the genetic algorithm. As long as we are analyzing the behavior of the genetic algorithm we will use simpler objective functions to evaluate the layout. The following four methods are provided:

- `CORNER`: This objective function pushes all turbines into the corners (Figure 9)

$$f(x, y) = \sqrt{(40 - x)^2 + (40 - y)^2} \tag{22}$$

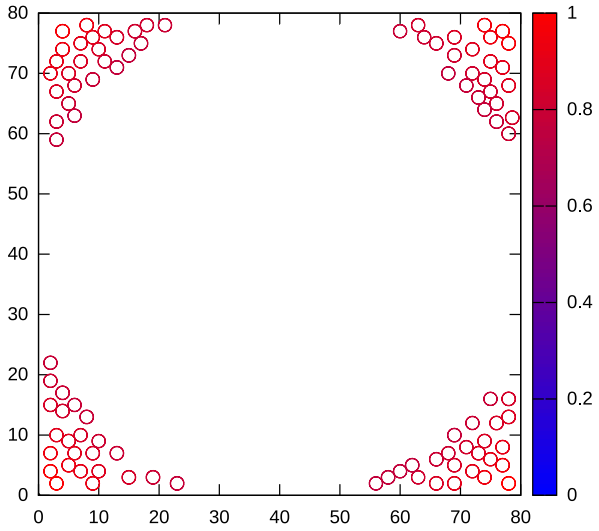


Figure 9: Visualization of the objective function `CORNER`. Turbines that are closest to the corners are evaluated with the highest fitness.

- `MAX SIDE`: This objective function promotes turbines with a high X value, pushing them to one side (Figure 10)

$$f(x, y) = |x| \tag{23}$$

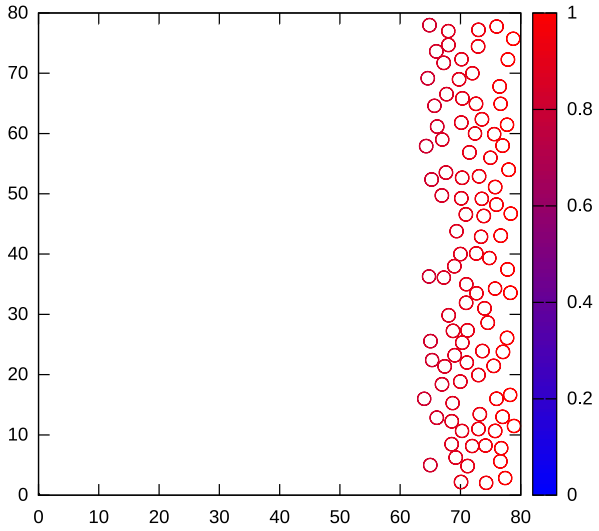


Figure 10: Visualization of the objective function `MAX SIDE`. Turbines with a greater X value get a higher fitness value.

- `TRIANGLE`: This objective function is similar to `MAXSIDE` but also maximizes the Y value of a turbine (Figure 11)

$$f(x, y) = x + y \tag{24}$$

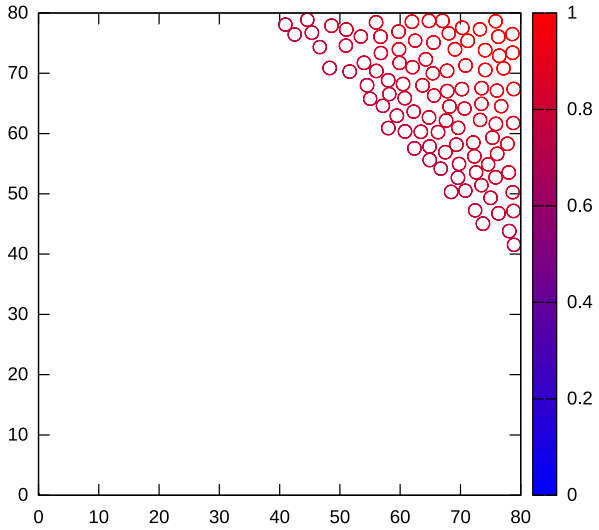


Figure 11: Visualization of the objective function `TRIANGLE`. Tubines with both a high X Value and Y Value get a high fitness.

- `RHOMBUS`: This objective function aims to fit all turbines within a rhombus at the

center (Figure 12)

$$f(x,y) = \frac{5 \cdot X_{Max}}{8} - |\frac{X_{Max}}{2} - x| - |\frac{X_{Max}}{2} - y| \tag{25}$$

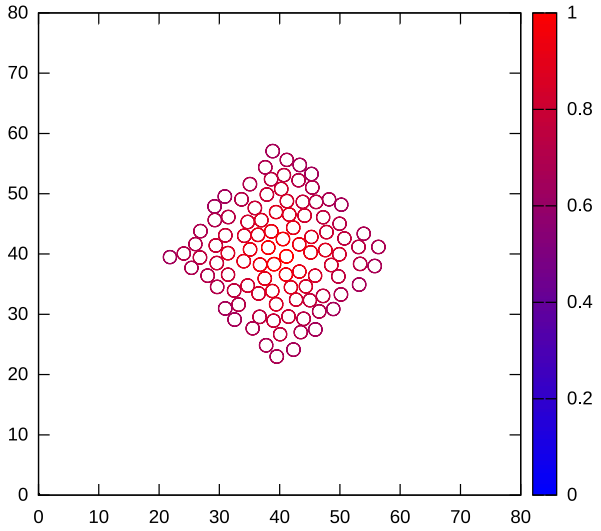


Figure 12: Visualization of the objective function `RHOMBUS`. Turbines close to the center get a high fitness value.

The objective functions introduced above present a viable alternative to test the performance of the genetic algorithm without the need to simulate the entire model.

3.7. Placement Restrictions

The purpose of the simulation however is first to determine where a turbine can be placed. Turbine locations are subject to three restrictions which need to be considered in the placement decision.

1. Every wind turbine has a position (x_i, y_i). It has to be placed within the designated plant site.

$$\begin{aligned} \forall i(X_{min} \leq & x_i \leq X_{max}) \\ \forall i(Y_{min} \leq & y_i \leq Y_{max}) \end{aligned} \tag{26}$$

2. No turbine may be placed within a restricted area, such as a nature reserve or cabling routes

$$\forall i(p_i \notin A_{restricted}) \tag{27}$$

Where $p_i = (x_i, y_i)$ is the position of a turbine, and $A_{restricted}$ denotes the restricted area.

3. The minimal distance between two turbines has to be greater than its wingspan.

$$\forall i, j \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < R \right) \tag{28}$$

where R represents the minimal distance between two turbines.

3.8. Summary

In this section we introduced the model for optimization with the genetic algorithm. We presented the four major building blocks of the model (*Wind Model, Wake Model, Power Generation Model* and *Cost Model*) and their interaction. We also presented the four smaller test functions, used to test and analyze the algorithm in a time efficient manner. Finally, we pointed out the placement restrictions every turbine has to obey, both in simulation as well as in any of the test functions.

4. Genetic Algorithms

This thesis analyzes the behavior of a genetic algorithm. In this section we will introduce the major building blocks of the GA, which are also known as genetic operators. First, we present the genetic algorithm library we used, followed by the two different chromosome encodings we employed. Next, we will present the eight components that define the genetic algorithm set-up we used. The operations are presented in the order as they are executed in the algorithm (see Figure 13).

1. Chromosome Factory
2. Stopping Criteria
3. Selection Operation
4. Coupling Operation
5. Crossover Operation
6. Choosing Operation
7. Mutation Operation
8. Replacement Operation

There are many ways to configure the GA, as every genetic operator can be instantiated in several ways. In this section, we will highlight the common operators and illuminate why we chose a specific one.

Genetic Algorithms are local search algorithms that are based on the mechanics of natural selection and evolution [17][26]. A set of solutions Γ is optimized by the principles of survival of the fittest. Solutions are ranked, get selected to form couples and produce new solutions (offspring), which in part, get mutated and form the next generation with the remaining set. The set of solutions is always treated as one group. Thus the algorithm is able to evolve towards multiple optima in parallel. The set is called *population*, each solution γ within the population Γ is called *chromosome*. Each population is assigned to one *generation*, which refers to the amount of iterations the algorithm has performed. Each chromosome within the population is unique and can be ranked based on its *fitness*:

$$f(\gamma_i) \in [0, 1]$$

The goal in each iteration is to find a chromosome that is better than the best of the previous iteration.

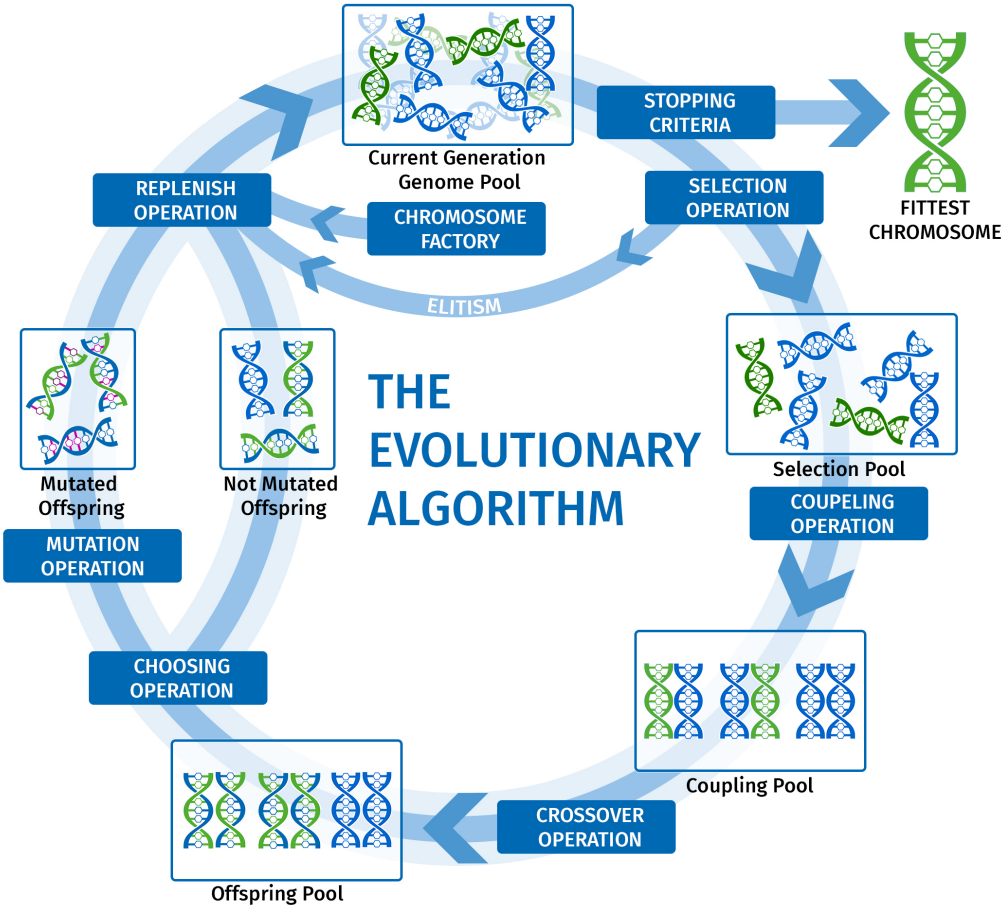


Figure 13: Simplified iteration of an evolutionary algorithm. The algorithm starts at the top with the current generation. Following a test for the *Stopping Criteria*, each chromosome is evaluated and ranked. Based on the ranking chromosomes are selected (*Selection Operation*) and copied to the next generation (*Elitism*). Only the selected chromosomes are paired to couples (*Coupling Operation*) and recombined to form new Chromosomes (*Crossover Operation*). A part of the offspring is chosen (*Choosing Operation*) to get mutated (*Mutation Operation*). All offspring and the elite is added to the old generation. The best chromosomes and some randomly generated ones (*Chromosome Factors*) form the new generation.

4.1. Genetic Algorithms Library

In this work, we use an extensible genetic algorithms library called GeneiAL [10][23]. The library is designed to be highly flexible. It permits the user to specify own chromosome types, genetic operators and fitness functions. It is multi-threading capable and provides several tools for analysis and diagnostics of the algorithm's performance. In order to compare both discrete- and contiuous-value based chromosomes, several operators are implemented in form of a plug-in.

4.2. Genotype Phenotype Mapping

The Genotype Phenotype Mapping defines within the context of genetic algorithms the method used to encode a solution for a given problem as the genetic information of a chromosome. We use two different encodings for the chromosomes. One encoding allowing the encoding of *Continuous Value Chromosomes* (CVC) and one allowing the encoding of *Discrete Value Chromosomes* (DVC). Therefore we have to provide adapted methods for all operations that work directly on values that are contained in a chromosome. This effects the *Chromosome Factory* (Section 4.3), *Crossover Operation* (Section 4.7) and *Mutation Operation* (Section 4.9). Other operations like the *Selection Operation* (Section 4.5) or the *Replacement Operation* (Section 4.10) are not affected, as they manage chromosomes as a population independent from the encoding.

4.2.1. Continuous Value Mapping

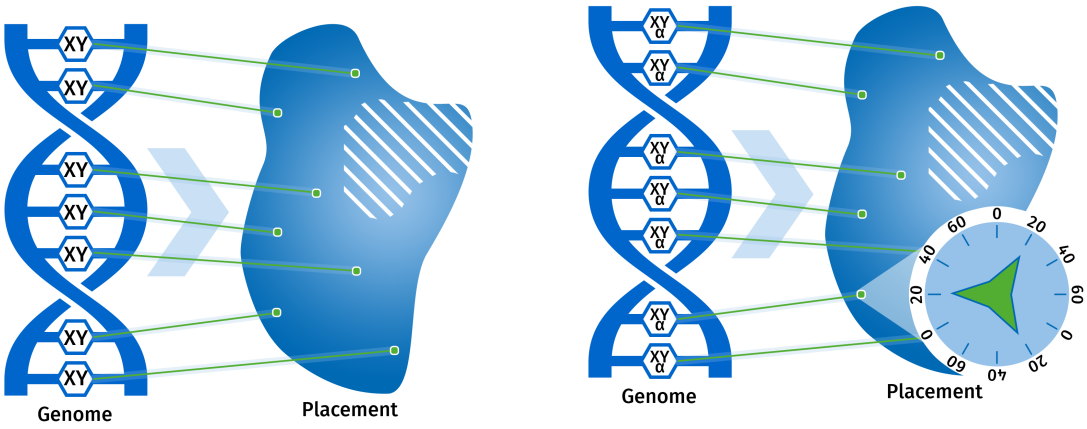
In this case we encode a layout L consisting of N_{gene} positions. Each position has an x-value x_i and an x-value y_i. In case we encode solutions for a special case of a three position parameter model[4], a position also contains a rotation value r_i. A chromosome γ is defined as the set of positions that are part of the layout (Figure 14a, Figure 14b).

$$x_i^c \in [x_{min}, x_{max}], y_i^c \in [y_{min}, y_{max}]$$
$$\gamma^c_{Offshore} = \{\{x_1^c, y_1^c\}, \{x_2^c, y_2^c\}, \cdots, \{x_n^c, y_n^c\}\}$$
$$\gamma^c_{Solar} = \{\{x_1^c, y_1^c, r_1^c\}, \{x_2^c, y_2^c, r_2^c\}, \cdots, \{x_n^c, y_n^c, r_n^c\}\}$$

4.2.2. Discrete Value Mapping

The layout can be encoded as a binary string by dividing the available area, into ρ^2 many cells, where ρ defines the density of the discretization. Each bit in the chromosome represents a cell and indicates whether a turbine is occupying it (Figure 15a,

[4]This algorithm is implemented to be capable to handle a simulation of a Solar Power Plant as well. A special case of that model requires a rotation value. Due to time shortage this model was not included.



(a) Continuous X and Y values only

(b) Continuous X,Y values and continuous rotation

Figure 14: The chromosome consists of a list of position tuples or triplets[4]. The length of the chromosome depends on the amount of turbines ($N_{gene} = N_T$).

Figure 15b). In consequence the resulting string has the length ρ^2. A continuous position can be mapped to a cell i as follows:

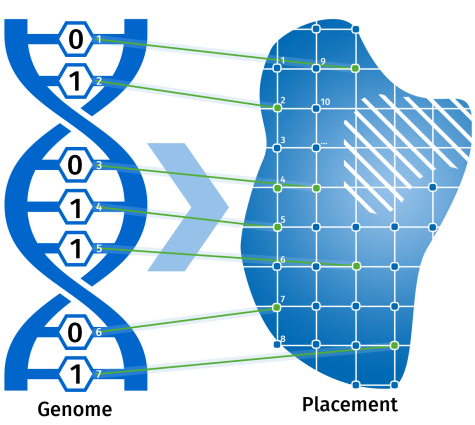
$$i = \left\lfloor \frac{x^c \cdot \rho}{x_{max}} \right\rfloor + \left\lfloor \frac{y^c \cdot \rho}{y_{max}} \right\rfloor \cdot \rho$$

Note that the algorithm ensures that there is always only one turbine placed within one cell.

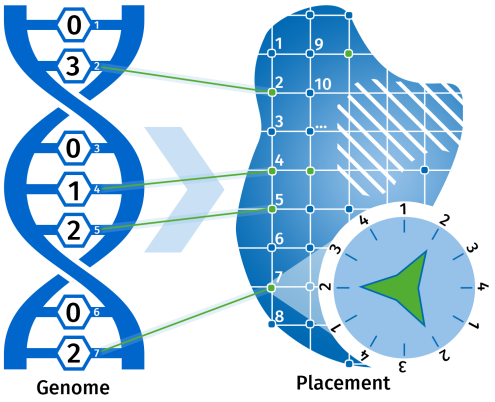
Example: Let $x_{min} = y_{min} = 0$ and $x_{max} = y_{max} = 1000$. Let us choose a resolution $\rho = 5$, resulting in 25 cells. The following positions are given:

$$p_1 = \{248; 183\}$$
$$p_2 = \{572; 301\}$$
$$p_3 = \{893; 520\}$$
$$p_4 = \{312; 621\}$$
$$p_5 = \{741; 952\}$$
$$\gamma^{Binary}_{Offshore} = \{\texttt{0,1,0,0,0, 0,0,1,0,0, 0,0,0,0,1, 0,1,0,0,0, 0,0,0,1,0}\}$$

This method of mapping is very intuitive, but also very inefficient. In case we increase the resolution in order to approximate the continuous mapping, the algorithm might exceed available memory resources: Using a resolution of $\rho = 10.000$ yields in a 100.000.000 character long chromosome, which, depending on the operating system can be up to 400 MegaByte in size. A typical population size of $N_{pop} = 1000$ causes



(a) List of boolean values indicating, if a turbine is placed in a cell.



(b) List of discrete values indicating, if and with which rotation a heliostat[4] is placed in a cell.

Figure 15: Discrete Value Mapping: A chromosome consists either of a list of boolean values, or a list of discrete values (e.g. `integer`). Each index in the list represents a cell in the available field. If a value at an index i is `true`, a Turbine is placed at cell i. If a value at index i is a positive discrete number r_i , a Heliostat[4] is placed in the center of cell i with the discretized rotation r_i.

the algorithm to require at least 400 GigaBytes of physical memory. In order to cope with these requirements a different mapping has to be used:

Similar to the continuous mapping we encode positions into the chromosome γ. Instead of using continuous values for a position, both x_i and y_i are discretized as follows:

$$x_i^d = a \cdot \frac{x_{max} - x_{min}}{\rho} + \frac{x_{max} - x_{min}}{2 \cdot \rho} \text{ with } a \in [0, \rho - 1] \wedge a, \rho \in \mathbb{N}$$

$$y_i^d = b \cdot \frac{y_{min} - y_{min}}{\rho} + \frac{y_{max} - y_{min}}{2 \cdot \rho} \text{ with } b \in [0, \rho - 1] \wedge b, \rho \in \mathbb{N}$$

$$r_i^d = c \cdot \frac{r_{max} - r_{min}}{\rho} + \frac{r_{max} - r_{min}}{2 \cdot \rho} \text{ with } c \in [0, \rho - 1] \wedge b, \rho \in \mathbb{N}$$

$$\gamma_{Offshore}^d = \{\{x_1^d, y_1^d\}, \{x_2^d, y_2^d\}, \cdots, \{x_n^d, y_n^d\}\}$$

$$\gamma_{Solar}^d = \{\{x_1^d, y_1^d, r_1^d\}, \{x_2^d, y_2^d, r_2^d\}, \cdots, \{x_n^d, y_n^d, r_n^d\}\}$$

We can approximate the size required for the different encoding. A chromosome consists of N_{gene} genes that each contain two `Double` parameters. There are N_{pop} many chromosomes in the population. For a 64-Bit operation system and $N_{pop} = 1000$, $N_{gene} = 50$, it follows:

$$\texttt{size(Double)} \cdot N_{pop} \cdot N_{gene} \cdot 2 = 800.000 \texttt{ Byte} \tag{29}$$

Employing the non-binary encoding results in a population that requires less physical memory by a factor of 500.000. In order to conserve resources, and to enable us to optimize complex simulations, we will rely on the latter encoding for DVCs.

4.3. Chromosome Factory

The genetic algorithm starts by obtaining a current population (*Current Generation Genome Pool*, Figure 13). The Chromosome Factory provides a strategy to create a set of random, but also sound, chromosomes. It is also possible to start the algorithm with a predefined set of chromosomes [10].

The genetic algorithm has to work with chromosomes containing viable solutions. All members of the population need to be evaluatable. In our set up, a chromosome is evaluable if it can serve as a template for a simulation. As introduced in Section 3.7, there are three basic conditions that must hold for each chromosome at any time. All contained positions must be within the limits of the simulation site, all contained positions must stay clear of restricted areas; and finally, all contained positions must keep a defined distance from each other. In order to create a chromosome that fulfills these conditions, random positions are added to the layout individually. After each insertion the validity of the layout is verified. In case the resulting layout is flawed the insertion is reverted and a new position is added. In order to be able to insert positions into dense layouts, a fall back method was implemented. In case too many attempts fail, a position and a direction is generated randomly, creating a sequence of steps, that iterate over the simulation site. The chromosome factory will try to add a position on this path until it reaches its originally generated position. In case this happens the algorithm assumes that the layout is too dense to add further positions and will terminate.

4.3.1. Differences between CVC and DVC creation

The method to resolve conflicts within a chromosome is identical for both value types. However the method to generate the position parameters themselves differ. A position for a CVC is randomly generated as a tuple of two random parameters that are within the limits of the simulation site. A position for a DVC is computed as follows: We divide the simulation site into $\rho \times \rho$ cells. There are $N_{position} = \rho^2$ many cells. A new cell index is randomly picked $r_i \in [0, ..., N_{position-1}], r_i \in \mathbb{N}$. Next we can convert the cell index r_i into a tuple (x_i, y_i).

$$
\begin{aligned}
x_i &= \underbrace{\frac{|x_{max} - x_{min}|}{\rho}}_{\text{horizontal cellsize}} \cdot \underbrace{\frac{r_i - (r_i mod\ \rho)}{\rho}}_{\text{row}} + \underbrace{\frac{|X_{max} - X_{min}|}{2 \cdot \rho}}_{\text{horizontal midpoint}} \\
y_i &= \underbrace{\frac{|y_{max} - y_{min}|}{\rho}}_{\text{vertical cellsize}} \cdot \underbrace{(r_i mod\ \rho)}_{\text{column}} + \underbrace{\frac{|Y_{max} - Y_{min}|}{2 \cdot \rho}}_{\text{vertical midpoint}}
\end{aligned}
\tag{30}
$$

4.4. Stopping Criteria

The *Stopping Criteria* defines a global objective that has to be reached by the algorithm to terminate. A common stopping criteria is the achievement of a certain fitness value. In case a chromosome with the desired quality could be evolved, the algorithm terminates. There is always the possibility that the GA will not reach the optimum under the current configuration, or the fitness of the global optimum is simply unknown. Therefore it is common to add a second criteria that terminates the algorithm after a defined amount of iterations. There are further, more complex criteria, that are able to determine whether the algorithm has stagnated, or whether no further progress is possible. In our set-up, we use a combination of the first two criteria (maximum amount of generations and achievement of fitness) mentioned above.

The genetic algorithm, used in our simulation set-up, terminates the algorithm if a fitness of 1 is reached or if more than 2000 generations were created.

4.5. Selection Operation

At the beginning of an iteration each chromosome is evaluated and ranked within the population. The result of this evaluation is called *fitness*. The evaluation of a chromosome is based on the *objective function* of the optimizer, and is often determined by simulation of a specific optimization problem. Each chromosome represents a possible solution for a specific problem. Thus its fitness provides a measurement how well this solution compares to other individuals. Once a ranking has been established a strategy is used to select chromosomes for the next generation. The selected chromosomes will be used for recombination (crossover operation). In order to enhance diversity among the selected chromosomes, it is recommended, not only to select chromosomes with a high fitness, but also to select a few 'bad' chromosomes. The diversity is needed to break free from local maxima of the objective function and reach the global one (*Selection Pool*, Figure 13). *Elitism* can be used at this point to prevent degradation of the overall population fitness. Elitism is an operation where the chromosomes with the highest fitness, the elite, is copied directly, without any modification, to the next generation. By doing so it is ensured that the best chromosomes of the current generation are at least as good as the best chromosomes of the previous one.

The selection operation provides the Elite-Pool and the Selection Pool. Depending on the selection method, the Elite-Pool can be part of the Selection Pool aswell. In our case the common selection operation *roulette wheel selection* [13] is used. Chromosomes are picked at random, but the probability to be picked is depending on the fitness of the chromosome. If f_i is the fitness of chromosome i then its probability to being selected is

$$p_{s,i} = \frac{f_i}{\Sigma_{j=1}^{N} f_i}$$

The selection operation handles the chromosomes as a whole on a population scale.

Therefore there is no distinction between the selection of CVCs and DVCs. Both set-ups use the identical implementation at this point.

4.6. Coupling Operation

This operation provides a strategy to form pairs among the selected chromosomes (*Coupling Pool*, Figure 13). Note that one chromosome can be coupled with multiple other chromosomes. The pairs can be formed at random, or with consideration of chromosome fitness. Genetic information of chromosomes with a high fitness can be promoted by coupling a few very good chromosomes with the rest of the selection pool. Other operators match each chromosome only a single time, forming $N_{sel}/2$ couples to produce offspring. We implemented a simple random coupling, that assigns chromosome pairs randomly. The results of this operation are therefore entirely dependent on the compilation of the selection pool. We chose this operation in order to minimize the configuration parameters we have to control.
Similar to the selection operation, the coupling operation manages a part of the population independently from the chromosome encoding, therefore there are no differences between the management of CVCs of DVCs.

4.7. Crossover Operation

Provided with chromosome pairs, this operation will use a strategy to blend each pair to a new chromosome. This process is also often referred to as *recombination*. The genes of both chromosomes can be swapped, or combined to form new genes. The resulting chromosome has to be influenced by both input chromosomes and has to be a viable solution that can be evaluated. We refer to the resulting chromosome set as *offspring* (*Offspring Pool*, Figure 13). Common methods for the crossover operation are *One-point-crossover*, *n-point-crossover* and *Uniform Crossover* [7][13]. These operations assume that the genes are stored in an ordered sequence within the chromosome. One-point-crossover randomly determines an index within that sequence (Figure 16). The first part of the resulting offspring chromosome consists only of genes from one parent, the other part consists of genes from the other parent. N-Point-crossover behaves similarly. Instead of one index, n indexes are chosen randomly. At each index the parent that provides genomes for the offspring chromosome is switched. A third method is the Uniform Crossover Operation. Here, at each index of the offspring chromosome the parent determining the gene at this location is picked at random using a uniform distribution.

Because of the used encoding (Section 4.2), the genomes in the chromosomes are not ordered. For that reason, we cannot simply implement the classical crossover operations mentioned above. It would be possible to run the classical operations on the binary encoded version of the discrete chromosomes, but we would not be able to compare both algorithms as they use different operators and we would not be able to run large simulations, as explained in Section 4.2. Instead we implement a crossover operation, that is based on *One-Step-Crossover* [24]. Richter et. al. faced the same

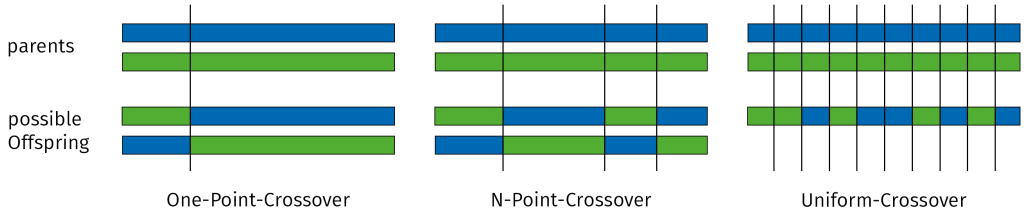


Figure 16: Typical crossover operations: *One-Point-Crossover* 'flips' both chromosomes at one point. *N-Point-Crossover* 'flips' the genetic information at multiple points. *Uniform-Crossover* picks at random from which parent to choose the genetic information.

problem, when performing a crossover operation. When merging two chromosomes with unordered genomes, we need to ensure that the resulting chromosome satisfies the constrains of the simulation, such as minimum distance to neighboring positions.

Figure 17 depicts the implemented crossover operation. First the positions of both parents (*layout parent A* and *layout parent B*) are merged into one *combined layout*. It is expected to find conflicts (e.g. positions are too close to each other) within the new layout. Conflicts are resolved by removing the position assigned to the layout of parent B. Because the simulation provides data on individual genes, and their contribution to the overall chromosome fitness, we are able to evaluate single positions, after resolving all conflicts. Once all positions are ranked, the N_{gene} best positions form the new layout (*reduced layout*).

The crossover operation creates new chromosomes and works directly with the values within these chromosomes. However, as will be demonstrated in Section 5.3, the crossover operation does not modify any values. Instead it simply recombines genes to new chromosomes. Values that have been formed before either as discrete parameters or continuous ones, will remain either discrete or continuous after the operation. We use the same operator for both DVCs and CVCs.

4.8. Choosing Operation

In order to introduce new aspects to the already existing solutions, some offspring is modified by the mutation operation. Because the mutation can also worsen good solutions, it is not applied to all chromosomes. The choosing operation provides a strategy, to select which chromosomes are mutated and which are not. There are different operations which prefer low fitness chromosomes to be mutated, or which apply the principles of *roulette-wheel-selection* at this point. For simplicity, we chose an operation in our set up, picking chromosomes at random. For each offspring chromosome there is a chance p_c to be chosen for mutation.
The choosing operation picks chromosomes without modifying or replacing any gene values. For both CVCs and DVCs the same operator is used.

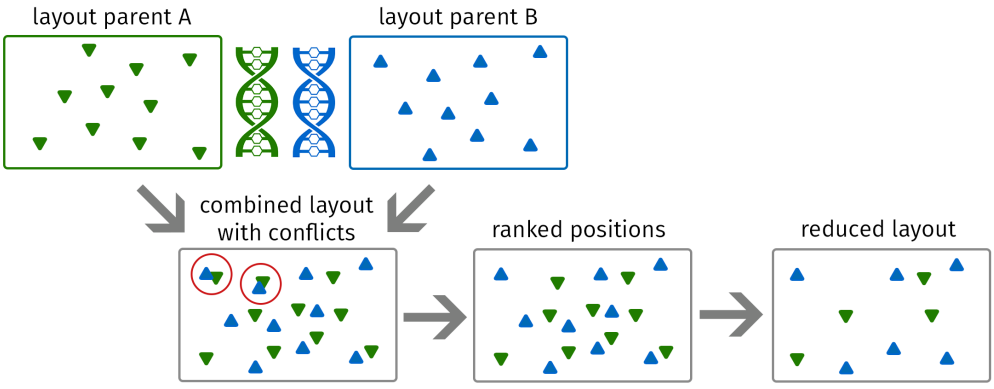


Figure 17: Implemented *One-Step-Crossover* Operation. Layouts of parent A and parent B are combined. All conflicts resulting from this union are resolved. Next all turbines are ranked individually. The amount of positions per chromosome is limited therefore only the N_{gene} best positions form the final output. One-Step crossover adds an evaluation step to the operator. Although this adds runtime to the operation, the overall runtime of the algorithm is lowered (see Section 5.3).

4.9. Mutation Operation

In order to amplify diversity, some of the offspring are chosen at random and are mutated. The mutation ideally will change small parts of the chromosome, leaving the majority of the genetic information in its original state.

At first, the n_m positions p_i within the chromosome that are about to be altered have to be determined. Where n_m is picked at random such that $n_{min} \leq n_m \leq n_{max}$. Each position $p_1 \cdots p_{n_m}$ is evaluated individually where $f(p_i) \in [0, 1]$. The positions within the chromosome are sorted by their fitness. The chance for a position p_i at index j to be picked is determined as follows: Let r' be a random variable where $r' \in [0, 1]$.

$$p(j) = \lfloor \frac{r'^2}{2} + \frac{r'^4}{2} \rfloor$$

A new index will be picked in case the resulting one was already chosen before. It is more likely for a position with a lower fitness to be picked than one which is among the positions with the highest fitness. This method is used to increase the likelihood of mutations, improving the fitness of a chromosome. The chances for improvement are higher when altering bad positions within the chromosome. Changes in good positions could even result in a deterioration of already achieved fitness levels.

The mutation itself generates a random position $p_{rand} = \{x_{rand}, y_{rand}\}$ and computes the mutated position p_{mut} between both positions. The value δ_{min} represents the minimal distance from the old value. The variable r is picked at random and is either 0 or 1. It is used to determine the direction in which a value is altered. The parameter

μ represents the mutation rate.

$$
\begin{aligned}
\delta_{min_x} &= \frac{x_{max} - x_{min}}{2\rho} \\
\delta(r) &= \begin{cases} Random(x_{old} + \delta_{min_x}, x_{max}), & \text{if } r > 0 \\ Random(x_{min}, x_{old} - \delta_{min_x}), & \text{if } r \leq 0 \end{cases} \\
x_{mut} &= ((x_{old} + \delta(r)_{min_x} \cdot r) \cdot (1 - m) + x_{rand} \cdot \mu) \\
\delta_{min_y} &= \frac{y_{max} - y_{min}}{2\rho} \\
\delta(r) &= \begin{cases} Random(y_{old} + \delta_{min_y}, y_{max}), & \text{if } r > 0 \\ Random(y_{min}, y_{old} - \delta_{min_y}), & \text{if } r \leq 0 \end{cases} \\
y_{mut} &= ((y_{old} + \delta(r)_{min_y} \cdot r) \cdot (1 - m) + y_{rand} \cdot \mu)
\end{aligned}
\tag{31}
$$

We aim to keep both algorithm set-ups as close as possible. Therefore both operators use the same computations to determine X and Y values. When using the discrete value setup, the algorithm uses a post processing step to move each generated value to the closest discrete position. We compute the cell index c as follows:

$$
c = \left\lfloor \frac{x_i \cdot \rho}{\frac{|X_{max} - X_{min}|}{\rho}} \right\rfloor + \left\lfloor \frac{y_i \cdot \rho}{\frac{|Y_{max} - Y_{min}|}{\rho}} \right\rfloor \cdot \rho \tag{32}
$$

Using c as input for Equation 30 provides us with a discrete position. The parameter δ_{min} ensures that the mutation of a single value is large enough to result in a different discrete value after this post processing step. We ensure that no cell is occupied by two turbines at any time.

4.10. Replacement and Replenish Operation

The replacement operation forms the new population Γ_{new} for the next generation. The N_{repl} best chromosomes of the offspring population (mutated and not mutated set) are added to the current population. The worst chromosome is removed until the population size matches $N'_{pop} = N_{pop} - (N_{rand} + N_{repl})$. N_{rand} represents the number of randomly generated chromosomes. Randomly generated γ'_i chromosomes are added until the population size reaches N_{pop}.

$$
\begin{aligned}
\Gamma_t &= \{\gamma_1, \gamma_2, \cdots, \gamma_{N_{pop}}\} \\
\Gamma_t \cup \Gamma_{\text{offspring}} &= \{\underbrace{\gamma_1, \gamma_2, \cdots, \gamma_{N'_{pop}}, \cdots, \gamma_{N_{pop}}, \cdots, \gamma_{(N_{pop}+N_{repl})}}_{\text{sorted by fitness}}\} \\
\Gamma_{t+1} &= \Gamma_{rand} \cup \{\gamma_1, \gamma_2, \cdots, \gamma_{N'_{pop}}\} \\
\Gamma_{t+1} &= \{\gamma_1, \gamma_2, \cdots, \gamma_{N'_{pop}}, \gamma'_1, \cdots, \gamma'_{N_{rand}}\} \\
|\Gamma_{t+1}| &= N_{pop}
\end{aligned}
\tag{33}
$$

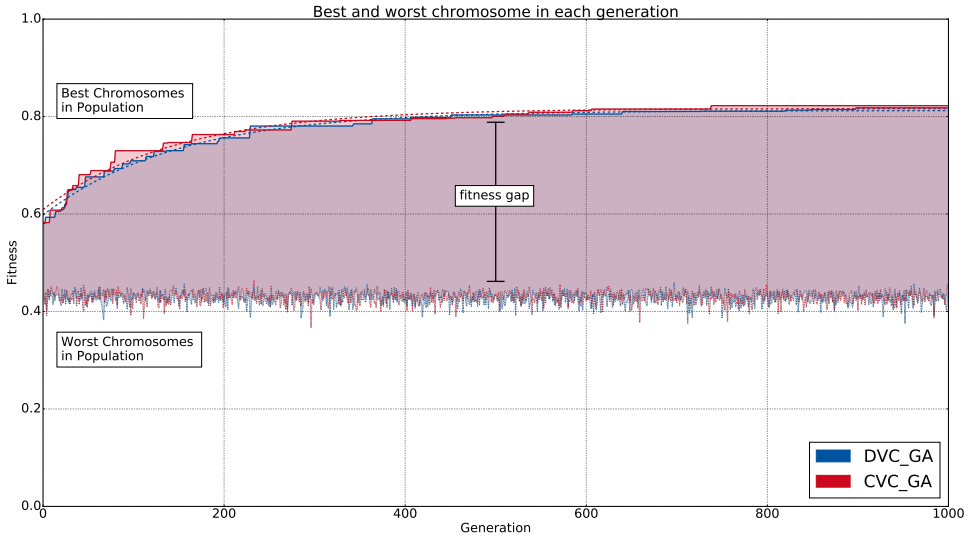


Figure 18: Maximum and minimum fitness values, that are present in a population in each generation. We call the difference between the worst and the best chromosome fitness gap.

In order to maintain a sufficient variety random chromosomes have to be introduced to the population after the elimination of the individuals with the lowest fitness. Since they did not undergo any evolution process these random chromosomes are very likely to have a fitness below the population average. With increasing number of generations the *fitness gap* between random chromosomes and the population average increases (Figure 18). Thus the chance of a random chromosome to remain within the population is very low, as they are likely to be replaced by offspring with a higher fitness. By adding the chromosomes after the elimination process there is a chance that the chromosome will be randomly selected for the mating pool, thus increasing diversity in the population.

The replacement operation itself does not differ between its operator for CVCs and for DVCs. However, the replacement operation includes a chromosome factory to generate new random chromosomes. As explained above, the chromosome factory itself differs greatly for both chromosome types.

4.11. Summary

In this section we went through one iteration of the genetic algorithm, highlighting each major operator of it. Although working on different value types, many operators are the same for both continuous value chromosomes and discrete value chromosomes. In those cases, where different operators had to be implemented (*Chromosome Factory, Crossover* and *Mutation*), attempts were made to keep the differences as small as possible.

After presenting the model, and the algorithm we use to optimize it, we will analyze in the next section, what effects changes in discretization settings and and optimizer configuration have upon the optimization results.

5. Convergence

Evolution is based on diversity. Natural selection uses it to produce new and better solutions for a given problem [18]. An evolutionary algorithm converges when the diversity of its population breaks down and most of its individuals become similar [13]. In the following sections we will analyze the operators and evaluate the effect they have on the diversity of the population and how their convergence changes upon parameter modification. Not every operator can be configured with a parameter. Both choosing operation and coupling operation are based on random selection and will therefore not be analyzed in this section. Although the stopping criteria is one of the fundamental building blocks of a genetic algorithm, it will not be covered by this section, as it has no influence on the convergence itself. This leaves us with an analysis of the following operators:

1. **Chromosome Factory**: We take a look on the effect of discretization on the precision and the search space of the optimization.
2. **Selection Operation**: We present the effects of different selection rates on population diversity
3. **Crossover Operation**: We verify the findings by Richter[24] upon which we chose this operator.
4. **Mutation Operation**: We analyze the effects of different mutation settings.
5. **Replacement Operation**: We analyze the effect of different replacement rates.

Finally we will use the findings to propose two configurations for the chromosome types of the genetic algorithm. If not displayed otherwise the default settings from Table 4 where used for any plots in this chapter.

The research of Goldberg [13] and Luis [18] to the topic of predictions on convergence behavior is based on genetic algorithms, which use ordered sequences of bits as chromosomes. They propose that a diverse population of a genetic algorithm converges towards a population in which all individuals become similar. One approach to predict convergence is to inspect the influence of each genetic operator on the diversity of the next generation. Gale uses [28] the average hamming distance over all chromosomes to measure the diversity of a population. We cannot use the same method to measure the diversity of our population, as our chromosomes neither consist of binary sequences, nor are they ordered. We will compute an indicator for diversity by computing the distances between the positions p_k from each chromosome i, j to another.

$$\Delta = \frac{1}{N_{pop} N_{gene}} \sum_{i \neq j} \sum_{k} |x_{i,k} - x_{j,k}| + |y_{i,k} - y_{j,k}| \tag{34}$$

The correlation between convergence and diversity is depicted in Figure 19. We can see that the fitness of the population converges as its diversity breaks down. In

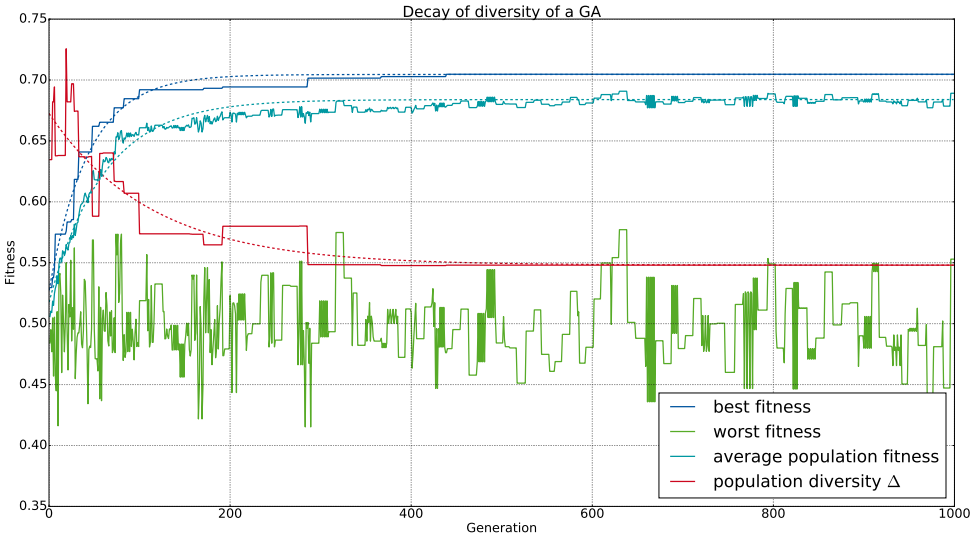


Figure 19: Convergence of a genetic algorithm using discrete value chromosomes. Settings for a low runtime (Table 5) with a reduced population size $N_{pop} = 10$ were used. Note that the population diversity is subject to a constant noise due to the injection of random chromosomes, although this noise becomes nearly negligible in the latter generations.

the early generations $(0 - 200)$ the average chromosome variety seems to change a lot within few iterations. This behavior is caused by changes in the order of the chromosomes. An encoding of one chromosome with N_{gene} genes can represent the same layout in $N_{gene}!$ different ways. Note that each layout is only represented once in the population. The diversity is expected not to converge towards zero as mutation and replacement operations keep adding new chromosomes.

5.1. Effects of discretization on convergence

Discretization of the genome parameters has multiple effects on behavior how the algorithm converges towards an optimum. In the context of this optimization we will take a look at two different aspects that are influenced by discretization: *Search-space* and *precision*.

5.1.1. Search-Space

The search-space A, also known as choice set, is the domain of the function that is optimized. We shall take a look, what effect discretization has on the count of its members $|A|$. When using a computer we have to rely on machine numbers in order to represent genome values of CVCs. In our implementation we rely on the IEEE754

`double` standard [2]. A `double` is represented as follows:

$$x = (-1)^{sign} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023} \tag{35}$$

We can compute the search-space for a genome parameter, such as the X-Value as follows: One approach is a simple count of all possible bits:

$$|D| = 2^{64} \tag{36}$$

It is impossible to represent more numbers with the same amount of bits, but we have to consider representations such as infinity and `NaN` as well:

1. There are two representations of zero: One with a positive and one with a negative sign.
2. There are two infinities, where the first 12 bits are `0x7ff` or `0xfff` and the mantissa is all zero.
3. There is a set of `NaN` values, encoded with sign+exponent bits `0x7ff` or `0xfff` and a nonzero mantissa.

The amount of distinct values that can be represented by a `double` is

$$|D| = 2 \cdot 2^{11} - 1 \cdot 2^{52} - 1 = 18.437.736.874.454.810.623 \approx 1,8438 \times 10^{19}$$

In addition we have to consider the range of the genome parameters. We will not be able to use the entire domain of the `double`-values. We can only use `double`-values that are within $X_{max} - X_{min}$. We compute the amount of distinct `double`-values for three parameter ranges that are common in our set up.

$$\begin{aligned} 1000_{10} &= \texttt{01000100 01111010 00000000 [...] 00000000}_{\texttt{double}} \\ e &= \texttt{10001000}_2 \\ m &= \texttt{11110100 00000000 [...] 00000000}_2 \end{aligned}$$

We compute the amount of possible values, by computing the amount of remaining values that result in a smaller number than 1000:

$$|D|_{1000} = (e-1) \cdot 52 + m \approx 4.4824 \times 10^{16}$$

Analogous we compute for $X_{max} - X_{min} = 5000$:

$$\begin{aligned} e &= \texttt{10001011}_2 \\ m &= \texttt{01000101 10011100 01000000 [...] 00000000}_2 \\ |D|_{5000} &= (e-1) \cdot 52 + m \approx 5.5037 \times 10^{16} \end{aligned}$$

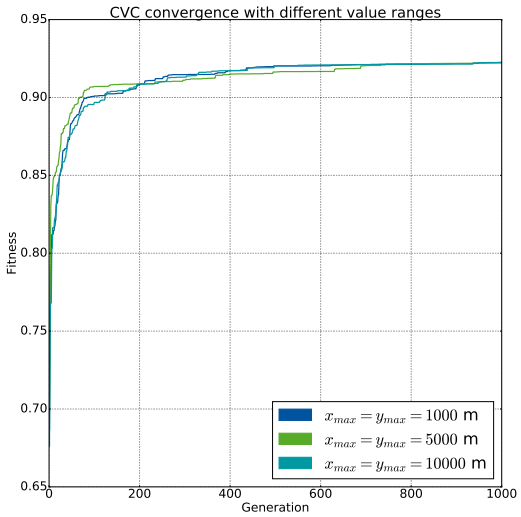


Figure 20: CVC GA convergence behavior. The search space dimensions do not change a lot when changing the Genome value limits between 1000 and 10.000. Thus, the algorithms within this range do not differ a lot.

We compute for $X_{max} - X_{min} = 10000$:

$$\begin{aligned} e &= \texttt{10001100}_2 \\ m &= \texttt{01000101 10011100 01000000 [...] 00000000}_2 \\ |D|_{10000} &= (e-1) \cdot 52 + m \approx 5.940 \times 10^{16} \end{aligned}$$

As presented in Figure 20, the differences in the used ranges result in negligible differences in the convergence. For simplicity let us assume that the approximate domain for a `double` value in a genome has $|D| = 5 \times 10^{16}$ elements.

As for the search-space of DVCs it is important to note that a genome value can take on ρ-many values. In contrast to the CVC-encoding this amount is independent on the range of the values $(X_{max} - X_{min})$. For DVSs there are $|D| = \rho$ many distinct values for each parameter in a genome.

After this determination of how many values a parameter can take on, we can compute the amount of possible chromosomes, and thus predict the size of the search-space the genetic algorithm is operating on. A chromosome γ encodes a layout L of N_{gene} positions. Each position is represented by a genome, that is encoded by either a tuple (x, y) or a triplet (x, y, r). We compute:

$$|A| = N_{gene} \cdot |D|^a \tag{37}$$

Where $a = 2$ in case we encode tuples and $a = 3$ in case we encode triplets. We can compute some examples in order to compare search-space dimensions. With $N_{gene} = 50$

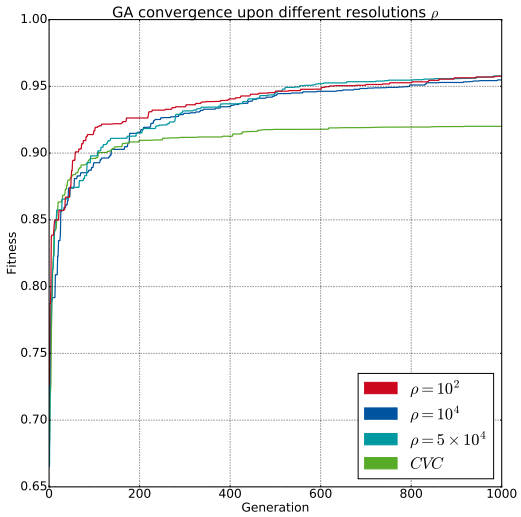


Figure 21: A GA using DVC can converge faster than a GA using CVC with the same settings. A higher resolution causes a greater search space, which slows the performance of the algorithm down.

and $a = 2$ we get:

$$|A|_{CVC} = 50 \cdot (5 \times 10^{16})^2 = 1,25 \times 10^{35} \tag{38}$$

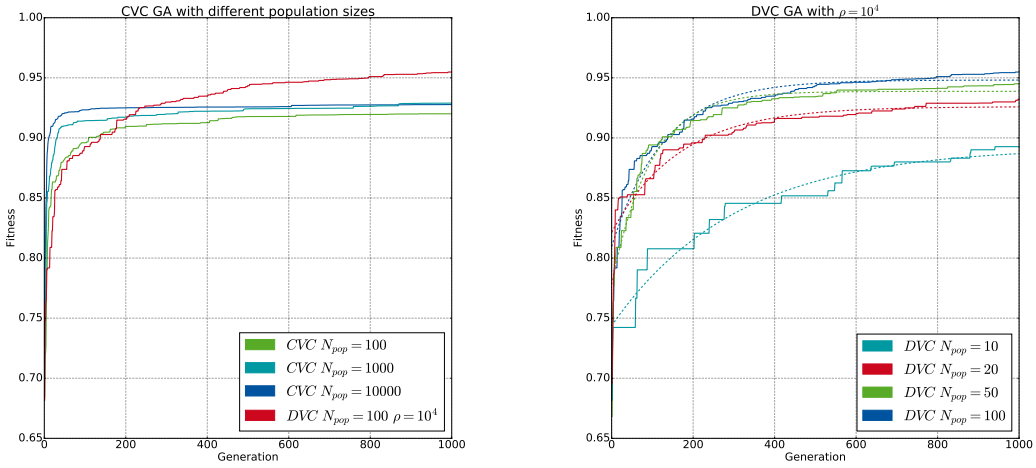
$$\text{with } \rho = 100 : |A|_{DVC} = 50 \cdot (10^2)^2 = 5 \times 10^5 \tag{39}$$

$$\text{with } \rho = 10^5 : |A|_{DVC} = 50 \cdot (10^4)^2 = 5 \times 10^9 \tag{40}$$

$$\text{with } \rho = 5 \times 10^5 : |A|_{DVC} = 50 \cdot (5 \times 10^4)^2 = 1,25 \times 10^{11} \tag{41}$$

We expect the GA to converge faster when it is exposed to a smaller search space. Figure 21 displays the fitness of the highest ranked chromosome in each generation. It is notable that the configurations for DVCs produce fitter chromosomes in the same amount of iterations. In contrast to our first observation, simply lowering the search-space size by discretization of genome values will not always improve performance. Note that the curves of the three DVC-GAs cross at multiple points. In Section 5.1.2 we will take a look at the influence of discretization on the precision of our model. This might explain why a smaller ρ not necessarily leads to a higher fitness, in a smaller amount of generations. In this thesis we introduce GAs as a search algorithm. Until now we tried to improve the search space of the algorithm by reducing the amount of possible distinct values for each parameter in a genome. We should also be able to improve the performance by increasing the sample size. The current population of a GA is the portion of the search-space the algorithm works on in parallel. By increasing its size, the drawback of big choice-set size should be compensable (Figure 22a). Doing the opposite should cause the DVCs convergence behavior to match the CVCs at a higher population size (Figure 22b).

Augmenting the population size N_{pop}, provides a higher growth in fitness per iteration,



(a) Differences in search-space sizes can be compensated in part by population size. Note that we also need to readjust all operators to have a similar effect on the larger population.

(b) By lowering the population size we can undo the benefit of discretization. A population consisting of 20 DVCs with $\rho = 10^4$ converges similarly to a population of 1000 CVCs

Figure 22: Effects of diversity on convergence

but at the same time the computation costs rise linearly with N_{pop} (Figure 23). Thus the gain in performance comes at the cost of runtime and computer resources.

5.1.2. Precision

When we increase the resolution ρ, the amount of available positions increases as well. With a greater variety of positions to choose from, we gain more precision in our placement of either wind turbines or heliostats. We can compute the precision of the current DVC configuration as follows:

$$\Delta x = \pm\frac{x_{max} - x_{min}}{\rho} \quad \Delta y = \pm\frac{y_{max} - y_{min}}{\rho} \tag{42}$$

Example: With $\rho = 10^4$ and $X_{max} - X_{min} = Y_{max} - Y_{min} = 5000\ m$, each position will be placed in an area of 0.5 m by 0.5 m (Figure 24a). With $\rho = 10$ and $X_{max} - X_{min} = Y_{max} - Y_{min} = 5000\ m$, each position will be placed in an area of 500 m by 500 m (Figure 24b). The objective function `TRIANGLE` will the configuration with the higher resolution above the one with a low resolution (Figure 25).

The amount of possible values is reduced by lowering the resolution. The lower the resolution, the more likely a value is unable to reach its global optimum. Reducing the resolution causes on average the values to drift away from their optimal positions. Lower resolution tend to cause layouts with a lower quality (Figure 25).

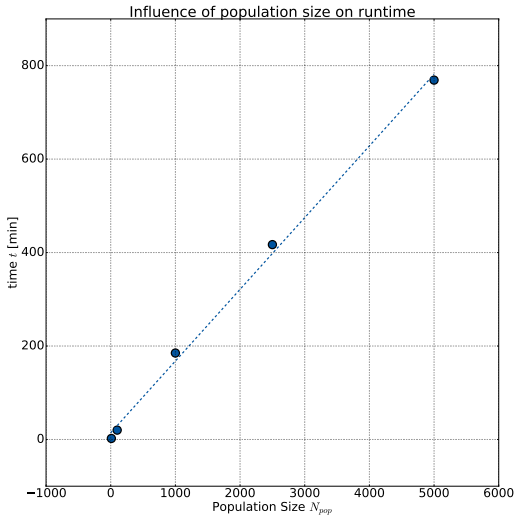
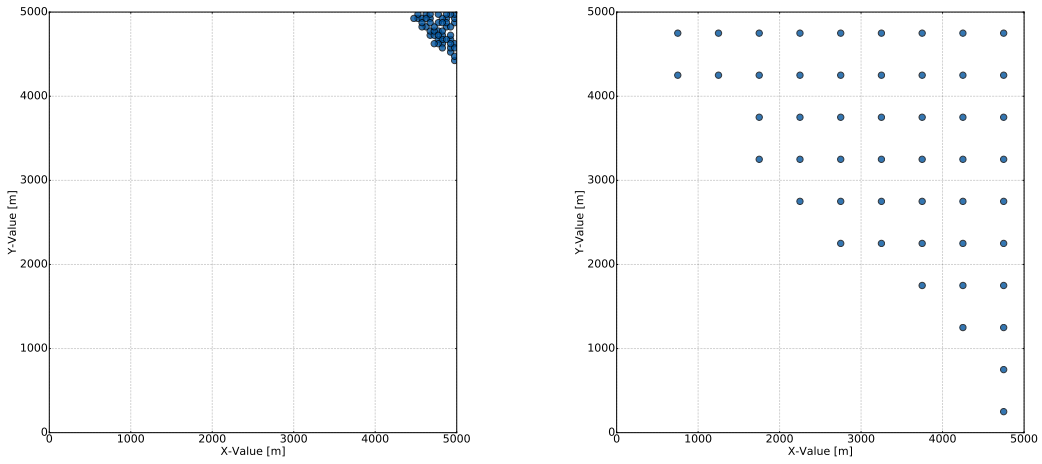


Figure 23: Runtime is linearly dependent on the population size N_{pop}. This limits our attempts within this thesis to population sizes between 2 and 10000.



(a) Optimized layout with DVCs, $\rho = 10^4$, for objective function `TRIANGLE`

(b) Optimized layout with DVCs, $\rho = 10$,for objective function `TRIANGLE`

Figure 24: Effects on ρ of placement. It is important to pick a resolution that is precise enough to reach an optimal placement. While both algorithms optimize for the same function, the algorithm on the left (Fig. 24a) reaches a fitness $> 90\%$. The plot on the right (Fig. 24b) is only a little bit above the fitness of a randomly generated chromosome (70%).

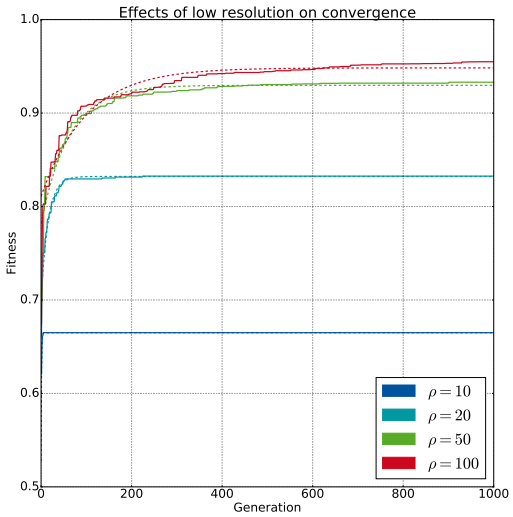
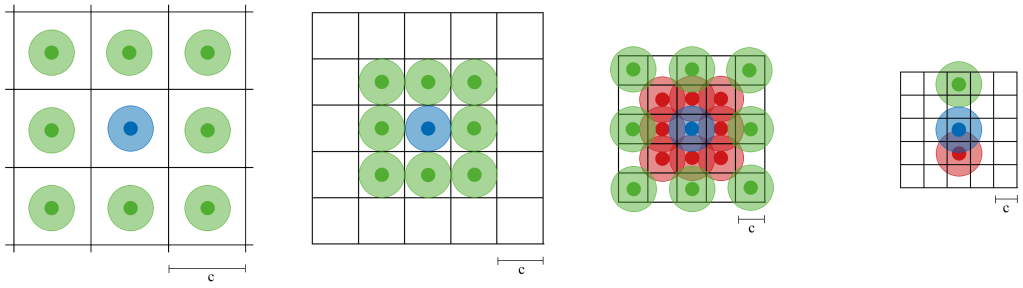


Figure 25: Genetic algorithms running on DVCs with a resolution that is too low will not converge towards the global maximum, as the optimal configuration can not be resolved.

Following the findings, the algorithm faces an improvement conflict. While on the one hand a reduction of the search-space leads to improvements for the progress of the algorithm, it also causes the configuration to be too inaccurate, and due to its placement will prevent the algorithm from reaching a global optimum.

It is important to find a good balance between search-space size and precision. We will now take a closer look at resolutions that are close to the precision of the simulation we are working with. Due to the placement restriction, each position must be placed at least R meters away from the closest position. This restriction effects the expected precision as displayed in Figure 26. In case we choose a resolution that results in a greater cell size c than the restricted area around each position, we can configure the chromosome to any set of positions without risking any position conflicts (Figure 26a). The same applies to Figure 26b, note that this is the configuration with the highest precision, where still no conflicts are expected. An additional resolution increase could result in blank cells where we can not place a position within the fields closest to a neighboring position, one cell has to be left blank. The resulting smallest distance with the higher precision can be greater than with a lower precision (Figure 26c). Further increment of the resolution minimizes this problem (Figure 26d).
While the section above only considered the precision of DVCs we should also consider the precision of CVCs. For CVCs we can refer to the *machine epsilon* in order to get an upper bound on the precision of CVC configurations. In case of a `double` value the maximum rounding error is $2^{-53} \approx 1.1102 \times 10^{-16}$ [12]. In the context of the offshore wind park model, this precision would hypothetically allow us to place a wind turbine with a precision of a tenth of a femto meter. The size of one proton amounts to $1.6fm$.



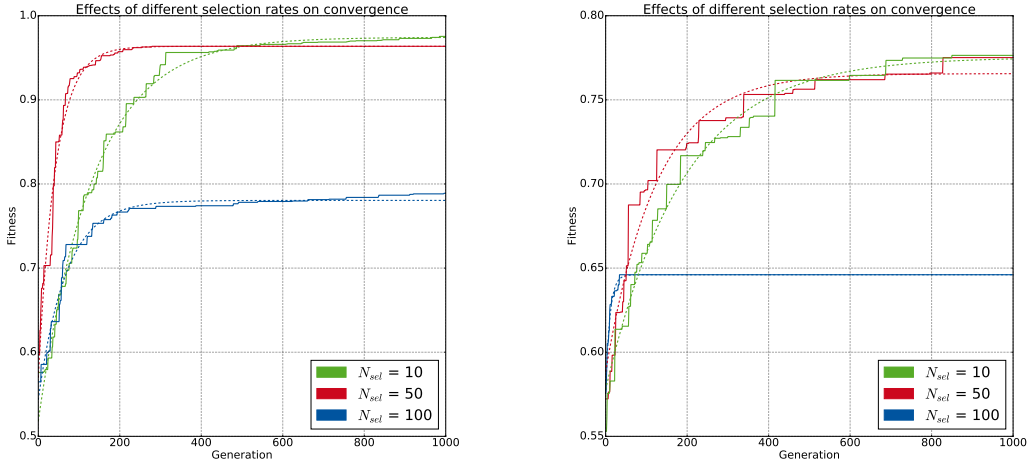
(a) $c > 2R$ (b) $c = 2R$ (c) $c < 2R$ (d) $c < R$

Figure 26: Effect of R on ρ: Placement restrictions and resolution interact. Figure 26b permits a more dense placement than in Figure 26c, although Figure 26c employs a higher resolution. Higher resolutions will allow denser placement again (Figure 26d).

Obviously this immense precision is not required. Neither the simulation is sufficiently accurate, nor is it possible to place any construction with such precision. The CVC algorithm is optimizing with an accuracy that is several magnitudes too large.
One side effect is the risk of diversity decline in the population. In case two chromosomes differ in one position by a femto meter it will be recognized as a different chromosome although the fitness function will not be able to compute a difference. This can lead to a monotone population that is unable to converge towards a global maximum. Default mechanisms to prevent duplicates will not be able to avert this unlikely scenario, as the population consists of seemingly different chromosomes.

5.2. Effects of Selection Operation on Population Diversity

Classical selection operations reduce diversity of the population with each iteration [18]. The amount of this reduction depends on the offspring that is generated by the selected individuals. Choosing a high selection rate will still result in a convergence. Although there is no promotion of any genetic material based on its fitness, there is still a random genetic drift [18][28], that minimizes diversity. Selecting the entire population to provide new offspring slows down the convergence rate (Figure 27a, blue line). Choosing a low selection rate (Figure 27a, blue line) will cause a rapid convergence, as the next generation will only be based on a small elite part of the current generation. A high fitness is reached in a few generations. This can be also observed for CVCs (Figure 27b, blue lines). Although starting with the highest growth rate the algorithm with the configuration with the highest selection rate is soon trapped in a local maximum. Choosing a low selection rate (Figure 27a, Figure 27b, green lines) provides best results for both chromosome types and is recommended.



(a) Convergence of a genetic algorithm using DVCs.

(b) Convergence of a genetic algorithm using CVCs.

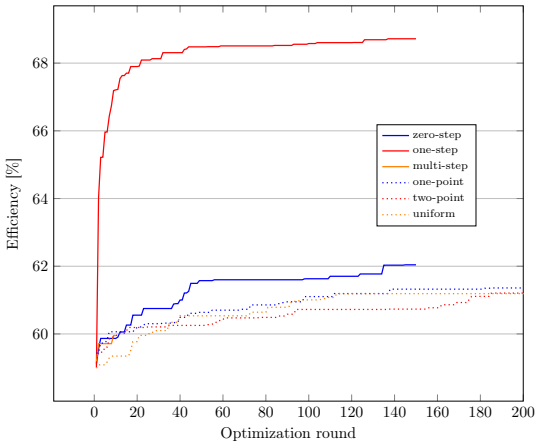
Figure 27: Effect of selection rate on convergence. Selecting a small portion of the population will result in a slow convergence, that will reach the highest values. Selecting half of the population will result in a fast convergence, but will converge towards a lower fitness. Selecting all chromosomes will converge fastest, but will also result in the lowest fitness.
Settings for a low runtime (Table 5) were used.

5.3. Effects of Crossover Operation on Convergence Behavior

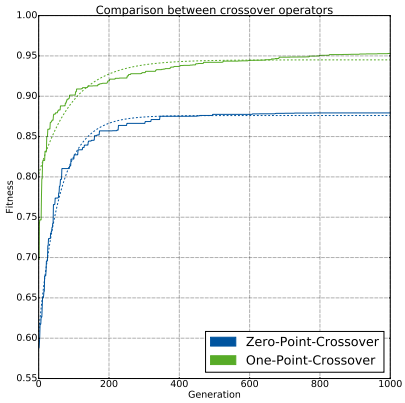
Analogous to the proof in Louis [18], the crossover operators 1-Point-Crossover and N-Point-Crossover do not have any effect on the diversity of a population, because these operators only change the order in which the genomes contribute to the overall diversity. The crossover operator used in our implementation is an adaptation of the *One-Step-Crossover* as presented in [24]. As Richter points out, the operator was modified in order to compute crossover for the specific genotype representation and adapted to prefer good genes in its offspring, thus improving the overall fitness of the population. The effect is illustrated in Figure 28. We are able to reproduce Richter's findings within our own implementation. This preferential treatment of alleles reduces the average diversity of all individuals to a greater extend than classical crossover operations such as 2-point-crossover. By passing on dominant genes to the offspring, it is likely to hand down a specific gene to multiple offspring, thus multiplying its presence in the population and reducing the average diversity.

5.4. Effects of Mutation Operation on Convergence Behavior

The mutation operation is a constant source for diversity in the population. Mutation is one of the few operations that will introduce new alleles into the population [28].



(a) Classical Crossover Operations in comparison. Experimental results from [24]



(b) Comparison of Zero-Point-Crossover and One-Point-Crossover within our set up.

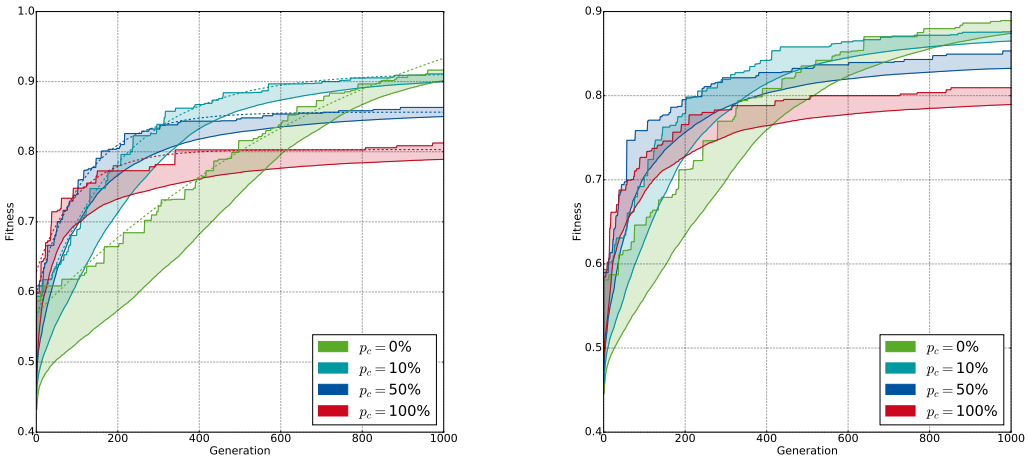
Figure 28: Performance of *One-Step-Crossover* in comparison to classical crossover operations and *Zero-Step-Crossover* as presented by Richter [24]. We are able to confirm his findings (Fig. 28a), by reproducing the measurements (Fig. 28b)

By modifying existing chromosomes it counteracts genetic drift. In case the mutation operation is set up too strong it can destroy alleles that had a positive impact on specific chromosomes and lower the overall quality of a population. If the mutation is too weak, it will slow down the convergence of the algorithm, and cause the algorithm to require many iterations to evolve a specific allele. We take a look at two properties of the mutation: The amount of mutated chromosomes (*Mutation Probability*) and the rate at which a genome of a chromosome is mutated (*Mutation Rate*).

5.4.1. Mutation Probability

Although the mutation operation increases diversity, it lowers the average fitness of the population (Figure 29). This can be explained by the fact, that the mutation only takes place in chromosomes that where generated from individuals with a high fitness. In later iterations it is very likely to undo evolved progress by a random mutation. Without any mutation and without the replacement operation as a second source for diversity, it is expected for the algorithm to perform worse than algorithms with a configuration that includes diversity providing operations. In the case displayed below, the initial population is diverse enough to enable a GA without mutation and replacement to perform better than those who include those operations. It is important here, that this progress relies entirely on the initial population. In case we reduce the population size or analyze a greater amount of iterations, the other configurations will perform better.

We can demonstrate the requirement for mutation by reducing the genetic infor-



(a) Convergence of a DVC GA using different mutation rates

(b) Convergence of a CVC GA using different mutation rates

Figure 29: Effect of mutation probability on convergence. Note that the replacement operation was disabled for this analysis. Mutation has a negative effect on the convergence. It seems that no mutation at all will yield the fastest convergence. The diversity required for the optimization was provided by the initial population. The lower the mutation rate, the more the algorithm relies on pure 'luck' that the genetic information was provided with the initial population. For this reason we have to employ at least a small mutation rate in order to guarantee that all solutions can be reached, even if they were not part of the initial population.

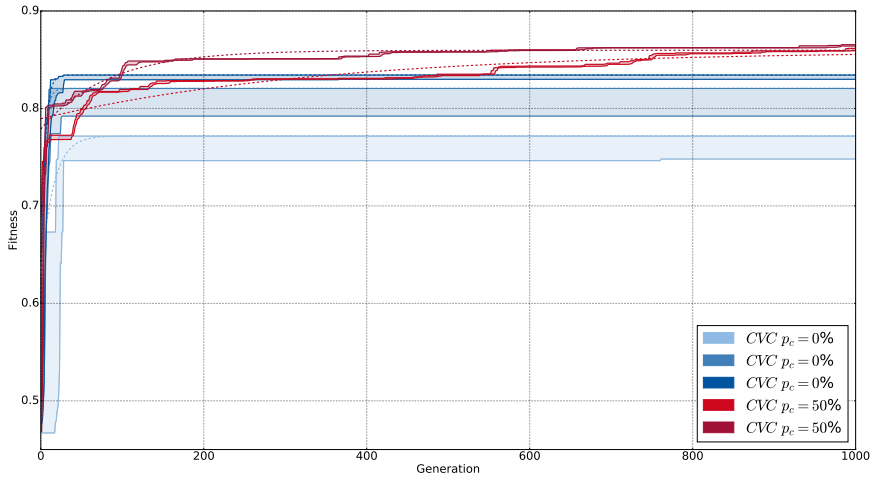
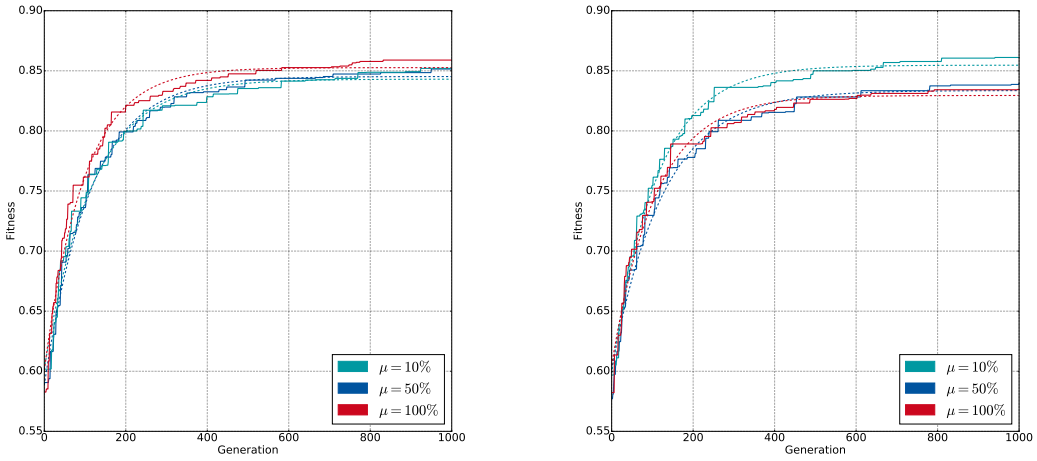


Figure 30: GA running with CVCs on a small population ($N_{pop} = 10$). Mutation is required to reach all possible solutions. Without diversity providing operators, multiple executions of the GA show nondeterministic behavior (blue lines). The algorithm relies entirely on the initial population to contain all genetic information to evolve the solution. With mutation (red lines) the GA converges to higher values and behaves deterministic.

mation in the initial population. By configuring the GA to use CVC with a very small population size ($N_{pop} = 10$) we will reach the upper limit of the convergence in less iterations (Figure 30). We observe that running the GA multiple times in this configuration will provide different results. The genetic algorithm does not behave deterministically anymore. The GA that is configured to use the mutation operation (red lines, Figure 30) provides better results and does not remain on a local maximum. Note, that with mutation the algorithm shows deterministic behavior.

5.4.2. Mutation Rate

The mutation rate μ determines how strongly a mutation can modify a value within a gene. For $\mu = 100\%$ the targeted value in a gene is replaced by a random value, for $\mu = 0\%$ the value is kept the same. (Figure 31). In a DVC set up high mutation rates tend to result in offspring with a higher fitness. Due to the discretization it is more difficult to make 'fine adjustments' to individual values via mutation. In a CVC set up it is possible to make minor corrections to the values. These small mutations support the progress of the algorithm once a certain level of fitness has been reached. When mutating a chromosome with a very low fitness, a high mutation rate significantly increases the chance of fitness improvement. The draw back of worsening its fitness is negligible, as it already is a low fitness chromosome. On the contrary, for high fitness chromosomes only minor changes should be applied in order to avoid a corruption of the properties. The similarity of chromosomes in two subsequent iterations is very high



(a) Convergence of a DVC Genetic Algorithm using different Mutation Rates

(b) Convergence of a CVC Genetic Algorithm using different Mutation Rates

Figure 31: Effect of mutation rates on convergence. A high mutation rate is beneficial for a DVC-GA, but for a CVC-GA the opposite applies. In both cases, the extend of the influence of the mutation rate is small.

which is why mutations should be done in smaller steps when a high fitness level is already reached. We can observe these benefits of both rates in Figure 31b: The high mutation rate provides more progress in the early iterations (red line) $(0 - 100)$, but as the average fitness of the population increases the GA that is configured to use a high mutation rate (turquoise line)(100-1000) provides better results.

5.5. Effects of Replacement Operation on Convergence Behavior

The mutation operation is the second source for diversity among the genetic operators. The replacement operation replaces old chromosomes with N_{repl} new ones from the offspring pool. After this replacement the worst N_{rand} chromosomes are replaced with random chromosomes. Since these random chromosomes did not undergo the evolution process they usually have an ever lower fitness than those removed from the optimized population. However, this is uncritical as the replacement operation does not aim to improve the current average fitness of a population, but to inject new genetic information into it and provide diversity without worsening the fittest chromosomes of the population. Therefore we expect the operation to have a beneficial effect on the fitness of the elite within the population despite lowering the average fitness of the population. Figure 32 displays the effect of a 10% replacement of the population with random chromosomes, compared to replacing none. The *fitness gap* indicates the range of fitnesses within a population. When relying only on Mutation to provide diversity, chromosomes with a low fitness are rapidly expelled from the population. When using replacement the very same chromosomes still get expelled, but new ones are added.

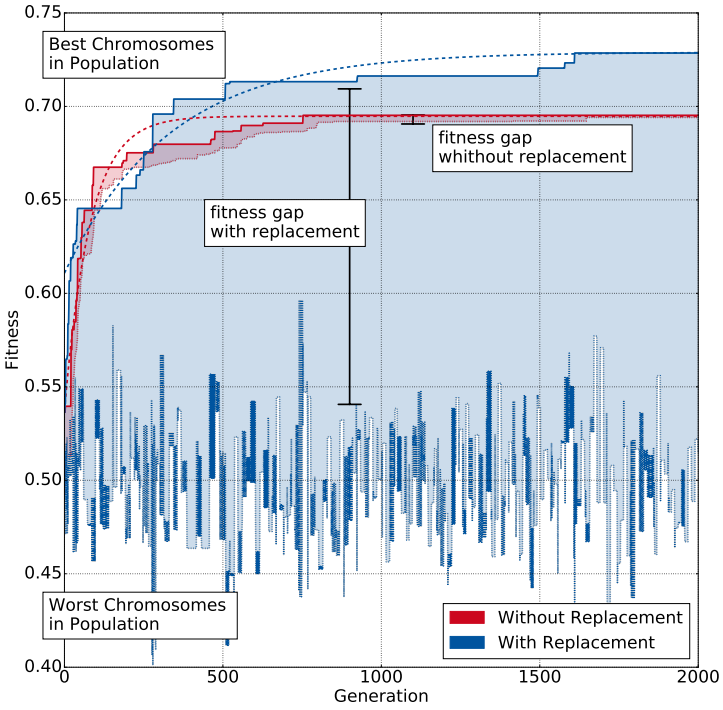
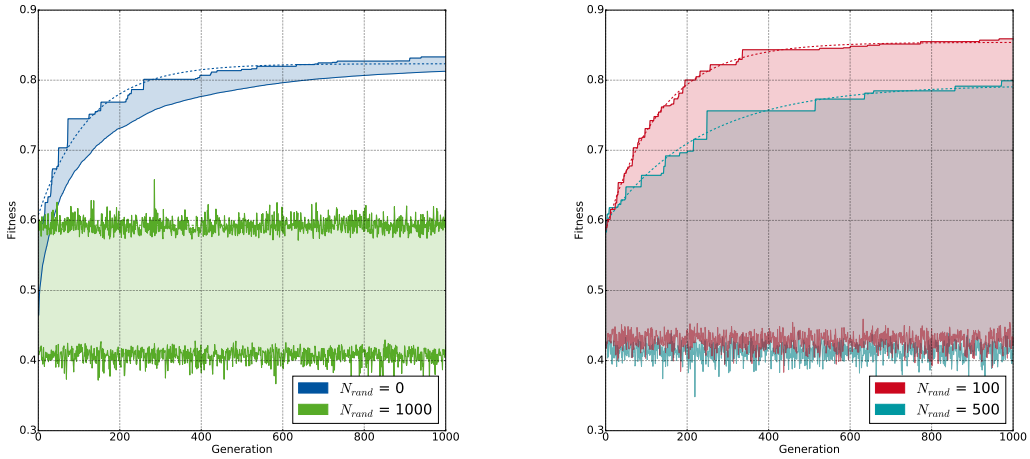


Figure 32: Convergence with and without replacement. Adding random chromosomes into the population introduces diversity, and lowers the lower fitness limit of the population. The fitness gap is wider in comparison to a GA that only uses mutation to manage diversity.

New chromosomes provide completely independent and random genetic information, that allows the algorithm to converge faster.

When replacing chromosomes of the population we have to ensure that the new chromosomes provide more benefit for the optimization than the previous ones. As we are adding chromosomes that are very likely to have a fitness that is below the average of the population we should only replace a small part of the population.

Similar to the improvement conflict described for mutation in Section 5.4, replacement also needs to be balanced to avoid a corruption of the population's achieved fitness level. While too many replacements would turn the algorithm into a Monte-Carlo-Algorithm and compromise its genetic character, too few would lead to a slow convergence for the same reasons as in Section 5.4 (Figure 33b). Figure 33b demonstrates the ideal replacement rate of 10% for a population of 1000 which is the percentage to be used in this simulation.



(a) DVC GA convergence using extreme replacement rates.

(b) DVC GA convergence using common replacement rates.

Figure 33: Effect of different replacement rates on convergence. Note that the Mutation Operation was disabled for this analysis. No replacement provides the same curve (blue line), as known from previous plots about lack of divergence (Fig. 30), replacing the entire population disables the evolutionary process. We observe the range fitness values, random chromosomes are likely to reach (green line). More reasonable replacement rates provide better results (Fig. 33b)

5.6. General Algorithm settings

The behavior of the genetic algorithm can be optimized by choosing a configuration that matches the simulated model. Based on the findings above we present two recommendations for the algorithm: One configuration for a GA using DVCs (Table 6) and one for a GA using CVCs (Table 7). The parameter of the configuration will be explained in detail in this section.

5.6.1. DVC-GA

- **Resolution** Based on Section 5.1 a resolution that matches the resolution of the simulation is recommended. In our case we assume that the placement of a wind turbine will occur with an error not less than $\pm 0.1m$. Thus we simply choose:

$$\rho = \frac{|x_{max} - x_{min}|}{0.1m} \tag{43}$$

- **Population Size** Based on Section 5.4 a population of $N_{pop} = 100$ provides sufficient diversity. A higher diversity would require more runtime, whereas a lower population would lower the progression per iteration of the algorithm.

$$N_{pop} = 100 \tag{44}$$

- **Selection Pool Size** According to Section 5.2 low selection rates are recommended. Best results were accomplished with 10% of the population.

$$N_{sel} = 0.1 \times N_{pop} \tag{45}$$

  In this thesis we do not modify the amount of produced offspring. Therefore for any experiment or computation in this thesis holds: $N_{offspring} = N_{sel}$

- **Mutation Probability** Because the mutation is necessary, but disadvantageous for the fitness at the same time, we choose a low rate (Section 5.4.1).

$$p_c = 0.1 \tag{46}$$

- **Mutation Rate** High mutation rates on DVCs lead to a steeper convergence than low mutation rates (Section 5.4.2). Accordingly we choose the maximal rate.

$$\mu = 1 \tag{47}$$

- **Replacement** The insertion of random chromosomes is important to prevent convergence to only a local maximum and enables the optimizer to reach a global one. Replacement provides the new genetic information without dramatically worsening the progress of the optimization. As presented in Section 5.5 we choose a low replacement rate.

$$N_{rand} = 0.1 \times N_{pop} \tag{48}$$

  In this thesis we do not further analyze changes to N_{repl}. For all experiments and computations holds: $N_{repl} = N_{offspring}$

5.6.2. CVC-GA

The configuration of a genetic algorithm for CVCs is very simililar to one for DVCs. Some differences however are essential.

- **Resolution** The resolution is predetermined by the number system that we use. Changing this parameter will not have any effect on the optimization.
- **Population Size** Based on Section 5.1 a population of $N_{pop} = 100$ performs worse than the same population encoded in DVCs. Incrementing the population size will cause more runtime. Accordingly we chose a slightly higher population size.

$$N_{pop} = 1000 \tag{49}$$

- **Selection Pool Size** Similar to the DVC-GA we choose a low selection rate of 10%.

$$N_{sel} = 0.1 \times N_{pop} \tag{50}$$

  In this thesis we do not modify the amount of produced offspring. Therefore for any experiment or computation in this thesis holds: $N_{offspring} = N_{sel}$

- **Mutation Probability** Similar to the DVC-GA we choose a low mutation rate of 10%. (Section 5.4.1).

$$p_c = 0.1 \tag{51}$$

- **Mutation Rate** High mutation rates on CVCs lead to a steeper convergence than low mutation rates, but in contrast to DVCs high mutation rates tend to be disadvantageous once the population has reached high fitness values, therefore we choose a low mutation rate (Section 5.4.2)

$$\mu = 0.1 \tag{52}$$

- **Replacement**As presented in Section 5.5 replacement causes a similar conflict as the mutation rate. To avoid an undermining of the population's fitness we choose a low replacement rate.

$$N_{rand} = 0.1 \times N_{pop} \tag{53}$$

  In this thesis we do not further analyze changes to N_{repl}. For all experiments and computations holds: $N_{repl} = N_{offspring}$

5.7. Summary

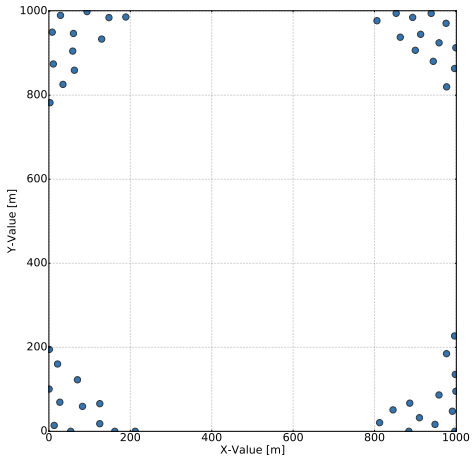
We presented the effects different parameters have on a Genetic Algorithm with a population consisting either of CVCs or DVCs. As expected, only the operators that interact directly with the chromosomes (Chromosome Factory and Mutation), require different settings for a CVC and a DVC configuration. The remaining operators only differ in the extend of the influence of a parameter change. We stated, that due to the great differences in search space, CVCs converge slower than DVCs. In conclusion, we presented changes to the parameters of the CVC configuration to counteract the draw back in part. The algorithm settings defined above allow us in the following section run the evolutionary algorithm for the optimization of wind turbine placements.

6. Experimental Results

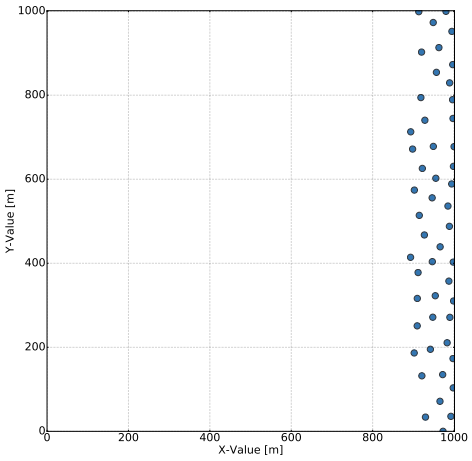
In this section we present the results of the complete set-up for the genetic algorithm. We will compare the results for both continuous values and discrete values. We will Optimize a model of the HornsRev1 wind park. As an objective functions *AEPnet*, *NPV* and *IRR* (See Section 3.5) will be used. These simulations provide sufficient complexity to evaluate the difference in performance of both GA-configurations.

6.1. Test Functions

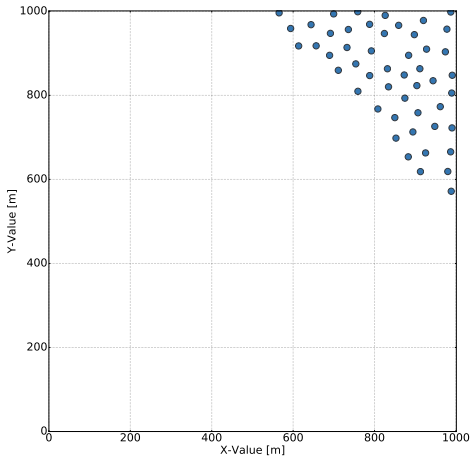
The following plots show the results of the four test functions we used to analyze the convergence behavior in the previous chapter.



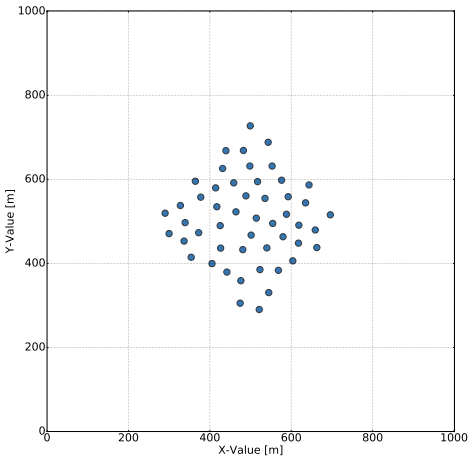
(a) Optimization for objective function `CORNER` $f = 0.86\%$



(b) Optimization for objective function `MAX SIDE` $f = 0.95\%$



(c) Optimization for objective function `TRIANGLE` $f = 0.84\%$



(d) Optimization for objective test function `RHOMBUS` $f = 0.83\%$

1

Figure 34: Overview of the different test function results.

6.2. Experimental Setup

The Model we use in this thesis is based on the HornsRev1 Power plant, located in the German bay (see Figure 1). HornsRev is a power plant, that consists of 80 horizontal axis wind turbines (Model Vestas V80). The plant covers an area of $3800m$ by 5000m. As discribed in section 3.1, the model uses statistical data of wind speed and direction,

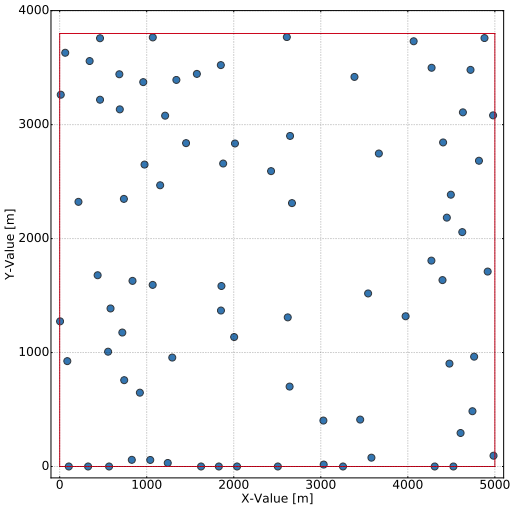
Average time required for:	CVC	DVC
iteration [ms]	41308	1141
selection operation [ms]	94	1
crossover [ms]	34375	844
mutation [ms]	951	24
replacement [ms]	5886	271

Table 1: Time spent on selected operations during optimization (AEP)

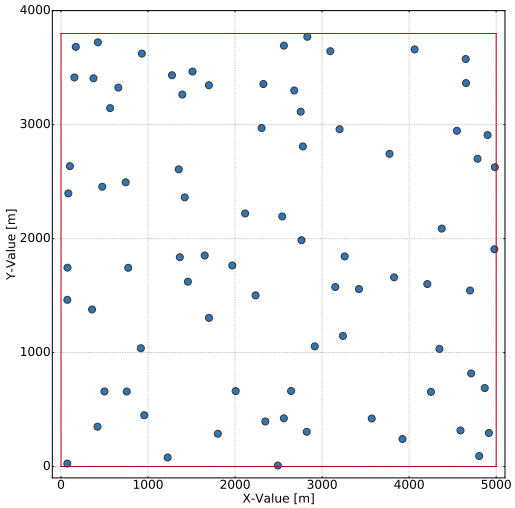
accounts for downtimes caused by maintainance or demage to turbines and includes economic data to compute the current value of the project.

6.3. Test case Horns Rev AEP

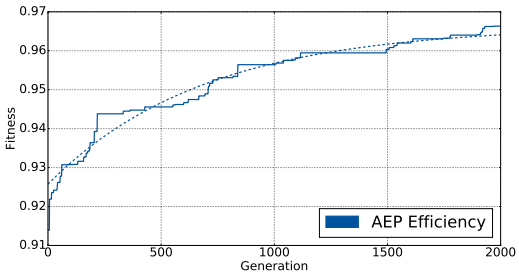
We optimized the offshore wind park layout of HornsRev1 in order to maximize the AEP as defined in Section 3.5.2. For this purpose we use two configurations: One employing CVCs (Figures 35c and 35d) and a second one, employing DVCs (Figures 35a and 35b). We observe far better results using DVCs. The Table 1 presents runtime results. The CVC-configuration required 24 hours (88732,62 s on an intel CORE i7) to compute. The DVC-configuration finished within 48 minutes on the same configuration.



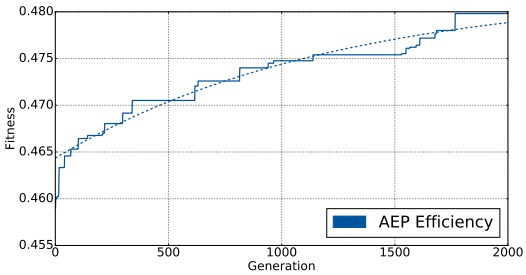
(a) AEP DVC



(c) AEP CVC



(b) Convergion of the genetic algorithm (Configuration as in Table 6)



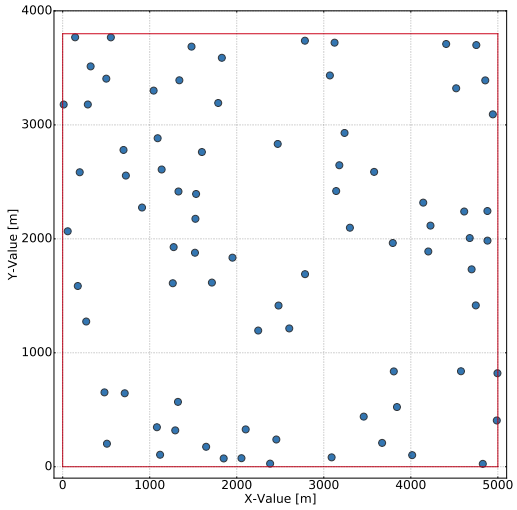
(d) Convergion of the genetic algorithm (Configuration as in Table 7)

6.4. Test case Horns Rev NVP

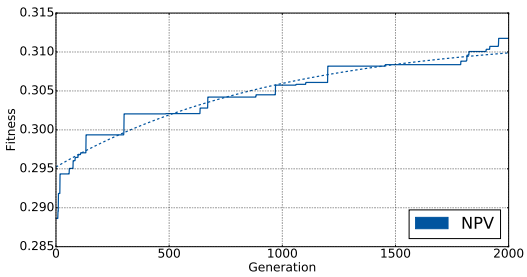
In this test case we compare the optimization of the two genetic algorithms, that optimizes the objective function NPV (see section 3.5.4). The performance of both algorithms differs a lot. The GA, configured to employ CVCs required a lot more time, converging towards a lower value within the same amount of iterations. The CVC-configuration took 20 hours (73594,79s) to compute while the DVC-configuration only needed 137 minutes. Table 2 provides detailed information on the runtime costs.

Average time required for:	CVC	DVC
iteration [ms]	36796	3966
selection operation [ms]	89	1
crossover [ms]	25684	873
mutation [ms]	932	24
replacement [ms]	5586	276

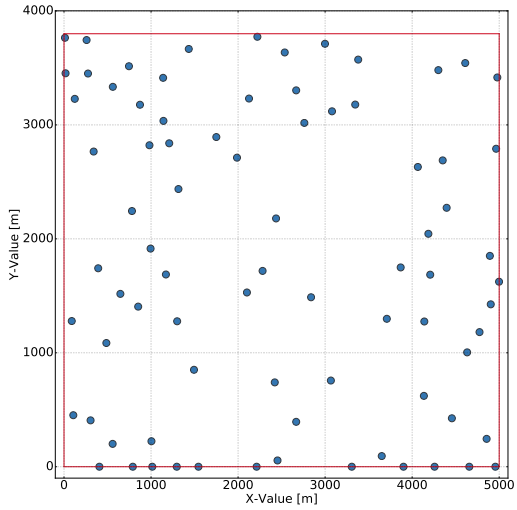
Table 2: Time spent on selected operations during optimization (NVP)



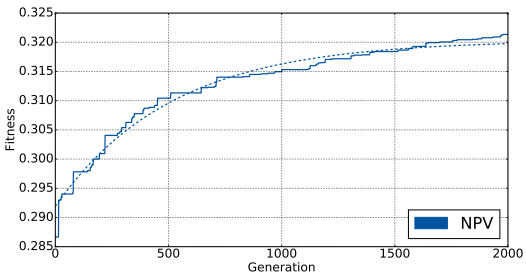
(a) NPV CVC



(b) Convergion of the genetic algorithm (Configuration as in Table 7)



(c) NPV DVC



(d) Convergion of the genetic algorithm (Configuration as in Table 6)

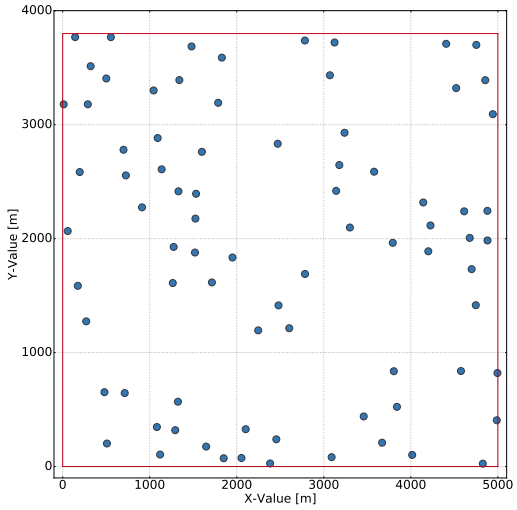
6.5. Test case Horns Rev IRR

In this test case we compare the optimization of the two genetic algorithms, that optimizes the objective function IRR (see section 3.5.5). We notice a difference in performance similar to the previous results. CVCs require more time to converge. The

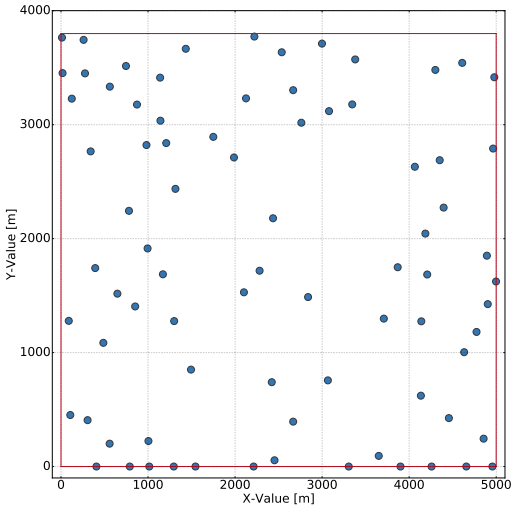
Average time required for:	CVC	DVC
iteration [ms]	6768	1173
selection operation [ms]	75	1
crossover [ms]	4338	843
mutation [ms]	180	25
replacement [ms]	2172	302

Table 3: Time spent on selected operations during optimization (IRR)

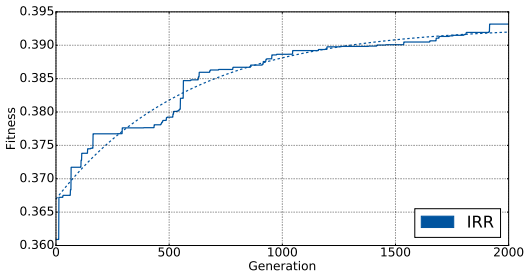
CVC-configuration took 5 hours (17768,83s) to compute while the DVC-configuration only needed 49 minutes. Detailed information on the runtime requirements can be found in Table 3).



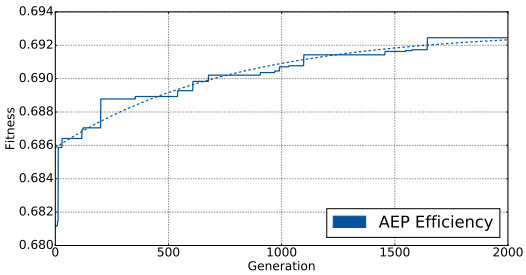
(a) IRR CVC



(c) IRR DVC



(b) Convergion of the genetic algorithm (Configuration as in Table 7)



(d) Convergion of the genetic algorithm (Configuration as in Table 6)

7. Conclusion

In this section we will summarize the findings of this thesis and provide some suggestions for future work.

7.1. DVCs and CVCs

We were able to evaluate the differences between the convergence behavior of CVCs and DVCs. Although CVCs permit a more precise placement of wind turbine, they converge slower due to their bigger search space. We were unable to compensate the slower convergence rate by increasing the population size and by adapting the configuration. There is no benefit in using a resolution that exceeds the one of the objective function. The additional placement options increase the search-space and slow down the optimization as a consequence. Figure 38 demonstrates the basic problem of discretization of genome values. For a fixed amount of generations the following can be observed:

- In case we observe a high fitness and a low runtime, it is very likely that the search space of the given problem is a small one.
- In case we observe a low runtime and a big search-space, it is very likely that the fitness that has been reached up to this generation is a small one.
- In case we observe a high fitness for a GA with a big search space, then it is very likely that the run time was very low due to a big population size that had to be used to reach the observed fitness.

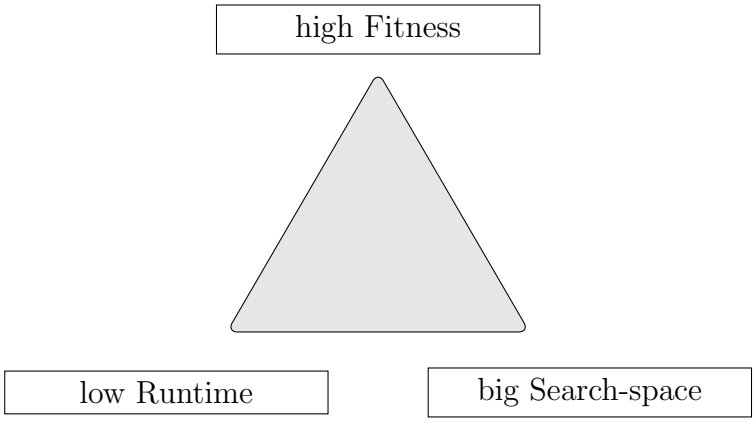


Figure 38: Runtime is influenced by the algorithms setup

Note, that GAs are heuristic search algorithms, and depending on their configuration can 'get lucky' in finding a solution (see Figure 30), therefore we can only can use Figure 38 as an indicator, on how these parameters interact. A GA should be setup with a configuration that minimizes the dependency on chance to produce a good

result, and instead be configured to converge towards the same result on any execution.

We conclude that a genetic algorithm that employs continuous gene parameters performs worse than one using discretized gene parameters, as long as the discretization is within a reasonable precision. Continuous parameters increase the search-space of the algorithm unnecessarily and can not path through their precision. Using discrete parameters is much more efficient and yields better results in all test cases.

7.2. GA Configuration

We also presented findings on how to configure the genetic algorithm, not only for the model used in this thesis, but also provided general principles on how to configure this set-up. A set of specific configurations was developed (Table 6).

7.3. Future Work

- The genetic algorithm that was developed for this thesis is also capable to optimize layout data of Solar Power Plants, as they are introduced in [24]. The algorithm could be configured to match the parameters of the Heliostat-Field model.
- We observed a high conversion rate when using smaller resolutions. However, those configurations converged to worse results than the higher resolution counterparts. A variable resolution could be implemented, that increases with each iteration of the algorithm (See [22]).
- As part of this thesis we developed a framework to facilitate easy comparison of optimizers. An evaluation of performance of the genetic algorithm and e.g. Tabu search is possible.
- The implemented genetic Library permits both pre- and postprocessing steps. The closer the optimization gets to the maximum, the slower it converges. A stopping criteria could terminate the genetic algorithm optimizer in advance and optimize the remaining part with an algorithm with a lower runtime.

A. Appendix

Simulation Settings	
Max X (x_{max})	5000
Max Y (y_{max})	5000
Amount Turbines	50
Turbine Radius	20
Objective Funktion	`TRIANGLE`
Resolution ρ	10000
Genetic Algorithm Settings	
Amount Chromosomes	1000
Selection Pool size	500
Elite Pass through	100
Offspring Pool size	500
Choosing probability p_c	0.5
Mutation Target Min	5
Mutation Target Max	12
Mutation Rate	0.3
Elitism	100
Replacement	100
Random Chromosomes	100
Maximum Iterations	1000
Maximum fitness	1

Table 4: Default settings

Simulation Settings	
Max X (x_{max})	5000
Max Y (y_{max})	5000
Amount Turbines	50
Turbine Radius	0
Objective Funktion	`TRIANGLE`
Resolution ρ	10000
Genetic Algorithm Settings	
Amount Chromosomes	100
Selection Pool size	50
Elite Pass through	10
Offspring Pool size	50
Choosing probability p_c	0.5
Mutation Target Min	5
Mutation Target Max	12
Mutation Rate	0.3
Elitism	10
Replacement	10
Random Chromosomes	10
Maximum Iterations	1000
Maximum fitness	1

Table 5: Settings for low algorithm runtime

Simulation Settings	
Max X (x_{max})	5000
Max Y (y_{max})	5000
Amount Turbines	50
Turbine Radius	20
Objective Funktion	`SIMULATION`
Resolution ρ	50000
Genetic Algorithm Settings	
Amount Chromosomes	100
Selection Pool size	50
Elite Pass through	10
Offspring Pool size	50
Choosing probability p_c	0.1
Mutation Target Min	1
Mutation Target Max	5
Mutation Rate	1
Elitism	10
Replacement	50
Random Chromosomes	10
Maximum Iterations	1000
Maximum fitness	1

Table 6: DVC settings

Simulation Settings	
Max X (x_{max})	5000
Max Y (y_{max})	5000
Amount Turbines	50
Turbine Radius	20
Objective Funktion	`SIMULATION`
Resolution ρ	50000
Genetic Algorithm Settings	
Amount Chromosomes	1000
Selection Pool size	500
Elite Pass through	100
Offspring Pool size	500
Choosing probability p_c	0.1
Mutation Target Min	1
Mutation Target Max	5
Mutation Rate	0.1
Elitism	100
Replacement	500
Random Chromosomes	100
Maximum Iterations	1000
Maximum fitness	1

Table 7: CVC settings

References

[1] News article about john holland. URL: https://www.santafe.edu/news-center/news/in-memoriam-john-holland.

[2] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008.

[3] Belarmino Adenso-Díaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006. doi: 10.1287/opre.1050.0243. URL https://doi.org/10.1287/opre.1050.0243.

[4] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, P. Jonathon Phillips, and Yui Man Lui. *Automatically Searching for Optimal Parameter Settings Using a Genetic Algorithm*, pages 213–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-23968-7_22. URL https://doi.org/10.1007/978-3-642-23968-7_22.

[5] Roni Cakar. Uncertainty quantification of wind farm models.

[6] Christian Steiness. Picture of hornsrev1 wake effects. URL: imgur.com/qruVcnu.

[7] Kenneth A. De Jong and William M. Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1):1–26, Mar 1992. doi: 10.1007/BF01530777. URL https://doi.org/10.1007/BF01530777.

[8] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, Jul 1999. doi: 10.1109/4235.771166.

[9] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19 – 31, 2011. doi: https://doi.org/10.1016/j.swevo.2011.02.001. URL http://www.sciencedirect.com/science/article/pii/S2210650211000022.

[10] Geneial. Genetic algorithm library geneial. URL: http://www.geneial.org.

[11] S. Ghoshray and K. K. Yen. More efficient genetic algorithm for solving optimization problems. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, volume 5, pages 4515–4520 vol.5, Oct 1995. doi: 10.1109/ICSMC.1995.538506.

[12] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. doi: 10.1145/103162.103163. URL http://doi.acm.org/10.1145/103162.103163.

[13] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, first edition, 1989.

[14] J. S. Gonzalez, A. G. G. Rodriguez, J. C. Mora, J. R. Santos, and M. B. Payan. A new tool for wind farm optimal design. In *2009 IEEE Bucharest PowerTech*, pages 1–7, June 2009. doi: 10.1109/PTC.2009.5281977.

[15] Gregor Heiming. Modellierung und simulation von offshore windparks.

[16] John H. Holland. Genetic algorithms.

[17] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[18] Sushil J. Louis and Gregory J. E. Rawlins. Predicting convergence time for genetic algorithms. 08 1994.

[19] Niels Otto Jensen. *A note on wind generator interaction*. Number 2411 in Risø-M. 1983.

[20] Ibrahim Kucukkoc, Aslan Karaoglan, and Ramazan Yaman. Using response surface design to determine the optimal parameters of genetic algorithm and a case study. 51:5039–5054, 06 2013.

[21] Matthew A. Lackner and Christopher N. Elkinton. An analytical framework for offshore wind farm layout optimization. *Wind Engineering*, 31(1):17–31, 2007. doi: 10.1260/030952407780811401. URL `https://doi.org/10.1260/030952407780811401`.

[22] Laura A. McLay and David E. Goldberg. Efficient genetic algorithms using discretization scheduling. *Evol. Comput.*, 13(3):353–385, September 2005. doi: 10.1162/1063656054794752. URL `http://dx.doi.org/10.1162/1063656054794752`.

[23] Johanna Nellen, Benedikt Wolters, Lukas Netz, Sascha Geulen, and Erika Ábrahám. A genetic algorithm based control strategy for the energy management problem in phevs, 10 2015.

[24] Pascal Richter, David Laukamp, Levin Gerdes, Martin Frank, and Erika Ábrahám. Heliostat field layout optimization with evolutionary algorithms, 09 2016.

[25] F. Rivas-Davalos and M. R. Irving. An efficient genetic algorithm for optimal large-scale power distribution network planning. In *2003 IEEE Bologna Power Tech Conference Proceedings,*, volume 3, pages 5 pp. Vol.3–, June 2003. doi: 10.1109/PTC.2003.1304483.

[26] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[27] Michele Samorani. The wind farm layout optimization problem. 01 2010.

[28] J. Bruce Walsh. Theoretical population genetics. by gale j. s.. london: Unwin hyman. 1990. 417 pages. *Genetical Research*, 58(1), 1991. doi: 10.1017/S0016672300029670.