University of Natural Resources and Life Science, Vienna
Department of Economics and Social Sciences
Institute for Sustainable Economic Development

Generic Optimization of a Wind Farm Layout using a Genetic Algorithm

Master thesis

to fulfill the requirements for the academic degree of

Dipl.-Ing.

Sebastian Gatscha, B.Sc.

H0507053

Supervisors:

Univ.-Prof. Dr. DI Erwin Schmid

Dr. DI Johannes Schmidt

Vienna, December 2016

Ich erkläre eidesstattlich, dass ich die Arbeit selbständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert, durch Fußnoten gekennzeichnet bzw. mit genauer Quellenangabe kenntlich gemacht habe.

Abstract

The wind industry is expanding rapidly such that competition for favorable wind farm locations is increasing. Optimized wind farm configurations to increase expected power outputs will therefore gain in importance. Although the available area of a wind farm is usually limited in its extent and shape, previous research has mostly concentrated on a hypothetical rectangular area. The aim of this thesis is to maximize the expected power output of any wind farm by optimizing the placement of its wind turbines. A new terrain effect model is applied using real geodata in order to take into account the effects of topography and land cover. A genetic algorithm has been developed for the optimization process and the Jensen Model has been used to measure the influential wake effects between the wind turbines of a wind farm. The terrain effect model is only valid for areas in Europe, because it requires a digital elevation model and the land cover types from the Corine Land Cover Database, which is only available for Europe. The elevation data is used to model local wind and air conditions and merged with the land cover information to model the surface roughness and wake spreading constant over an area. The genetic algorithm allows for different possible parameter settings, the effects of these on power output are examined on a small test data set. The best setting of the algorithm is used to solve a widespread, more complex wind farm layout problem and the results are compared to previous methods. The best setting is also used in a terrain effect model to analyze a real wind farm with irregular area. The code of the algorithm used for the thesis can be found in the appendix and the latest version is available online (*https://github.com/YsoSirius/windfarmGA*) and can thus be used free of charge, modified or expanded as desired.

Zusammenfassung

Die rasante Expansion der Windindustrie erhöht den Wettbewerb um Gunststandorte für Windparks mit hohen Windgeschwindigkeiten und leichtem Zugang. Daher gewinnen optimierte Windparkkonfigurationen mit hohen erwarteten Leistungsausgaben an Bedeutung. Obwohl die verfügbare Fläche eines Windparks in ihrer Ausdehnung und Form begrenzt ist, hat sich die bisherige Forschung hauptsächlich auf eine hypothetische rechteckige Fläche konzentriert. Ziel dieser Masterarbeit ist es, die erwartete Leistung eines beliebigen Windparks zu maximieren indem die Platzierungen der Windturbinen optimiert werden. Ein neues Geländeeffektmodell wird mit realen Geodaten angewendet, um die Auswirkungen von Topographie und Landbedeckung auf die erwartete Leistung zu untersuchen. Für den Optimierungsprozess wird ein genetischer Algorithmus verwendet. Das Jensen-Modell wird verwendet, um die einflussreichen Wake-Effekte zwischen den Windturbinen eines Windparks zu messen. Das Geländeeffektmodell ist nur für Gebiete in Europa gültig, da es ein digitales Höhenmodell und die Landbedeckungstypen der Corine Land Cover Database benötigt, die derzeit nur für Europa verfügbar sind. Die Höhendaten werden für die Modellierung von lokalen Wind- und Luftbedingungen verwendet und mit den Landbedeckungsinformationen ergänzt, um die Oberflächenrauhigkeit und die Wake-Ausbreitungskonstante heterogen über die betrachtete Fläche darzustellen zu können. Da der genetische Algorithmus unterschiedliche Einstellungen zulässt, werden die Auswirkungen dieser auf die erwartete Leistung in einem ersten Schritt an einem kleinen Testdatensatz untersucht. Die beste Einstellung des Algorithmus wird verwendet, um ein weitverbreitetes, komplexeres Windpark-Layout-Problem zu lösen, wobei die Ergebnisse mit früheren Methoden verglichen werden. Schließlich werden mit

der besten Einstellung die Geländeeffekte eines echten Windparks mit unregelmäßiger Fläche analysiert. Der Code des Algorithmus, der für diese Arbeit verwendet wurde ist im Appendix zu finden und die neueste Version ist online verfügbar (*https://github.com/YsoSirius/windfarmGA*) und kann somit kostenlos verwendet, modifiziert oder erweitert werden.

Table of Contents

1. Introduction

The use of wind energy has been growing globally for the past 20 years. In 2015, the globally installed wind power capacity reached about 433 000 MW (Global Wind Energy Council, 2015). Increased deployment of wind turbines was facilitated by decreasing costs and ongoing technical improvements, such as the increase in turbine height and rotor radius (International Renewable Energy Agency, 2012).

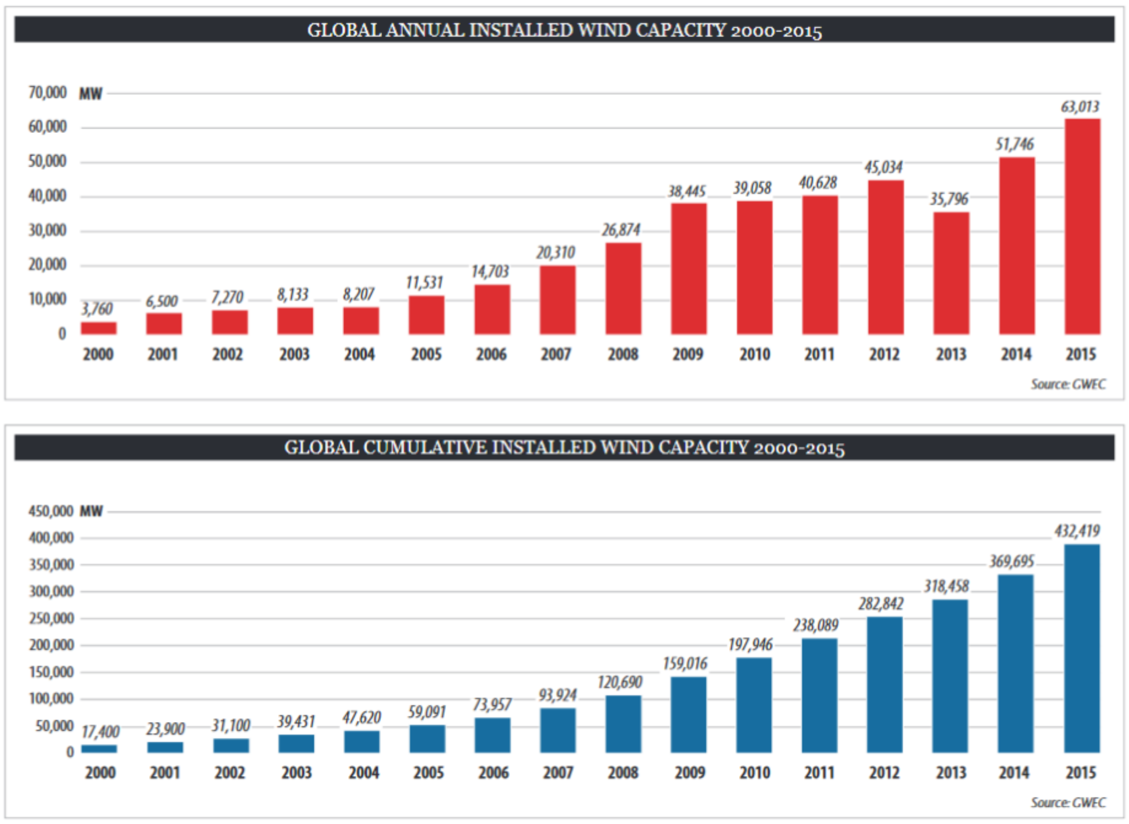


Figure 1: Global annual installed and cumulative installed wind capacity from 2000 - 2015 (Global Wind Energy Council, 2016, p. 14)

These improvements facilitated increasing energy performance of wind turbines. However, as wind turbines are rarely placed individually, but rather as a part of a wind farm with several wind turbines, it is important to consider the negative effects of mutual influencing wind wakes on the overall output of the wind farm (Kusiak & Song, 2009). When wind hits the blades of the turbine rotor, energy is extracted from the wind, therefore reducing the wind speed and influencing the wind turbulence profile behind the turbine. This phenomenon is known as a wake effect (Samorani, 2013).

Therefore micrositing, the specific placement of turbines in a wind farm, plays a crucial role in planning and constructing efficient wind farms and may become even more essential in the future (Yang, et al., 2015). The wind farm layout problem is considered to be Nondeterministic Polynomial Time (NP) Complete, which means that the time required for an exact solution is super-polynomial in the instance of the problem, at least as long no polynomial algorithm for NP problems is found (Williams, 2014). Therefore, the use of heuristic search techniques, like genetic algorithms, is often used in micrositing optimization problems, as here computational time and optimality of the solution can be well balanced (Chen, et al., 2013). Plenty of other heuristic methods were applied to the wind farm layout problem, such as Simulated Annealing, Particle Swarm Optimization, Ant Colony Search Algorithm or Global Greedy Algorithm, but "(...) *the share of genetic algorithms is more than 75% for wind farm layout optimization*" (Shakoor, et al., 2015). Although those heuristic methods have shown reasonable results, recent work has been carried out on gradient-based local search techniques, which outperform genetic algorithms in terms of computational cost and quality (Guirguis, et al., 2016).

Previous research focused mainly on a hypothetical wind farm with rectangular shape and often disregarded elevation or terrain effects. The aim of this thesis is to maximize the expected power output of a wind farm by optimizing the configuration of its wind turbine locations. A new terrain effect model is applied using real geodata in order to take into account the effects of topography and land cover on energy outputs..

The thesis addresses the behavior and the essence of genetic algorithm by discussing and visually comparing different available settings and methods on a simple test case. The best and most efficient setting is used for two further, more complex optimization problems. First, a well-known wind farm layout problem is optimized and second, the new terrain effect model is applied to a layout problem. The impacts of the terrain model on the expected energy outputs and the resulting layouts are examined.

One of the main driving ideas behind of this research is that the program code is free to use and open to public access, thus, inviting people to further test, expand and improve the behavior and performance of the proposed algorithm.

The following chapter 2 describes the theoretical fundamentals of wind energy production and genetic algorithms. Chapter 3 then looks more closely at the developed genetic algorithm and describes the required and optional input data as well as the methods used for the optimization process. In order to check the algorithm and to find out the best settings, various tests are performed in Chapter 4 and their results are visualized and described. Finally, in Chapter 5, these results are further discussed, conclusions are drawn and possibilities for improvement proposed.

2. Theoretical Background

The following chapter outlines the theoretical background of the proposed optimization algorithm. First, the mathematical models and conditions describing the wind energy production of a wind farm are discussed. Second, a brief description of genetic algorithms is presented, showing the general procedure and some limitations of genetic algorithms.

2.1. Theory of Wind Energy

Wind is the natural flow of air, consisting of various gases in the atmosphere of the earth. That wind movement is actually the kinetic energy of wind. In classical mechanics, the kinetic energy contained in a mass point is relative to its mass *m* and its velocity *v,* and is described by the following equation (Kalmikov & Dykes, 2011):

$$E_{kinetic} = \frac{1}{2}\, m\, v^{2} \qquad \text{Equ. 1}$$

The mass of air that is passing through the wind turbine, the rotor area *A*, in a certain momentum is calculated with the velocity of the wind *v*, the time *t* and the air density *ρ* through the following equation (Kalmikov & Dykes, 2011):

$$m_{air} = A\, v\, t\, \rho \qquad \text{Equ. 2}$$

2.1.1. Wind Farm Power Production

Inserting equation 2 in the preceding equation 1 and differentiating with respect to time gives the total wind power equation (Zahoransky, 2010):

$$P = \frac{dE}{dt} = \frac{1}{2}\, \rho\, \mathrm{A}\, v^{3} \qquad \text{Equ. 3}$$

To obtain all the power of the wind, the total energy would need to be extracted and the wind velocity behind the turbine would then decrease to zero, which is theoretically considered impossible, as the captured air must also pass the turbine to keep the airflow circulating.

2.1.2. Theoretical Limit of Wind Energy Extraction

German physicist Albert Betz analyzed the correlation between the input and output wind velocity of a wind turbine in 1919. He found a theoretical limit for wind energy extraction, known as Betz's Law, for any given wind turbine at 16/27 or 59.3% of the kinetic energy in the wind inflow (Zahoransky, 2010). No currently existing wind turbine can extract energy at that factor. State-of-the-art wind turbines reach at most 75 - 80% of the Betz limit. Nevertheless, the Betz coefficient is used as constant in the calculation of total wind power in the proposed algorithm. The equation for total wind power for a single wind turbine is consequently (Zahoransky, 2010):

$$P = 0.593 * \frac{1}{2} \rho \, A \, v^{3} \qquad \text{Equ. 4}$$

2.1.3. Wake Effect Model by Jensen

To analyze the wake effects between the turbines, the analytical wake model by Jensen is used. He considered a closed-form wake region expanding linearly and the reduction of wind velocity decreasing linearly in the direction of the wind flow (Renkema, 2007). Figure 2 shows the wake effect proposed by Jensen.

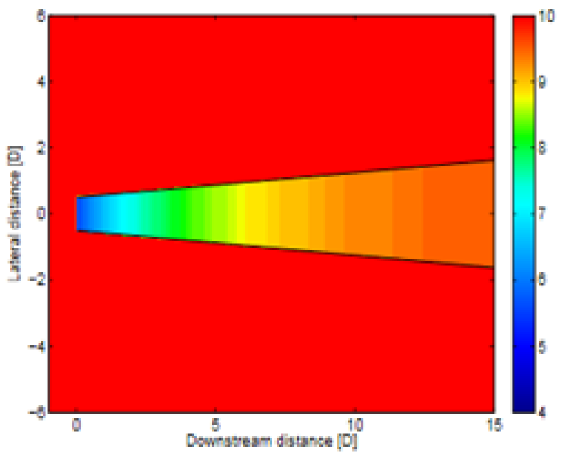


Figure 2: Jensen's wake behind a turbine with V = 10 and K = 0.075 (Renkema, 2007, p. 6)

The diameter of the wake cone at a certain distance is calculated with the rotor radius of the turbine *R*, the wake decay constant *K* and the distance in wind direction between two turbines *D* according to the following equation (Kusiak & Song, 2009):

$$WakeD = 2R + 2KD \qquad \text{Equ. 5}$$

If a wind turbine is inside the wake of another turbine, the wind velocity is reduced. This reduction is calculated with the variables above and the thrust coefficient of a turbine C_T, in this model taken as a constant value of 0.88, given the following equation (Kusiak & Song, 2009):

$$VelDef = \frac{1\text{-}\sqrt{1-C_T}}{(1+\frac{KD}{R})^2} \qquad \text{Equ. 6}$$

As a wind turbine may not be fully within the spectrum of a wake region, my model includes partial shadowing according to Wang (2010).

2.1.4. Partial Wake Effect Modeling

For any given wind turbine in a wind farm, 3 different scenarios are possible which are described and illustrated on the next pages. Either a wind turbine can be completely or only partially in the shadow of another turbine wake, or the turbine is not affected at all by the wake, due to the distance between the two turbines perpendicular to the wind direction being substantial enough.

The variables taken into account for the conditions and calculations are described below:

- R is the radius of the wake cone at the position of the wind turbine
- r is the radius of the turbine rotor
- D is the distance between the wind turbines perpendicular to the wind direction
- A is the shadowed area from the wake of the wind turbine rotor area
- yR is the opening angle of the wake circle segment, at the intersection of the turbine rotor area.
- yr is the opening angle of the rotor circle segment, at the intersection of the wake circle.

The materialization of partial shadowing is shown below with an illustration of all the relevant variables.

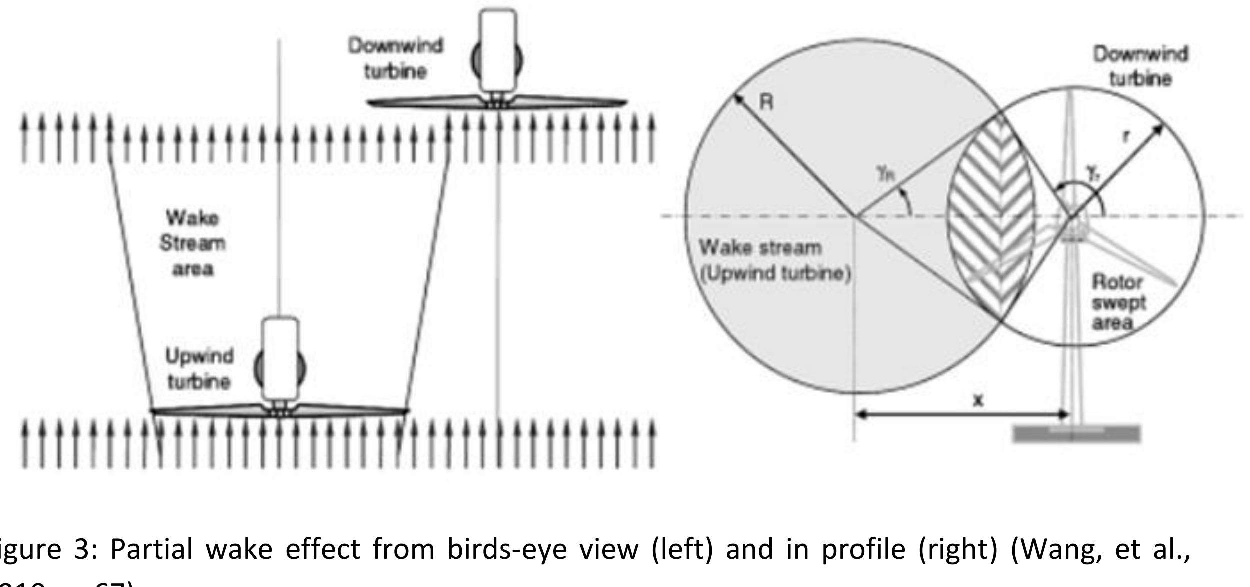


Figure 3: Partial wake effect from birds-eye view (left) and in profile (right) (Wang, et al., 2010, p. 67)

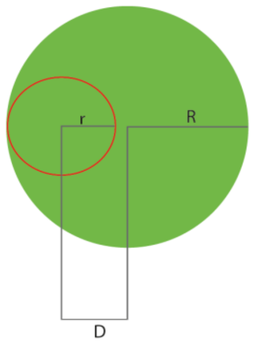
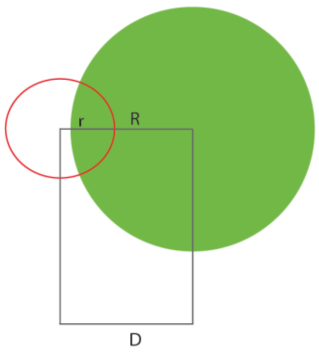
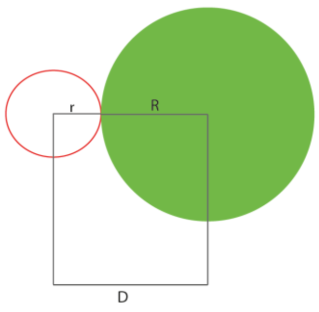
Scenario 1		
	**Description**	The turbine rotor area, framed in red, is fully inside the wake of another wind turbine, colored in green. The turbine area is considered fully shadowed.
	Condition	$\boldsymbol{R - r \geq D}$
	Shadowed Area	$\boldsymbol{A = \pi r^2}$
Scenario 2		
	**Description**	The red turbine area is partially inside the green wake area, so the turbine area is only partially shadowed from the wake and therefore the wind velocity reduction decreases, in comparison to a fully shadowed turbine.
	Condition	$\boldsymbol{R - r \leq D \geq R + r}$
	Shadowed Area	$\boldsymbol{A = R^2 * \left(yR - \left(\frac{\sin 2yR}{2}\right)\right) + r^2 * \left(yr - \left(\frac{\sin 2yr}{2}\right)\right)}$ $\boldsymbol{yR = acos\left(\frac{R^2 + D^2 - r^2}{2DR}\right)}$ $\boldsymbol{yr = acos\left(\frac{R^2 + D^2 - r^2}{2DR}\right)}$
Scenario 3		
	**Description**	The red turbine is not inside the green wake, so velocity reduction of the wind does not occur and the turbine rotor area is considered not shadowed.
	Condition	$\boldsymbol{R + r \leq D}$
	Shadowed Area	$\boldsymbol{A = 0}$

Table 1: The three possible shadowing scenarios of any given turbine

To calculate the wind velocity reduction according to partial shadowing, the shadowed area is taken as a percentage of the whole turbine area and is multiplied with the velocity reduction in the wake, shown in the following equation (Wang, et al., 2010):

$$VelDefPart = \frac{Shadowed\ Area\ A}{\pi r^{2}} * VelDef \qquad \text{Equ. 7}$$

2.1.5. Multiple Wake Effect Model by Katic

If a turbine is inside the wakes of multiple wind turbines, Katic et al. (1987) suggested the following equation to calculate the total wind velocity reduction:

$$VelDefTotal = \sqrt{\sum VelDefPart^{2}} \qquad \text{Equ. 8}$$

The remaining wind velocity that is needed for the evaluation of power output is then computed by subtracting the total velocity deficit from the given wind velocity as shown below (Samorani, 2013):

$$Vnew = V - VelDefTotal \qquad \text{Equ. 9}$$

For every wind farm, 2 power outputs are calculated using equation 4, firstly with respect to wake effects and velocity reduction and secondly, without any wake effects or velocity reductions (Katic, et al., 1987).

$$Reduced\ Power\ Output = 0.593 * \frac{1}{2}\ \rho\ \mathrm{A}\, Vnew^{3} \qquad \text{Equ. 10}$$

$$Full\ Power\ Output = 0.593 * \frac{1}{2}\ \rho\ \mathrm{A}\, V^{3} \qquad \text{Equ. 11}$$

The ratio between these is then calculated and is known as the wind farm efficiency rate (Katic, et al., 1987). The equation thus is:

$$Efficiency = \frac{Reduced\ Power\ Output * 100}{Full\ Power\ Output} \qquad \text{Equ. 12}$$

2.2. Theory of Genetic Algorithms

A genetic algorithm is an evolutionary algorithm, more precisely a guided random search technique, that is widely used when extensive search spaces are evaluated, looking for optimal combinations for a given problem.

2.2.1. Methodology

It is inspired by Darwin's *Theory of Evolution* and consists mainly of the following natural mechanisms to optimize a combinatory problem (Hirsch, 2012):

- Fitness
- Selection
- Crossover
- Mutation

A possible solution to a problem is encoded as a chromosome consisting of several genes that contain the problem-relevant information. The whole set of chromosomes, or individuals, is considered a population, and a population to a particular iteration is referred to as generation. For every individual, a fitness function evaluates an objective value that represents the efficiency of the individual solving the given problem. From all the individuals a subset is subsequently selected, whereof "good" or "fit" parents and an adequate amount of "bad" parents must be available, to avoid the algorithm being stuck in local optima. Parents create new offspring by means of a crossover method which divides their genetic code into several pieces and then reassembles them so that the offspring contain genetic information from their parents. A low mutation rate randomly converts the genetic information to take into account possible random changes. The general sequence of a genetic algorithm for a given problem is as follows (Hirsch, 2012):

1. Create an initial population with a certain amount of individuals
2. Evaluate the fitness value of all individuals
3. Select parents based on their fitness value
4. Recombine their genetic code through crossover
5. Mutate a certain amount of genes to avoid premature convergence
6. Go to step 2 until a stop criterion is met

2.2.2. Limitations to Genetic Algorithms

Genetic algorithms represent a robust tool that can be applied to a variety of optimization problems and additionally, their implementation efforts are comparably low (Mehmeti, et al., 2014). Although genetic algorithms and heuristic methods in general are widely used and achieve overall good results, some major drawbacks should be outlined and discussed here.

Since binary-encoded genetic algorithms require a finite set of genes, the search space is divided into a discrete set of possibilities, while the residual search space is disregarded. The real global optima of a search space may therefore be outside of the space that can be reached by such a genetic algorithm. Combined methods that first use a genetic algorithm to find good solutions and then try to improve these solutions with more accurate methods by using a continuous variable formulation for the turbine coordinates might be a way around this problem (Guirguis, et al., 2016).

Even if the real global optimum is a possible solution to the genetic algorithm, there is no guarantee that it will be found - the genetic algorithm may only find local optima (Wendy, 2015). In addition, the method is not able to demonstrate how far the results differ from the global optimum.

The quality of a genetic algorithm also depends strongly on the correct choice and definition of the necessary parameters. A perfectly configured genetic algorithm for a specific problem may be an appropriate means of approaching the solution with low application of resources. However, in the long-term mixed and exact procedures may replace genetic algorithms for important optimization problems (Mehmeti, et al., 2014).

3. Data & Methods

The input data used in the proposed method is generated by the software R and the free, open-source software QGIS. QuantimGIS (QGIS) is used to create the shapefiles where a wind farm layout should be optimized, while R is used to create the remaining input variables, which are described in the following section. The QGIS shape file is loaded into R using the package "rgdal" with the following R command:

```
Shapefile <- readOGR(dsn= "C:/Shapefiles",layer= "NameOfShapefile")
```

R Output 1: Loading a shapefile into R using the function "readOGR" from the library "rgdal"

The new created variable "Shapefile" is then assigned to the input variable (*Polygon1*) of the proposed algorithm.

3.1. Data

The required input variables created in R can be classified into obligatory input data, that must be assigned in order for the algorithm to work and optional input data, that may be assigned - if they are not defined, default values are used. The input variables are shown below with a short description and are further described in Chapter 3 with their corresponding R-function.

3.1.1. Obligatory Input Data

Input Variable Name	Short description of obligatory input variable
Polygon1	The shape file representing the desired area
Rotor	The desired rotor radius in meter
n	The amount of desired wind turbines
vdirspe	The data frame containing the wind speed data

Table 2: Obligatory Input Variables of the proposed method

3.1.2. Optional Input Data

Input Variable Name	Short description of optional input variable
fcrR	A factor of the rotor radius for grid spacing - Default is set to 3
Proportionality	The proportionality factor for grid calculation in percentage - Default is set to 1
iteration	The amount of iterations or generations for the genetic algorithm - Default is set to 100
referenceHeight	The reference height of the wind speed data in meter - Default is set to 50
RotorHeight	The hub height of the wind turbine in meter - Default is set to 100
SurfaceRoughness	The surface roughness of the given area in meter - Default is set to 0.14
mutr	A mutation rate in percentage - Default is set to 0.008
topograp	A Boolean term for consideration of terrain influence - Default is set to "FALSE"
elitism	A Boolean term that indicates whether elitist selection is included - Default is set to "TRUE"
nelit	The amount of elitist individuals per generation - Default is set to 6
selstate	The selection method used; it can either be fixed ("FIX") at 50% of total individuals or at a variable percentage ("VAR"), depending on the development of fitness - Default is set to "FIX"
crossPart1	The crossover method used; the crossover points are either generated at equal intervals ("EQU") or at random intervals ("RAN") - Default is set to "EQU"
trimForce	A Boolean term used for the adjustment of the desired amount of turbines after crossover and mutation. Either a probabilistic approach is taken ("TRUE") or the adjustment is randomly ("FALSE") - Default is set to "TRUE"

Table 3: Optional Input Variables of the proposed method

3.2. Methods

The following chapter discusses in detail the procedure of the proposed optimization algorithm. I wrote the genetic algorithm completely in R-Studio, which is a free Integrated Development Environment (IDE) for the programming language R. In addition to the base packages of R, the following freely available packages are included to solve and visualize the wind farm layout problem:

- package calibrate
- package data.table
- package dplyr
- package ggplot2
- package googleVis
- package gtools
- package maptools
- package raster
- package RColorBrewer
- package rgdal
- package rgeos
- package RgoogleMaps
- package varhandle

The algorithm is intended to work as a generic model which in this case means that it can be applied to any given shape file, representing the considered area for the wind farm, containing a desired and fixed amount of turbines with desired uniform hub height and rotor radius.

Wind can come from maximum 36 different wind sectors, each covering 10 degrees of a wind rose with total 360 degrees, as illustrated in the figures below. The size of a single wind sector is correlated to the occurrence of wind coming from this sector and therefore to its probability. The colors indicate the incoming wind velocities.

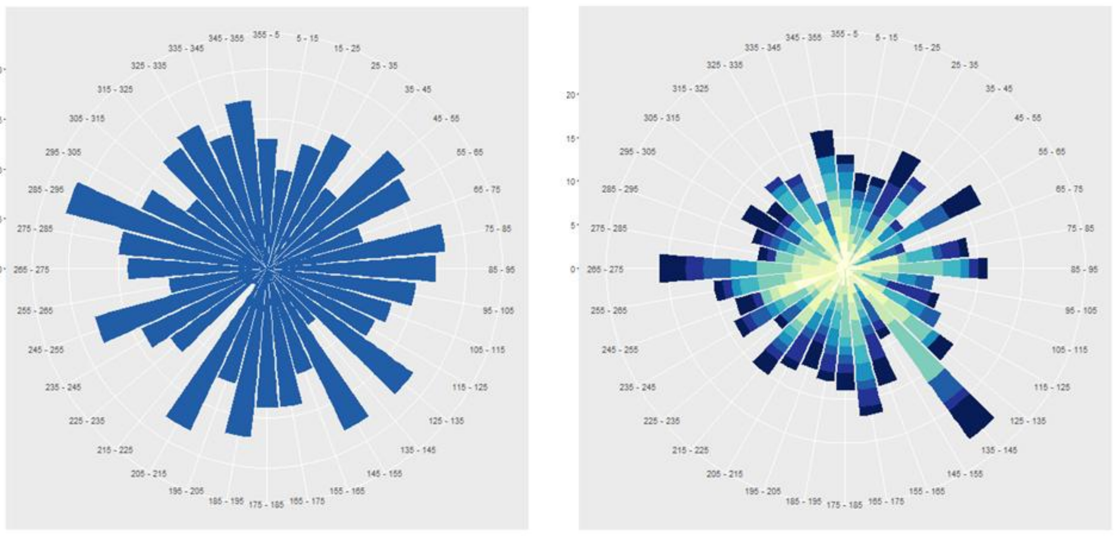


Figure 4: Random wind rose with 36 different wind direction sectors and uniform wind velocity (left) or varying wind velocities (right)

The proposed algorithm takes the mean wind velocity for every wind sector for the calculation of wind energy output, as I consider it sufficient for the optimization of a wind farm layout. For more realistic energy output estimations, the algorithm should be extended so that the Weibull distribution of wind speeds can be used instead.

3.2.1. Polygon to Grid (*GridFilter*)

The first step of the algorithm is to fit a grid in a given shape file. This shape file must represent an area in Europe as the algorithm uses the European Terrestrial Reference System ETRS89 / ETRS-LAEA, which is the EU-recommended frame of reference for European geodata that accurately represents areas.

The centroids of each grid element are possible locations for wind turbines. The resolution of a single grid element depends on the turbine rotor radius and must cover at least the whole rotor diameter, as well as some safe distance. It is calculated by multiplying the rotor radius with a certain factor. This factor "*fcrR*" can be given optionally to the algorithm but must at least be bigger than 2; otherwise a default of 3 is used. A hypothetical wind turbine with a rotor radius of 20 meters would then be in the center of a 60 X 60 squared-meters area.

A grid contains several squares and may not fit exactly over irregular areas. Therefore, the variable "*Proportionality*" is introduced which indicates the minimum percentage that a particular grid cell must overlay the given area to be displayed as a grid element. The possible values range from 1 to 0.01.

The R function "*GridFilter*" solves this first task and was developed by José Hidasi-Neto and released online. A detailed description of this feature can be found on his blog (Hidasi-Neto, 2014). The following examples illustrate the function with changing parameters on a random irregular area with ~1.8 km^2.

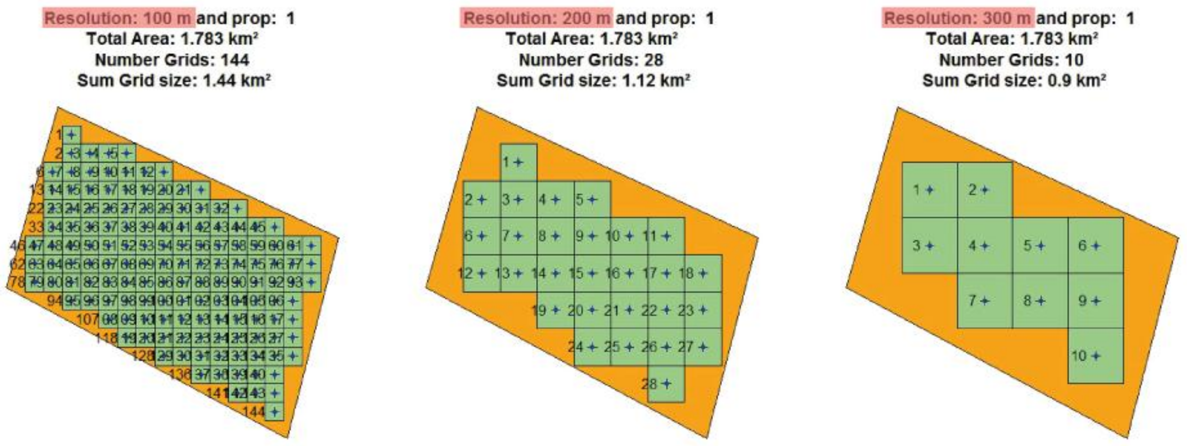


Figure 5: *Resolution* - Changing resolutions with turbine rotor radius of 50 meters, "fcrR"-values of 2, 4 and 6 and a constant "*Proportionality*"-value of 1

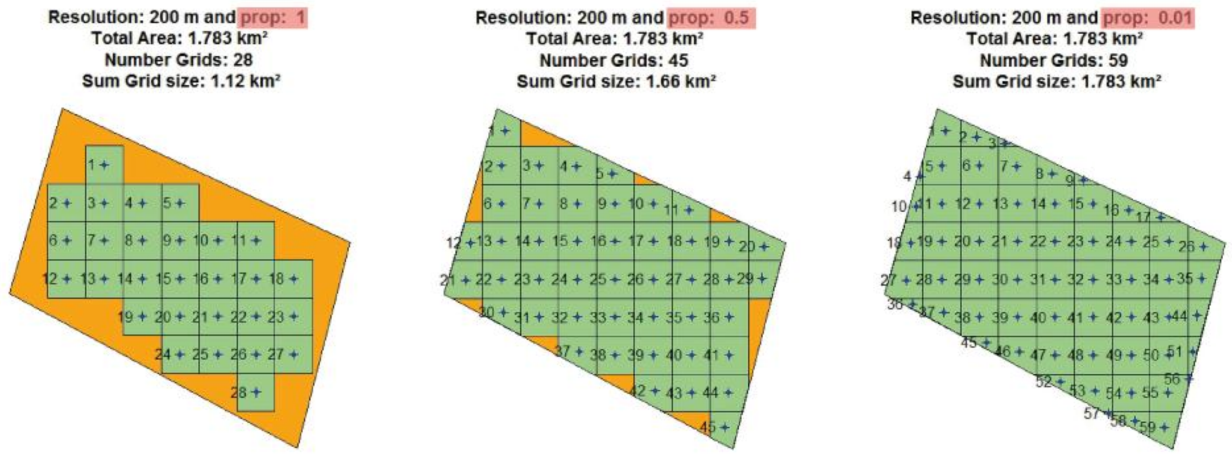


Figure 6: *Proportionality* - The variable "*Proportionality*" is changed from 1 to 0.5 and 0.01 with constant turbine rotor radius of 50 meters and "fcrR"-value of 4

The orange area indicates the remaining area that is not taken into account in the optimization and the blue centroids inside the green grid cells represent the possible wind turbine locations. The result of this function is an indexed data frame containing the centroids as X and Y coordinates. The calculated grid of this function is stored as "*SpatialPolygonsDataFrame*" and as a global variable ("*dry.grid.filtered*") for plotting purpose.

3.2.2. Initialize a Population (*StartGA*)

To ensure that he algorithm has reasonable starting conditions, this function verifies that the number of grid elements corresponds at least to twice the amount of desired turbines. When this test passes, this function randomly generates an initial population, i.e. a certain amount of wind farms with the desired number of wind turbines located at the previously obtained coordinates. The initial population sizing is a prominent and well-researched topic in evolutionary computing. Important findings are that the optimal population size is proportional to the difficulty of the problem and that if the initial population is good, the algorithm has higher chances of finding good solutions (Diaz-Gomez & Hougen, 2007). "Seeding" of a genetic algorithm is an often-used technique that relies on this concept, i.e. the initial population is "seeded" with some good solutions, that must be available and known in advance, which is not the case in the proposed method.

Since there is no general rule for the correct population sizing, and in order to find the best parameters, a separate optimization process would have to be performed, the proposed algorithm instead uses a simple method to dimension the initial population.

$$nStart = \frac{nGrids * nTurbines}{Iteration} \qquad \text{Equ. 13}$$

The variable "*nGrids*" represents the amount of grid cells, "*nTurbines*" the number of required turbines and "*Iteration*" is the total number of iterations. The product of the total amount of grid cells multiplied by the number of turbines gives an indicator for the complexity of the combinatory problem. The more turbines must be placed on an increasing number of locations, the more difficult it is to solve the problem. This product is then divided by the number of required iterations.

I assume that the more iterations the genetic algorithm is able to proceed, the lower the problem size and the higher the chance of finding good solutions will be. The initial population size will therefore be small. With very few iterations, the algorithm is not able to take full advantage of his converging abilities and has to rely on a lot of randomness which will result in big problem sizes and low probabilities of finding good solutions. The initial population size will then be big. Although (Pelikan, et al., 2000) pointed out, that the required population size and number of iterations are both positively correlated to the given problem size, this algorithm uses the preassigned iteration value for the problem size calculation and evaluates the initial population size depending on the resulting problem size.

To illustrate the previous equation, consider a layout problem in which 20 turbines must be optimized on 50 possible grid cells; with 1 iteration, the initial population size “*nStart*” would be 1000, with 100 iterations “*nStart*” would be 10 and with 1000 iterations “*nStart*” would be 1. As a population with only 1 individual would not be able to reproduce, a lower bound of 70 is used for “*nStart*”, which should also ensure for enough diversity in the initial population. The upper limit of “*nStart*” is 200 to ensure that an optimization process does not require too much computational time. The magnitude of the initial population is therefore very limited, and since the population size in the present method is able to grow and shrink according to the crossover parameters, the size of the initial population in this method may be subordinate.

The resulting integer variable "*nStart*" specifies how many random wind farms are created in the initial population. This function supplements the initial individuals with a new binary variable, which indicates the existence of wind turbines. The number 0 does not represent a wind turbine in the grid cells, while a 1 represents a wind turbine.

The result of this function is a list with individuals, containing the Grid-ID's, the coordinates and the new binary variables for n-desired wind turbines. The R-Output below shows a first exemplary individual in the output list of this function.

```
> StartGA(Grid,n,nStart)[1]
[[1]]
ID    X        Y    bin
3  4600134 2758495   1
12 4600404 2758405   1
25 4599954 2758135   1
27 4600134 2758135   1
28 4600224 2758135   1
29 4600314 2758135   1
30 4600404 2758135   1
32 4600044 2758045   1
35 4600314 2758045   1
36 4600404 2758045   1
```

R Output 2: Exemplary output of the function *StartGA* for the first individual in the list

The output of the *StartGA*-function is only used in the first iteration of an optimization process. After the first iteration, the algorithm uses the output of the function *getRects* instead*,* which is described in chapter 3.2.8.

3.2.3. Evaluation of Fitness (*fitness*)

In order to obtain a fitness value for each individual, the resulting wake effects must primarily be evaluated for all wind turbines of all wind farms under all given wind directions. As a first step, the data frame containing the wind information (*vdirspe*) is rearranged. This wind data frame must contain two columns called "*ws*", giving the wind speed values in m/s and "*wd*", indicating the wind direction in degrees and can contain the column "*probab*", showing the probability of a certain wind speed and direction. If this column is not given, I assume a uniform probability for all wind speeds. If several wind directions in the input data correspond to a 10-degree wind sector, the algorithm reduces the multiple directions to a single direction per wind sector and transforms their wind speeds and probabilities. The data frame is then ordered with ascending wind directions. This process is shown in the following example and in the figure below with the raw input data on the left and the adapted data on the right.

```
> data.in
  ws      wd
1 10  300.00
2 14    0.00
3 10    5.00
4 10  180.00
5 10  182.34
```

```
> data.in
  ws   wd probab
2 12    0     40
4 10  180     40
1 10  300     20
```

R Output 3: Exemplary wind speed data frame before (left) and after the conversion (right)

In this example, each given wind direction receives a probability of 20% as no probabilities are given and 5 directions are at hand in total. Rows 2 and 3 are reduced to one row, as their wind directions 0° and 5° are inside a 10° wind sector (0° to 10°). The probability of this sector is 40, the sum of the just assigned probabilities of rows 2 and 3. The final wind speed for this sector is calculated by multiplying the available wind speeds with their relative probability within the wind sector and then summing them. In this case, the equation is $\left(14 * \frac{20}{40}\right) + \left(10 * \frac{20}{40}\right)$ = 12. Following the same procedure for rows 4 and 5 results in a new wind speed of 10 m/s with a probability of 40%. If the resulting sum of probabilities does not amount to 100%, the algorithm scales the probabilities appropriately. In order to calculate the wake effects and expected energy production for a single wind farm with the converted wind data, the algorithm uses the next function.

3.2.3.1. Calculate Energy Production (*calculateEn*)

This function calculates the expected energy output of an individual using the mathematical model described in chapter 1.1, taking into account all available wind sectors. Furthermore, some other methods are implemented that try to model the problem more realistically.

As wind speed data is mostly obtained and referenced to a certain height which can differ from the desired hub height of the wind turbine, I use the wind profile power law to get an estimation of the wind speed at the desired hub height. The wind speed at another height "*ws*" is calculated as follows (Afanasyeva, et al., 2013):

$$ws = ws0 * \left(\frac{RotorHeight}{referenceHeight}\right)^{SurfaceRoughness} \qquad \text{Equ. 14}$$

The input variable "*ws0*" represents the input wind speed at the referenced height ("*referenceHeight*") and "*RotorHeight*" is the given hub height of the wind turbine. "*SurfaceRoughness*" is an indicator of terrain roughness and has in this method a default value of 0.14, which is characteristic for flat terrain. Although terrain characteristics, e.g. elevation, slope, roughness or land use, have strong influences on the expected energy output and the best layout of a wind farm, only few research papers and optimization models were found with an extended terrain effect model (Zhang, 2013).

This algorithm can optionally include a terrain effect model which tries to expand previous methods. To take terrain effects into account, the input variable *topograp* must be set to "TRUE" (topograp="TRUE") - as the default value is "FALSE".

3.2.3.2. Terrain Effect Model

The terrain effect model consists of 4 sub methods that try to model local wind speeds, air densities, surface roughness, and wake decay values according to the given terrain.

3.2.3.2.1. Wind Speed Multiplier

This method is based on the orography model of Saavedra-Moreno (2011). Existing hills and valleys of the terrain influence local wind conditions. The rule may not be coherent for all cases, but it is assumed that wind speeds at the top of a hill will be higher than at the bottom of a hill. This rule is incorporated in the Wind Speed Multiplier method.

To accomplish this task, first, a CIGAR-SRTM elevation raster with a resolution of 90 meters is downloaded with the R function "getData" of the library "raster", as shown below:

```
srtm <- getData('SRTM', lon=LonPol, lat=LatPol);
```

R Output 4: R-Function to get a SRTM - elevation dataset

The elevation data is downloaded as a 5 x 5 degree tile, where the longitude and latitude coordinates ("*LonPol*" and "*LatPol*") of the uploaded polygon are enclosed. The tile is then masked and cropped with the input polygon, resulting in an elevation raster for the desired area only. The height values of every raster cell are then divided by the mean height of the total area. If a turbine site has a higher elevation than the average elevation of the entire area, its wind speed multiplier will be greater than 1 and its wind speed will be intensified. If a turbine is located at a lower altitude than the mean elevation, the wind speed multiplier will be lower than 1 and wind speed will be reduced.

Since the exact formula of the multiplier method is not mentioned in Saavedra-Moreno's (2011) paper, I had to implement my own formula which is currently based solely on the difference of the heights of all raster cells to the average height of the entire surface. However, I would like to point out that neither Saavedra-Moreno's (2011) nor my method of the linear wind speed multiplier were empirically validated and are only theoretical methods, which simplify real-world conditions strongly, for example: ignoring tunneling effects of the terrain or neglecting nonlinear relationships between altitude and wind speed.

The following illustration shows an example of the implemented concept on a random area with a mean elevation of 578 meters.

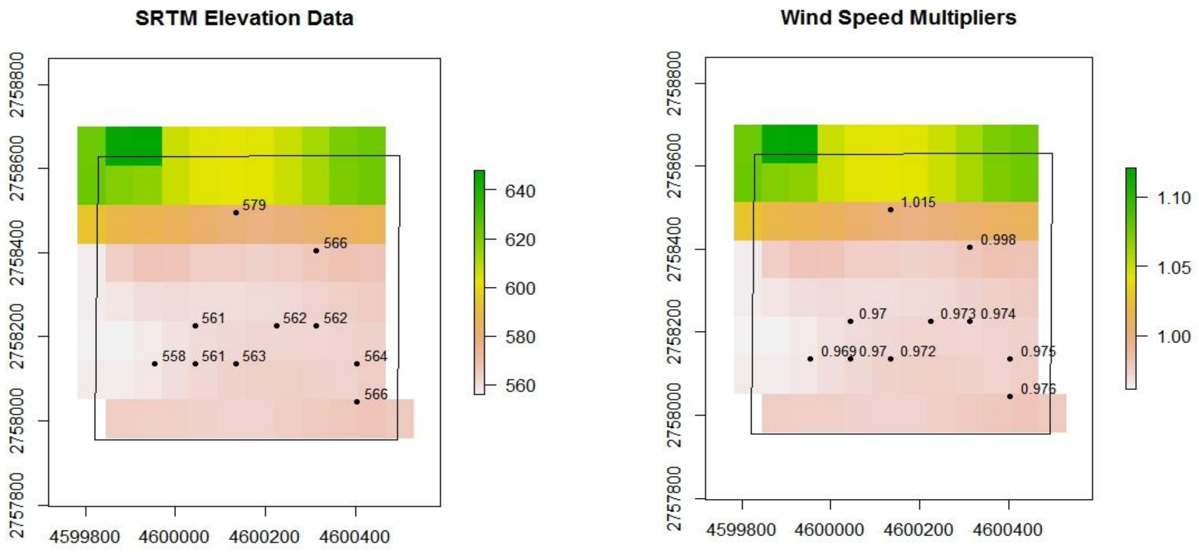


Figure 7: Left: SRTM elevation raster of a random area in Austria - Right: Calculated orographic influence raster with 10 random point examples

The northernmost point is at a slightly higher height (579m) than the average height of the total area (578m). Its corresponding wind speed multiplier is therefore greater than 1. All remaining points are below the average terrain height and have multiplier values less than 1. The wind speed multipliers of each turbine location are then multiplied by the previously calculated speed of the current wind sector. Turbines at higher altitudes contribute to higher expected energy outputs corresponding to "good" or "fit" individuals, which are therefore more frequently selected.

The altitude of an area has a further influence on the expected energy production that is evident in the wind power equation (Equ. 4). Wind power is proportional to air density, which in turn is dependent on height. In general, air density diminishes with increasing height and consequently the expected energy output decreases similarly. A wind farm in the mountains produces thus less energy compared to an identical wind farm at sea level. Since the conditions are most likely not identical and positive effects of wind farms at high elevations, such as greater wind speeds and better land availability should not be neglected, the negative effect of lower air density values should also be taken into account to model realistic conditions.

3.2.3.2.2. Corrected Air Density (*BaroHoehe*)

If the terrain effect model is not activated, 1.225 kg/m³ is taken as default value, representing air density at sea level according to the International Standard Atmosphere Model (Cavcar, 2005).

If it is enabled, the function "BaroHoehe" takes the height values of all wind turbines as input and calculates their corresponding air pressure, air density, temperature in Kelvin and temperature in degrees Celsius with the international barometric height formula, which is valid for the troposphere up to 11.000 meters. It also assumes that the temperature decreases with 6.5 Kelvin / 1000 meters (Hakenesch, 2016). The equations used for this method are described below with the following parameters:

- h – the height value
- p_h – the air pressure at height h
- p_0 – the standardized air pressure at sea level (101325 Pascal)

- r_h – the air density at height h
- R – the specific gas constant for dry air (287.058 J/kgK)
- $TempK$ – the temperature in Kelvin at height h
- $TempC$ – the temperature in degrees Celsius at height h

$$p_h = p_0 * \left(1 - \frac{0.0065*h}{288.15}\right)^{5.225} \quad \text{Equ. 15}$$

$$TempK = 288.15 - (0.0065 * h) \quad \text{Equ. 16}$$

$$TempC = 288.15 - (0.0065 * h) \quad \text{Equ. 17}$$

$$r_h = \frac{p(h)}{R*TempK} \quad \text{Equ. 18}$$

The air density correction in this method solely considers the elevation values and an estimation of the temperature, even though local conditions such as humidity or dryness of the regarded area also have an impact on air density. The method does not claim to be highly accurate and should only be a rough representation of local air conditions which can subsequently be included in the calculations.

As can be seen in the next illustration, with these formulas, air density falls almost to 0.7 kg/m³ at an altitude of 4000 meters and starts to be less than 1 kg/m³ at about 1800 meters.

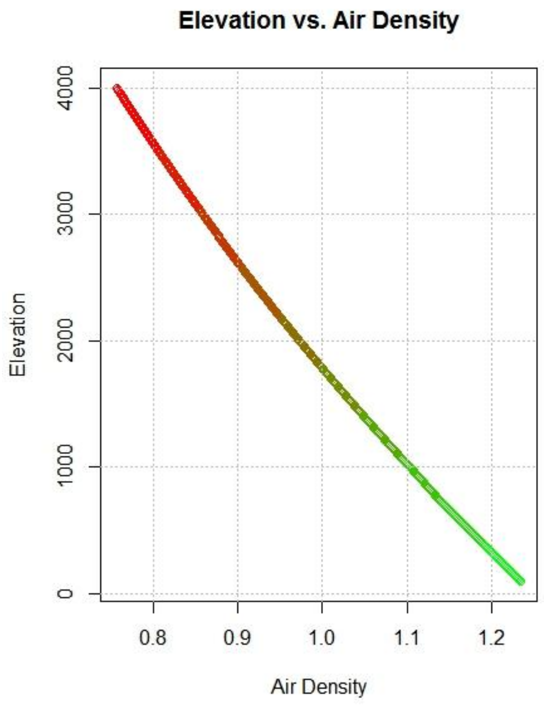


Figure 8: Relationship between elevation and air density according to the barometric height formula

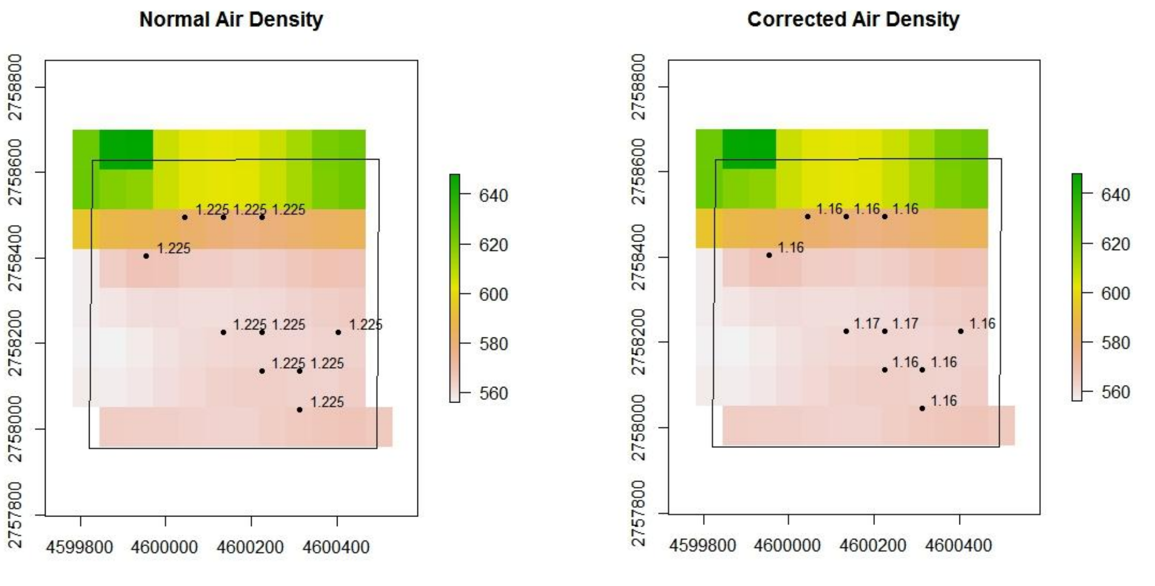


Figure 9: Left: Default air density values for a random area with mean elevation of 578 meters - Right: Corrected air density values according to barometric height formula

The corrected air density values drop in this example from 1.225 kg/m³ to about 1.16 - 1.17 kg/m³ for the 10 turbine locations in the wind farm and the expected energy output drops from ~12.830 kW with default air density values to about 12.000 kW with corrected air density values. This slight loss of about 6% may not seem high, but the wind farm is not located at very high altitude and larger losses will occur at higher altitudes due to air density reductions.

Another influential factor on local wind speed estimation is the roughness of the surface area. As mentioned earlier, wind speed decreases with decreasing height and increases with increasing height. This is mainly due to the surface roughness which boosts wind turbulence closer to the ground and therefore reduces the wind speed. Wind turbines are placed inside the atmospheric boundary layer that is prevailing to about 600 - 2000 meters as illustrated below, depending on the local weather and the time of the day (Schmelmer, 2013).

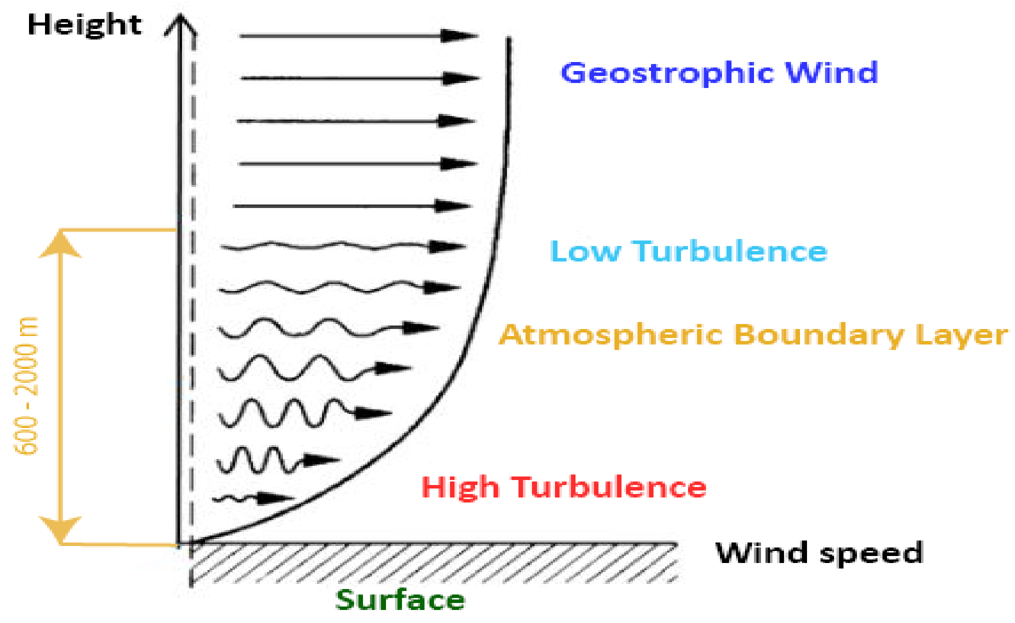


Figure 10: Atmospheric boundary layer with turbulence profiles (Liersch, 2012, p. 17)

The surface roughness length (z_0) is an indicator of the surface roughness and represents the height above the ground to which the wind speed due to obstacles is theoretically 0 (Schmelmer, 2013). The roughness length affects the curve of the wind speed in the figures below and therefore the recovery of wind speed with increasing height. Areas with no obstacles as water areas will have very low roughness lengths of 0.0002m and areas with high obstacles such as big towns will have roughness lengths of 1.6m (Solle, 2011).

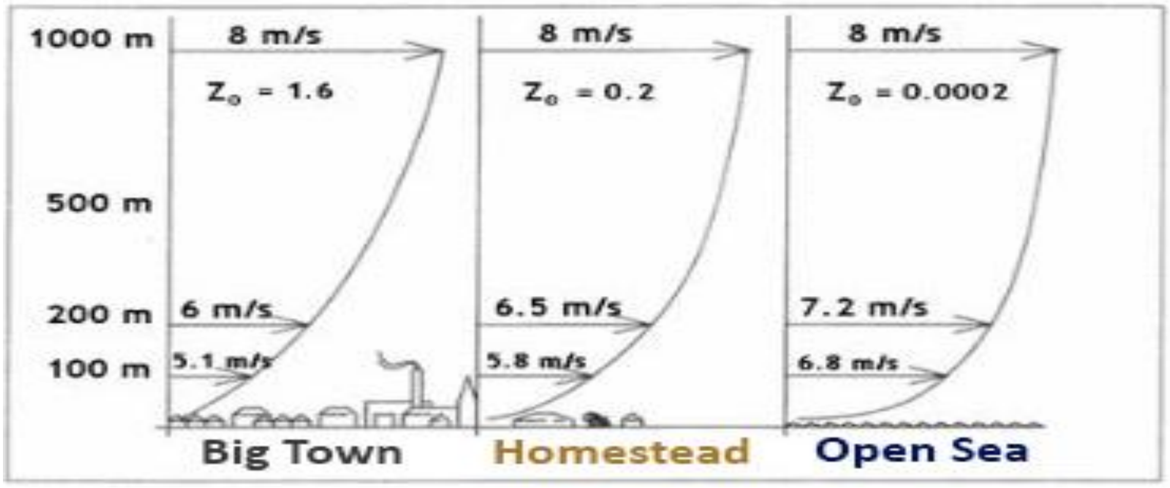


Figure 11: Wind speeds at different surface roughness lengths (z_o) (Solle, 2011, p. 17)

Roughness Length (z_0)	Brief s**urface type** description
0.0002	Water surfaces
0.0024	Open terrain with a smooth surface, e.g. Concrete, runways at airports, mown grass
0.03	Open agricultural terrain without fences and hedges, possibly with sparsely scattered houses, very gentle hills
0.055	Agricultural land with some houses and 8 meter high hedges with a distance of about 1250 meters
0.1	Agricultural land with some houses and 8 meter high hedges with distance of about 500 meters
0.2	Agricultural site with many houses, bushes, plants or 8 meters high hedges with a distance of about 250 meters
0.4	Villages, small towns, agricultural buildings with many or high hedges, forests and very rough and uneven terrain
0.8	Larger cities with tall buildings
1.6	Large cities, high buildings, skyscrapers

Table 4: Surface Roughness Lengths according to the European Wind Atlas (Schmelmer, 2013, p. 21)

3.2.3.2.3. Modified Surface Roughness Length z_{mod}

If the terrain model is activated, a surface roughness value is calculated for every turbine location, otherwise a default value of 0.14 is used which is a typical value for onshore wind farms and represents areas with a few lower obstacles. To estimate a more realistic roughness length the Corine Land Cover raster 2006 from the European Environment Agency is used (EEA, 2016). The raster has a resolution of 100 meters and covers Europe, as can be seen on the following R-plot of the raster.

It classifies the land cover in 44 categories, which are roughly divided into Artificial Surfaces, Agricultural Areas, Forest and semi natural areas, Wetlands and Water bodies.

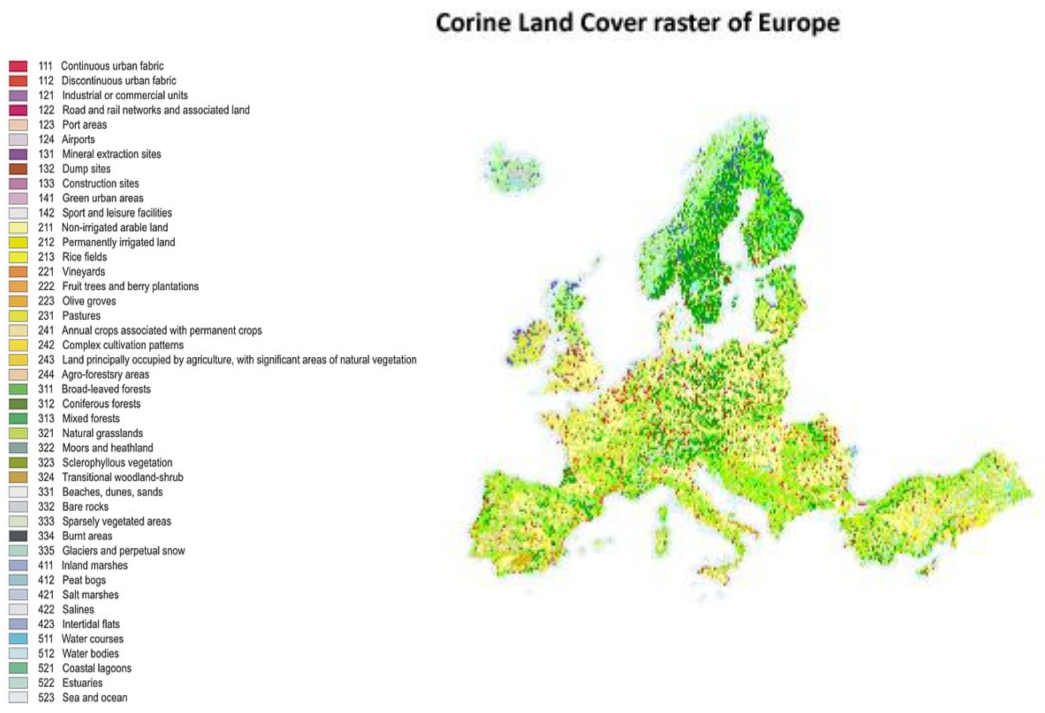

Corine Land Cover raster of Europe
111 Continuous urban fabric
112 Discontinuous urban fabric
121 Industrial or commercial units
122 Road and rail networks and associated land
123 Port areas
124 Airports
131 Mineral extraction sites
132 Dump sites
133 Construction sites
141 Green urban areas
142 Sport and leisure facilities
211 Non-irrigated arable land
212 Permanently irrigated land
213 Rice fields
221 Vineyards
222 Fruit trees and berry plantations
223 Olive groves
231 Pastures
241 Annual crops associated with permanent crops
242 Complex cultivation patterns
243 Land principally occupied by agriculture, with significant areas of natural vegetation
244 Agro-forestsry areas
311 Broad-leaved forests
312 Coniferous forests
313 Mixed forests
321 Natural grasslands
322 Moors and heathland
323 Sclerophyllous vegetation
324 Transitional woodland-shrub
331 Beaches, dunes, sands
332 Bare rocks
333 Sparsely vegetated areas
334 Burnt areas
335 Glaciers and perpetual snow
411 Inland marshes
412 Peat bogs
421 Salt marshes
422 Salines
423 Intertidal flats
511 Water courses
512 Water bodies
521 Coastal lagoons
522 Estuaries
523 Sea and ocean

Figure 12: Corine Land Cover Raster with a resolution of 100 m

The land cover information from this raster and the information in Table 4 are used to extend the raster with the surface roughness value (z_0). For any location in Europe, a land cover roughness value is subsequently available.

Several previous studies pointed out, that the roughness of a surface does not depend only on the land cover, but also on the topographic conditions of the wind farm. As wind conditions in complex terrain are still not well understood, no method was found which implemented topographic effects on the surface roughness.

The following method is a simple attempt to integrate topographic effects on the surface roughness estimation. The additional roughness indicator is calculated using the SRTM-elevation data and the "*terrain*" function from the "raster" library. With this function an elevation roughness indicator is calculated, which is defined as the difference between the maximum and the minimum elevation value of a cell and its 8 surrounding cells. The two roughness indicators are then combined to obtain a modified surface roughness value according to the following equation:

$$Z_{mod} = Z_{Land} * \left(1 + \frac{Z_{Elev}}{Res_{Elev}}\right) \qquad \text{Equ. 19}$$

- Z_{Land} – is the land type roughness indicator from the Corine raster
- Z_{Elev} – is the elevation roughness indicator from the SRTM raster
- Res_{Elev} – is the resolution of the SRTM raster (90m)
- Z_{mod} – is the modified surface roughness length

Unless the elevation roughness indicator is 0, the modified roughness value will always be higher than the original Corine roughness value which in turn will reduce the expected energy output. Flat terrain has low elevation roughness indicators, as the differences of the maximum and minimum elevation values are small and the land cover roughness will therefore be only slightly increased. Complex terrain on the other hand has high elevation roughness values which will increase the land cover roughness more heavily.

The above formula was calibrated only in Excel using random values and the reference data in Table 4 and was not empirically validated. Assuming that the difference between the maximum and minimum height (i.e. Z_{Elev}) of a particular raster cell to its 8 neighboring raster cells is 10m, then the multiplicative term in brackets is 1.11 (1 + 10/90 = 1.11), with which the land cover roughness is expanded. The formula therefore assumes a change in the land cover roughness by 1.1% for each meter of the elevation roughness. An elevation roughness of 500m would lead to an increase of 555.5%. Assuming the land cover roughness would be 0.03, the modified roughness value would be only 0.18, which would probably be too low. A value of about 1.6 would be necessary according to Table 4. The linear relationship of this formula is certainly imprecise, and presumably underestimates the influence of high elevation roughness while over-emphasizing the influence of low elevation roughness. An exponential correlation and the consideration of the desired turbine hub height might represent the influences more accurately. Both asperity values are demonstrated on the next illustrations on Figure 13 with the resulting modified surface roughness on Figure 14.

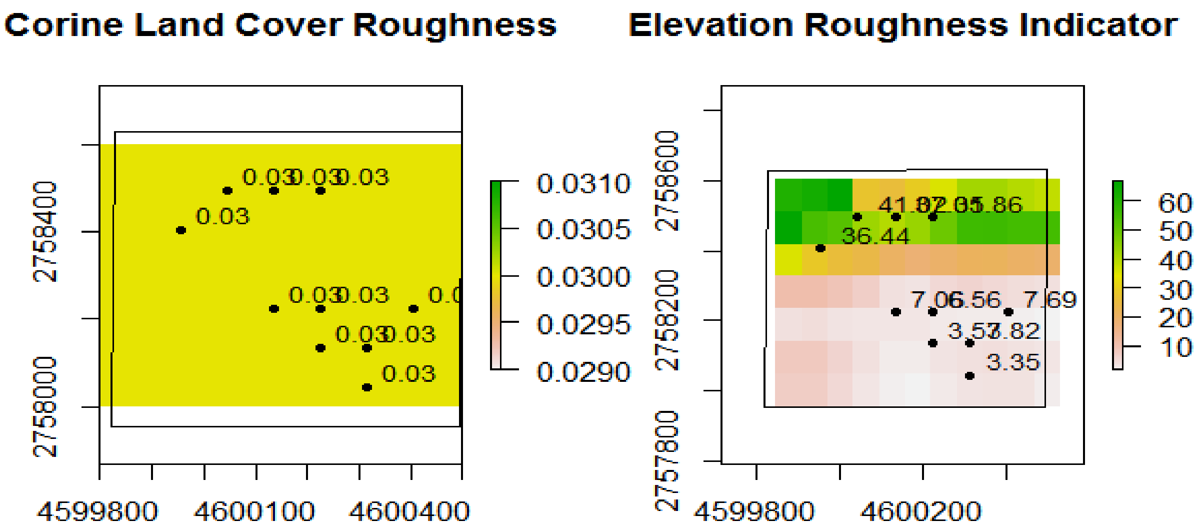


Figure 13: Land cover and elevation roughness indicator

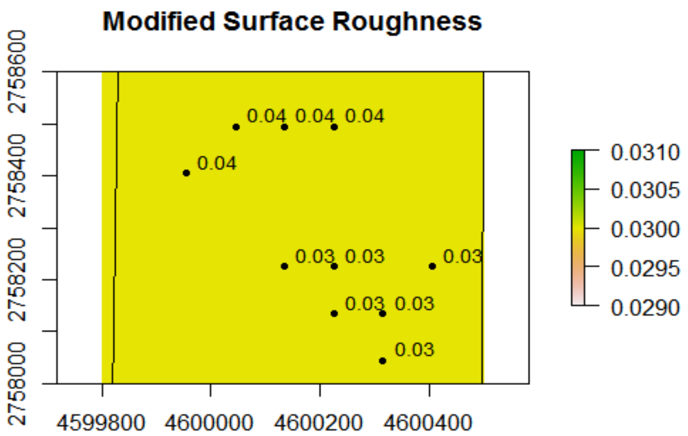


Figure 14: Modified surface roughness length

These figures show that the original and uniform Corine land cover roughness increases from 0.03 at the 4 northernmost locations to about 0.04 while the modified values of the southern sites are only slightly reduced and appear unchanged due to rounding.

3.2.3.2.4. Variable Wake Spreading Constant

The modified surface roughness has an influence on the wind profile law and on the wake spreading constant *K.* If the terrain model is not activated, a default value 0.075 is taken for *K*, which is a good reference for onshore wind farms and flat terrain. If the terrain model is activated, a variable wake spreading value *K* is calculated with the following equation (Chowdhury, et al., 2011):

$$K = \frac{0.5}{ln\left(\frac{RotorHeight}{z_{mod}}\right)} \qquad \text{Equ. 20}$$

The new wake spreading value changes the radius of the wakes and the velocity reductions.

After the desired evaluation of terrain effects, the algorithm jumps back to the "*calculateEn"* function, where energy outputs are finally calculated. For each given wind direction, the shape file and the turbine coordinates are aligned to the north since the algorithm can only calculate wake effects under this direction. The wind turbines that could potentially influence others are evaluated with the functions "*InfluPoints*" and "*VekWinkelCalc*". The latter function calls 2 other functions "*WinkelCalc*" and "*PointToLine2*" which calculate the relevant distances and angles for the wake evaluations. These 4 functions are not fully described in this work, as their only task is to draw triangles, calculate distances and angles, and then evaluate which turbines will certainly not affect others because their angles or distances are too large. With those resulting inputs, the wake effects are estimated and the energy output is calculated.

Amongst others, the resulting energy outputs and efficiency rates for all given wind directions are stored as a list. The amount of wind directions gives the length of the list. An exemplary output of this function for one wind direction is shown below.

```
> str(calculateEn_Output)
List of 1
 $ :Classes ‘tbl_df’, ‘tbl’ and 'data.frame':17 obs. of
18 variables:
  ..$ Punkt_id          : int [1:17] 1 2 3 4 5 5 6 7 7 8
  ..$ Ax                : num [1:17] 0 0 4600280 4600280
  ..$ Ay                : num [1:17] 0 0 2758554 2758554
  ..$ Bx                : num [1:17] 4599954 4600044 4600257
  ..$ By                : num [1:17] 2758495 2758495 2758354
  ..$ Laenge_B          : num [1:17] 0 0 200 200 200
  ..$ Laenge_A          : num [1:17] 0 0 23 23 23
  ..$ Windrichtung      : num [1:17] 0 0 0 0 0 0 0 0 0 0
  ..$ Windmean          : num [1:17] 9 9 9 9 9 9 9 9 9 9
  ..$ RotorR            : num [1:17] 30 30 30 30 30 30 30 30
  ..$ WakeR             : num [1:17] 0 0 45 45 45
  ..$ A_ov              : num [1:17] 0 0 2499 2499 2499
  ..$ TotAbschProz      : num [1:17] 0 0 88.4 88.4 188.4
  ..$ V_New             : num [1:17] 9 9 6.69 6.69 4.45
  ..$ Rect_ID           : num [1:17] 1 2 5 9 11 11 12 13 13
  ..$ Energy_Output_Red : num [1:17] 3487 3487 3487 3487 3487
  ..$ Energy_Output_Voll: num [1:17] 7334 7334 7334 7334 7334
  ..$ Parkwirkungsgrad  : num [1:17] 47.5 47.5 47.5 47.5 47.5
```

R Output 5: Exemplary output of the function *calculateEn* for one wind direction

After the energy outputs of all individuals in the current population have been evaluated, the algorithm returns to the fitness function, where the information of an individual is transformed to a single fitness value.

If there are several wind directions per individual, there are also several energy outputs. To get a single energy output value considering all wind directions and speeds, the expected energy outputs of all wind directions and speeds are multiplied with the corresponding wind probability. The weighted energy outputs are then summed up to a single energy output value. To get a single efficiency rate for all wind directions, the same procedure is applied.

The following example illustrates this process. For an exemplary individual with 2 wind directions, the expected energy outputs are 3500 kW with an efficiency rate of 45% for wind direction 0° degrees, which will occur to 30%, and 5000 kW with an efficiency rate of 70% for wind direction 290° degrees, which will occur to 70%.

```
Windrichtung Energy_Output_Red Parkwirkungsgrad probabDir
           0              3500               45        30
         290              5000               70        70
```

R Output 6: Exemplary energy and efficiency outputs for 2 input wind directions

The energy outputs are multiplied by their share in total production. In this example 3500 is multiplied by 0.3 and 5000 is multiplied by 0.7, resulting in 1050 and 3500 kW, as shown below.

```
Windrichtung Energy_Output_Red Parkwirkungsgrad probabDir Eneralldire
           0              3500               45        30        1050
         290              5000               70        70        3500
```

R Output 7: Exemplary weighted energy outputs of an individual

The weighted energy outputs are summed up to 4550 kW (1050+3500) and the efficiency rates are likewise multiplied by their share in total production and summed up, resulting in 62,5% as (45*0,3)+(70*0,7) = 62,5.

Such a weighted energy output is calculated for each individual and assigned as their fitness score. In other genetic algorithms, the fitness of an individual is often proportional to the fitness of the whole population. As the population size is not constant in this method, a proportional approach would lead to incomparable values. After experimenting with several fitness formulas, the unmodified weighted energy output seemed to be a good choice.

To contrast two successive generations numerically, this algorithm calculates a comparative fitness score with values of the current and previous generation as follows:

$$FitGen = 0{,}8 * (max_{t0} - max_{t-1}) + 0{,}2 * (mean_{t0} - mean_{t-1}) \quad \text{Equ. 21}$$

The relevant input parameters are:

- max_{t0} – is the maximum fitness value of the current population
- max_{t-1} – is the maximum fitness value of the previous population
- $mean_{t0}$– is the mean fitness value of the current population
- $mean_{t-1}$– is the mean fitness value of the previous population

This equation is the result of several tests in the implementation process, where I observed that the maximum fitness values often remained constant, which would indicate stagnation. Another value had to be included to better indicate an improvement or degradation of the evolutionary process. Since the minimum fitness values can and should sometimes be low due to mutation processes, they are ignored in this equation and instead the difference of the average fitness values are taken and weighted with 20%.

This comparative generation fitness value is used in the variable selection method and in the crossover function to determine the number of crossover parts.

3.2.4. Selection of Individuals (*selection1*)

This function deletes the worst 4 individuals from the population, based on the fitness values calculated in the previous chapter and selects a certain amount of individuals but at maximum 100. The roulette-wheel selection method is used to find out which individuals should be selected. The probability that an individual is selected is proportional to its fitness value. Those selected individuals then form pairs of parents and recombine their genetic information through a crossover function.

The two possible input values "FIX" and "VAR" for the variable *selstate* determine the percentage which is selected from the current population. It can be either at a fixed or at a variable percentage. The variable percentage has an upper bound of 75% and a lower bound of 20%. If the population consists of less than 20 individuals, the selection percentage is set to 100% to keep all individuals in the population, produce more offspring, and prevent the population from extinction. Furthermore, in the case of less than 20 individuals, the algorithm increases the crossover point rate by 0.07, which is described in chapter 3.2.5.

3.2.4.1. Fixed Selection

If the selection method is fixed (*selstate* = "FIX"), a constant value of 50 % is selected from the current population.

3.2.4.2. Variable Selection

If the selection method is variable (*selstate* = "VAR"), the percentage of selected individuals changes and is dependent on the performance of the current population, compared to the previous one. This comparative performance indicator is calculated by Equ. 21.

The percentage starts at ~75% and decreases by about 0.8% when the fitness values of the current population are better than the previous ones. Consequently, fewer individuals will be selected for the next generation. If the current population performs worse than the previous, the percentage increases by about ~0.5%, and more individuals will be selected. The idea behind this concept is that as long as the fitness values during the evolution rise, the selection pressure on the population should rise likewise, to exploit

good / fit individuals and try to converge to an optimum in the search space. If the fitness values drop, the selection pressure is reduced likewise and more individuals will be selected for the next generation. This is known as exploration or the spreading of individuals over the whole search space, to increase the chance of finding the global optimum (Kruse & Held, 2013).
If the selection percentage is lower than 20%, the algorithm seems to have converged to a small amount of individuals which will reduce the variance and the population size. The crossover point rate will then be increased by 0.09 as a counter-balancing effect. This should increase the crossover parts and thus in turn the variance and the population size.

3.2.4.3. Elitism

According to Scholl (2002) the implementation of an elitism selection will lead to a global optimum if the amount of generations goes to infinity. The idea of elitism is to save the best individuals separately and add them again to the population after the mutation and crossover methods, so random effects do not influence their genetic information.
The implemented elitism method in this algorithm differs from the original idea. If the input variable *elitism* is "TRUE", which is the default, elitism is activated, and the input variable *nelit* determines how many individuals should be part of the elite section. The default for *nelit* is 6 individuals. This elite section is not saved separately, but its fitness values are increased by a factor of 10, mainly to simplify implementation. The probability that they will be selected is therefore much higher than for the remaining individuals, although the selection process will still be random. If *elitism* is "FALSE", this method is not activated, the input variable *nelit* is not needed and the fitness values will not be increased.

Regardless of whether or not elitism is taken into account, the algorithm is extended with a kind of top-level guarantee. After the first 20 interactions, the algorithm compares the current maximum energy value with all previous maximum values. If 7 iterations in succession result in a worse maximum value than the best solution so far, then in the following generation 2 of the hitherto best individuals are integrated into the population. This top-level guarantee was added belated based on test results, where the fitness values crashed due to mutation or crossover effects and the algorithm sometimes was not able to converge to his previous best result. It can be regarded as a conditioned seeding method with previously found best individuals of the current optimization process.

Every individual from the selected parents has genetic information that is passed to the crossover function. The information encoded in an individual consists of a binary variable for every grid element, 0 represents no turbine in the grid cell and 1 represents a turbine in the grid cell. Separately the fitness values of all parents are passed to the crossover function as well.

3.2.5. Crossover of Genetic Information (*crossover1*)

To recombine those genetic codes and create new individuals, n crossover-points are generated, either at equal or random intervals, where the genetic code is cut in pieces. The amount of distinct combinations is given by the formula for permutations with repetition and order n^k, where n is the sum of individuals in a parental group and k is the amount of crossover parts. For example, $2^5 = 32$ recombinations are generated from 2 individuals forming a parental group, each composed of a pentamerous genetic code.

In this algorithm, the crossover point rate determines the amount of crossover parts. As the amount of crossover parts always has to be an integer, the crossover point rate is truncated to the nearest integer towards 0. This integer determines the amount of locations at which the genetic code is cut in pieces. The amount of genetic code pieces or crossover parts will therefore always be 1 unit bigger than the amount of crossover points, which is illustrated below.

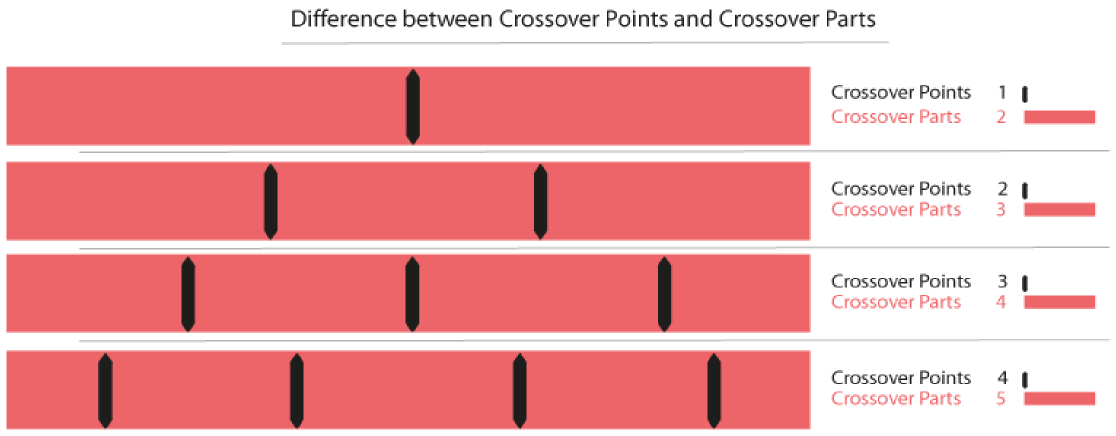


Figure 15: Illustration of four cases, where the genetic code is cut with increasing crossover points and resulting crossover parts

The algorithm starts with a crossover point rate of 1.1 and therefore 2 crossover parts, but is able to change according to the evolution of fitness values in Equ. 21. If the compared fitness values of the current population are greater than the previous population, then the crossover point rate is increased by 0.03 and decreased by 0.06, when the fitness values are smaller. These defined parameters are based on several tests in the implementation process in which they have proved to be useful. Besides that, the algorithm showed best converging abilities with 2 crossover parts, which were also taken as starting conditions. If the fitness values of a hypothetical optimization process would increase 34 times in a row, the crossover point rate would increase by 1.02 (34*0.03=1.02), and the amount of crossover parts would increase by 1.

Respectively, the amount of crossover parts would decrease by 1, when the fitness values would decrease 17 times in a row (17*0.06=1.02).
The minimum of crossover parts is 1, which means that the crossover method is not able to generate new individuals. If for example 100 individuals form 50 parents, each composed of 2 individuals, and then cross their information with 1 crossover part, the amount of distinct permutations is $50*2^1 = 100$. The resulting 100 individuals are just copies of the 100 parental individuals. The variable selection percentage will in this case be increased by about 15%.
As the amount of possible permutations increase exponentially with increasing crossover parts and the proposed algorithm should produce good results in short time, this method has an upper limit of 300 possible permutations in a population. Considering again the case of 100 individuals, forming 50 parental groups with 3 crossover parts, $50*2^3$ or 400 permutations are possible, but only 300 will be selected. If from those 400 possible permutations 300 or 75% would be randomly selected and further evaluated, the loss of information would be significant. To address this problem, the fitness values of all individuals are passed to this function. In the crossover method, the mean fitness value of each parental group is first evaluated. Then each parental fitness value is divided by the mean fitness value of all parental groups. The fitness indicator of a parental group is then assigned to all permutations created from this parental group. The illustration below shows the fitness values of 24 random individuals with a mean fitness value of about 8400, indicated by the horizontal red line on the left. The recalculated parental fitness values on the right plot are between 1.1 and 0.9 and the new mean is 1, indicated by the red line on the right.

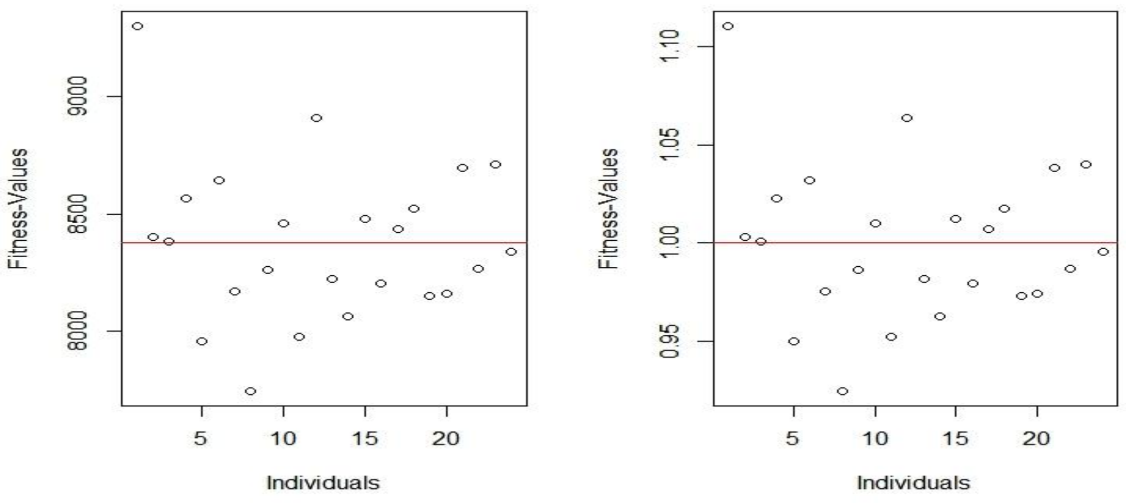


Figure 16: Original and rearranged fitness value for crossover operation

If the possible permutations exceed 300, this function selects exactly 300 individuals based on their recalculated parental fitness values to ensure that good individuals are very unlikely to be deleted.

The maximum of crossover parts is 5. For 50 parents with 5 crossover parts the amount of permutations is $50*2^5 = 1600$. Because of the high amount of distinct permutations with 5 crossover parts, the selection percentage would drop to 25% of the current population, so that only the fittest individuals generate new permutations. However, the algorithm most certainly will not reach 5 crossover parts under the actual parameters and will peak at a maximum of 3 crossover parts. This is mostly because of the higher crossover point rate reduction of 0.06 compared to the crossover point rate increment of 0.03 and secondly, because of the increase in variance and the possible loss of information under 3 crossover parts. As the number of individuals grows rapidly with 3 crossover parts and the permutations have an upper bound of 300, the algorithm will start losing information with more than 38 parents ($38*2^3=304$). The loss of information with 3 crossover parts and more parents will therefore grow and the evolution of fitness values will most certainly not only rise, leading to more crossover point rate reductions. The condition under 3 crossover parts is with current parameters consequently not stable and likely to fall back to 2 crossover parts due to the aforementioned arguments.

In the following two subchapters, the 2 possible crossover methods "EQU" and "RAN" are discussed, that determine where the genetic code is cut in pieces.

3.2.5.1. Equal Crossover Partitioning

If the crossover points should be located at equal intervals, the input variable *crossPart1* has to be set to "EQU" (crossPart1="EQU"). The length of the genetic code is divided by the number of crossover parts, resulting in the number of genes per crossover part. For example, an individual with a genetic code of 36 genes and 2 crossover points will lead to 3 crossover parts, each having 12 genes, as shown in the R-Output below. The variable "*t1*" is in this case 12.

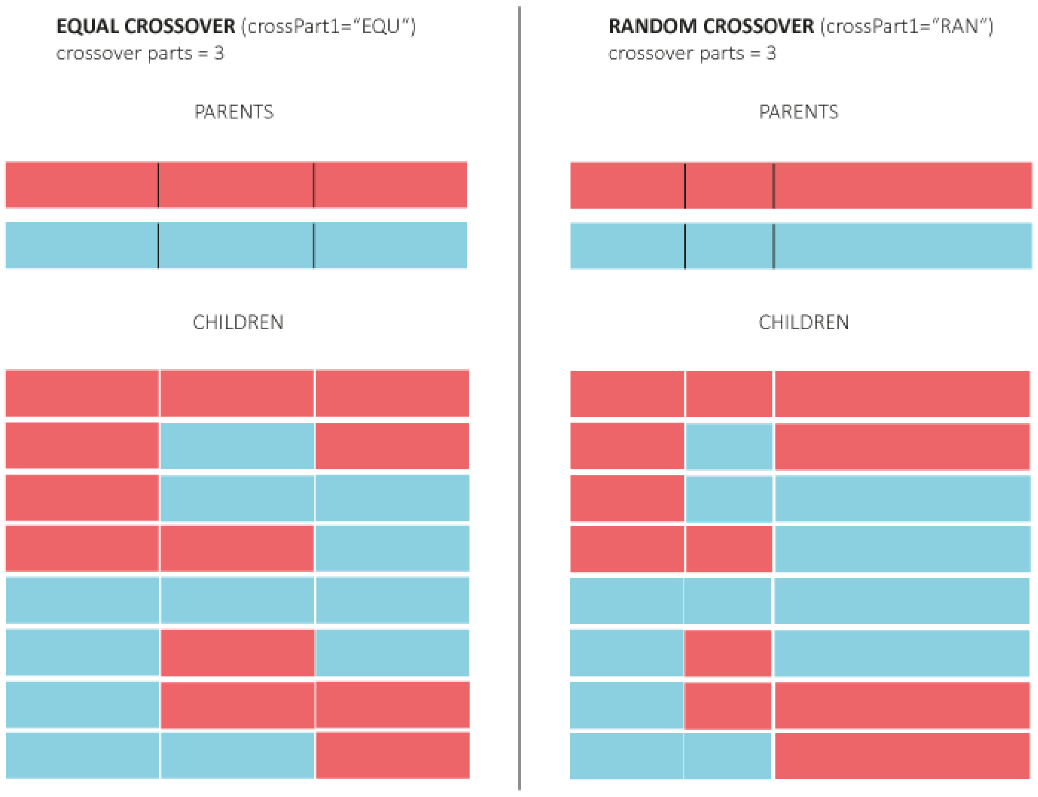
```
> Indiv
[1] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0

>GenParts<-split(Indiv,as.numeric(gl(length(Indiv),t1,length(Indiv))))
$`1`
[1] 1 0 0 0 0 0 0 0 0 0 0 0
$`2`
[1] 0 0 0 1 1 0 0 0 1 1 0 1
$`3`
[1] 0 1 0 0 0 0 1 0 0 1 1 0
```

R Output 8: Exemplary 2 point equal crossover with an individual consisting of 36 genes

3.2.5.2. Random Crossover Partitioning

If the crossover points should be located at random locations, the input variable *crossPart1* has to be set to "RAN" (crossPart1="RAN"). The following Figure 17 illustrates the difference between the two partitioning methods. The red and blue bars represent the genetic code of 2 parental individuals, the strings of 0`s and 1`s representing the wind turbines in this algorithm. The black lines in the genetic code of the parents represent the crossover points, where the genetic code is split. As explained before, 2 crossover points result in 3 crossover parts. For 2 parents with 3 crossover parts, the amount of children is 8 ($2^3 = 8$) which are illustrated below.



EQUAL CROSSOVER (crossPart1="EQU")
crossover parts = 3
PARENTS
CHILDREN
RANDOM CROSSOVER (crossPart1="RAN")
crossover parts = 3
PARENTS
CHILDREN

Figure 17: Comparison of Equal and Random Crossover for 2 parental individuals with 3 crossover parts

While Equal Crossover always cuts the individuals at identical locations, assuming that the number of crossover parts remains the same, Random Crossover generates new random crossover point locations for every new parental group. The actual method works only with one crossover method at a time. A mixing strategy could be interesting, as I assume that with Equal Crossover the populations will more rapidly converge to optima, while with Random Crossover, the variance between the individuals and populations will be higher and the evolution will not be as steady. The idea of exploration and exploitation of the search space is again noticeable for both crossover methods. This could be integrated in a mixed method. Ceteris paribus, Equal

Crossover will more likely exploit the search space because of invariable conditions and Random Crossover can be more classified as an exploration method, as differing crossover locations will more likely result in different permutations and individuals.

The generated permutations of this function represent already the new children of the next generation, whereby their genetic information is still left to the mutation function.

3.2.6. Mutation of Genetic Information (*mutation*)

The mutation function randomly changes certain genes of the individuals to avoid premature convergence to local optima (Williams, 2014). The mutation probability or rate (*mutr*) of a gene should be considered quite small, often ranging from 0.01 - 0.1. The default for the mutation rate in this algorithm is 0.008. It is comparatively low, mainly because the selection and crossover methods have other random effects, which have already been addressed and which are discussed in more detail in chapter 5.

3.2.6.1. Fixed Mutation Rate

In the implementation process, the algorithm was tested with a fixed mutation rate of 0.01 and a uniform wind direction. This often did not lead to optimal results even for small problems, as the populations seemed to be stuck in local optima, near the global optimum. In most cases, only 1 or 2 turbines would have to be displaced to achieve the global optimum. The mutation rate was too low in those cases to explore the relevant search space, and a higher mutation rate of 0.1 often was already too high to converge to any relevant optima. A fixed mutation rate only therefore did not seem to be the right choice.

3.2.6.2. Variable Mutation Rate

The fixed mutation rate is supplemented with a variable mutation rate. During the evolution, good individuals reproduce more, resulting in similar and identical good results or individuals in a population. The more generations pass by, the more individuals will represent the actual best solution. The variable mutation rate comes into action when more than 3 individuals embody the current best solution and when those individuals are identical. A new mutation rate is then randomly generated between 0.03 and 0.1, and multiplied by a factor that is proportional to the amount of iterations and identical best solutions, which at minimum are 3.

$$vmutr = \left(mutRan * \left(1 + \frac{(nOpti*1.25)}{42}\right)\right) + \left(\frac{i}{20*iteration}\right) \qquad \text{Equ. 22}$$

- $vmutr$ – is the variable mutation rate
- $mutRan$ – is the random mutation rate between 0.03 and 0.1
- $nOpti$ – is the number of current best and identical individuals
- i – is the current iteration of the running optimization process
- $iteration$ – is the total amount of requested iterations

The observed variable mutation rates of pre-tests range somewhere between 0.037 and 0.2, and are significantly higher than the fixed mutation rate of 0.008. When several individuals represent the actual best solution, the algorithm seems to have converged to an optimum. As the algorithm is not able to see if this optimum really represents the best solution or the global optimum, it increases the mutation rate strongly to increase the variance between the individuals, explore the search space and probably lose the actual best solution to increase the chance of finding other good solutions and the global optimum.

The crossover and the mutation function change the number of turbines in the wind farms. The proposed method is however designed to operate with a constant number of turbines. This is mainly because the fitness value of an individual is the expected energy output. If the number of turbines were not regulated, the algorithm would continue to add new turbines to a wind farm, which would lead to fitter individuals, since they would have higher expected energy outputs, albeit at lower efficiency rates. The proposed algorithm could be configured with some adaptations so that a variable amount of turbines would be possible and useful. A possible adjustment would be to determine the fitness value of an individual in proportion to its efficiency rather than its energy yield or the algorithm could be expanded by a restrictive economical or technical function in the case of multi-objective optimization.

The proposed method optimizes as single criterion the energy output of a wind farm with a constant number of turbines. A function is therefore required which addresses the problem of deviating numbers of wind turbines that can arise due to the altering effects of crossover and mutation.

3.2.7. Adjust to n-desired Turbines (*trimton*)

The function *trimton* adjusts the number of wind turbines to the preassigned number. It checks every individual and counts the total amount of 1's, representing wind turbines in its genetic code. The total count of turbines can be lower or higher than the required quantity of turbines. Depending on the difference, this function either adds or removes wind turbines from the current wind farm. There are 2 available methods to achieve this task, which can be set with the input variable "*trimForce*".

3.2.7.1. Random Adjustment

If "*trimForce*" is set to "FALSE", the adjustment process is random, which is the default. For every individual the sum of current wind turbines is subtracted by the preassigned number of turbines. If this leads to a positive integer, then too many wind turbines are within the current wind farm, and the integer specifies the amount of excessive turbines that must be removed from the existing turbines. If the result is a negative number, additional grid cells that are not present in the current wind farm configuration are selected in which a turbine is then placed. The absolute of the resulting negative number indicates how many turbines have to be selected, so that the wind farm then consists of the required number of turbines. With this adjustment method, also good located turbines, that have few total wake impacts or are part of a wind farm with high fitness value, may be deleted.

The other available adjustment method, described next, attempts to overcome this problem.

3.2.7.2. Probabilistic Adjustment

This method is activated if the input variable "*trimForce*" is set to "TRUE". The probabilistic adjustment uses the fitness and total wake information of all individuals in the current population. All individual genes are grouped together according to their grid cell ID. The mean fitness value and mean total wake value of all evaluated grid cells are calculated. If a grid cell is not used in the current population, it receives the 25% quartile wake effect and the average fitness value of the whole population. With a fitness and a total wake value for every grid cell, two new variables are calculated. They represent the likelihood that a grid cell is deleted if there are too many turbines in the wind farm, or that a grid cell is selected if the wind farm does not have enough wind turbines. The two probability variables are calculated as follows:

$$P_D = \frac{1}{\frac{\left(1+\frac{Wd_{(i)}-Wd}{1+Wd_{(i)}}\right)}{\left(1+\frac{Fd_{(i)}-Fd}{1+Fd_{(i)}}\right)^k}} \qquad \text{Equ. 23}$$

$$P_S = \frac{\left(1+\frac{Ws_{(i)}-Ws}{1+Ws_{(i)}}\right)}{\left(1+\frac{Fs_{(i)}-Fs}{1+Fs_{(i)}}\right)} \qquad \text{Equ. 24}$$

The input variables for those equations are listed and described below:

- P_D – is the resulting deletion probability of the grid cells that are used in the current individual. This variable is used for wind farms with excessive wind turbines.
- Wd – is the mean wake effect of a grid cell respective to all individuals. This variable is evaluated only for grid cells that are used in the current individual.
- $Wd_{(i)}$ – is the maximum wake effect of all grid cells which are used in the current individual
- Fd – is the mean fitness value of a grid cell respective to all individuals. This variable is evaluated only for grid cells that are used in the current individual.
- $Fd_{(i)}$ – is the maximum fitness value of all grid cells which are used in the current individual
- P_S – is the resulting selection probability of the grid cells which are not used in the current individual, but in the rest of the population. This variable is used for wind farms with insufficient wind turbines.

- $Ws -$ is the mean wake effect of a grid cell respective to all individuals. This variable is evaluated only for grid cells that are not used in the current individual.
- $Ws_{(i)}$ – is the maximum wake effect of all grid cells which are not used in the current individual
- $Fs -$ is the mean fitness value of a grid cell respective to all individuals. This variable is evaluated only for grid cells that are not used in the current individual.
- $Fs_{(i)}$ – is the maximum fitness value of all grid cells which are not used in the current individual
- $k -$ is a weighting parameter with a constant value of 3

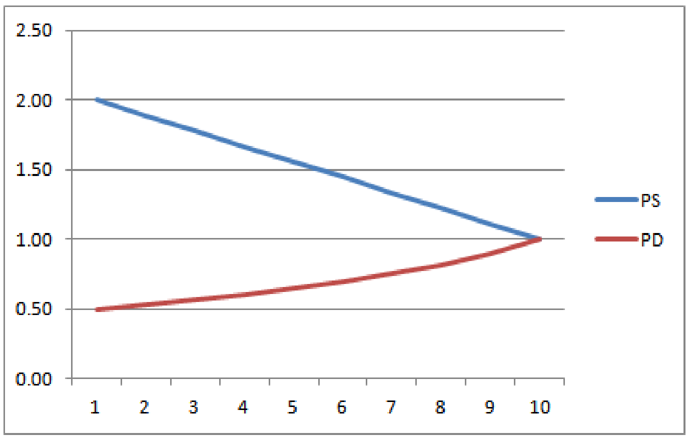
The two probabilities are illustrated next in 4 different and abstract cases. The exemplary wind farm for the calculations consists of 10 grid cells, where random values are used for the "Mean Wake Effect" and "Mean Fitness Value". All variables are illustrated in a traffic light scheme, with red indicating "bad" and green indicating "good". Variable k was set to 15 for these illustrations.

Case 1

Assumption 1:

Inputs			Outputs	
Grid Cell	Mean Wake Effect	Mean Fitness Value	PS	PD
1	0	6000	2.00	0.50
2	100	6000	1.89	0.53
3	200	6000	1.78	0.56
4	300	6000	1.67	0.60
5	400	6000	1.55	0.64
6	500	6000	1.44	0.69
7	600	6000	1.33	0.75
8	700	6000	1.22	0.82
9	800	6000	1.11	0.90
10	900	6000	1.00	1.00

Increasing Wake Effect, Fixed Fitness Value



Description: All grid cells have the same fitness value, but different and increasing wake effects.

PS: The lower the expected wake effects of a grid cell, the higher the chance that this grid cell will be selected if wind turbines are missing in the wind farm and, the higher the wake effect, the lower the probability of the grid cell to be selected.

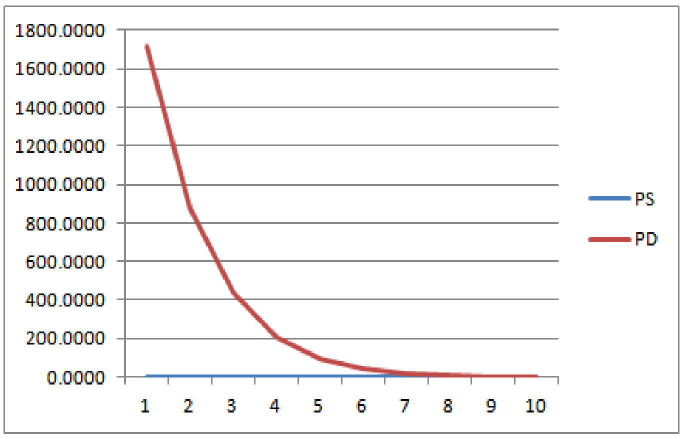
PD: The lower the wake effect of a grid cell, the lower the probability that this grid cell will be deleted if too many turbines are in the wind farm. The higher the wake effect, the higher the chance to be deleted.

Case 2

Assumption 2:

Inputs			Outputs	
Grid Cell	Mean Wake Effect	Mean Fitness Value	PS	PD
1	100	5000	0.0006	1713.2271
2	100	6000	0.0011	879.5296
3	100	7000	0.0023	437.7375
4	100	8000	0.0047	210.5668
5	100	9000	0.0103	97.5576
6	100	10000	0.0231	43.3574
7	100	11000	0.0544	18.3963
8	100	12000	0.1350	7.4100
9	100	13000	0.3553	2.8146
10	100	14000	1.0000	1.0000

Fixed Wake Effect, Increasing Fitness Value



Description: All grid cells have the same wake effect, but different and increasing fitness values.

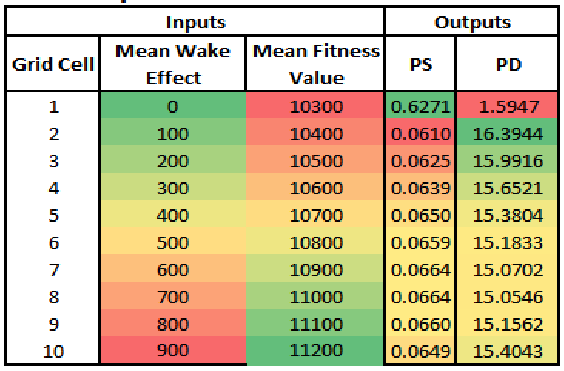
PS: The lower the fitness value of a grid cell, the lower the chance that this grid cell will be selected if wind turbines are missing in the wind farm.

PD: The lower the fitness value of a grid cell, the higher the probability that this grid cell will be deleted if too many turbines are in the wind farm.

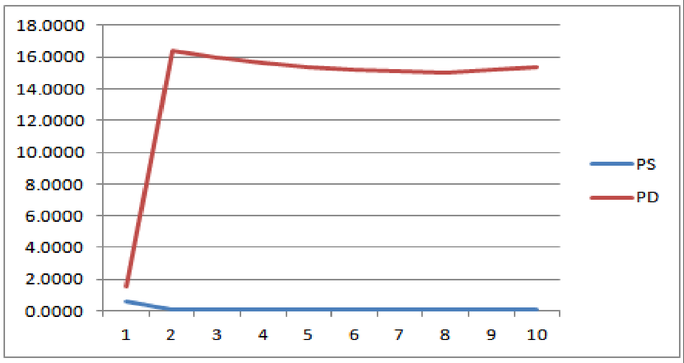
Figure 18: Adjusting *trimton*-probabilities PS and PD for cases 1 & 2

Case 3

Assumption 3: **Increasing Wake Effect, Increasing Fitness Value**



Inputs			Outputs	
Grid Cell	Mean Wake Effect	Mean Fitness Value	PS	PD
1	0	10300	0.6271	1.5947
2	100	10400	0.0610	16.3944
3	200	10500	0.0625	15.9916
4	300	10600	0.0639	15.6521
5	400	10700	0.0650	15.3804
6	500	10800	0.0659	15.1833
7	600	10900	0.0664	15.0702
8	700	11000	0.0664	15.0546
9	800	11100	0.0660	15.1562
10	900	11200	0.0649	15.4043



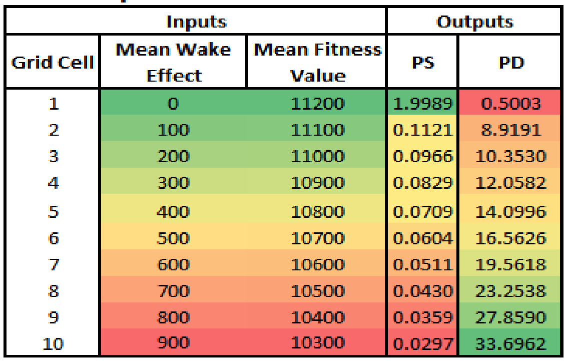
Description: The grid cells have increasing wake effects and increasing fitness values. Although this relation is very unlikely, it is used to illustrate two special issues. The parameter *k* will play a big role in this case, as it increases the weight of the fitness values. If the fitness values vary very strong, the parameter *k* should be quite small, if the fitness values are close together a higher parameter *k* is needed.

PS: The idea is that, some wake effects on a grid may be high, but may correspond to a wind farm constellation, that achieves a high fitness value. If a grid cell achieves a high fitness value, but has high expected wake effects, its probability to get selected, would still be considerably as can be seen for grid cell 10 in this case with *k* = 15.

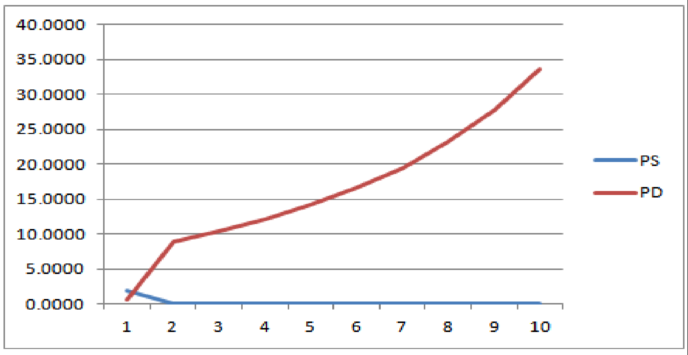
PD: If a grid cell has a low wake effect and a low fitness value, the probability to be deleted will be high. This does not count for the grid cell with the minimum wake effect. Although grid cell 1 has the lowest fitness value, the probability to be deleted will be very low.

Case 4

Assumption 4: **Increasing Wake Effect, Decreasing Fitness Value**



Inputs			Outputs	
Grid Cell	Mean Wake Effect	Mean Fitness Value	PS	PD
1	0	11200	1.9989	0.5003
2	100	11100	0.1121	8.9191
3	200	11000	0.0966	10.3530
4	300	10900	0.0829	12.0582
5	400	10800	0.0709	14.0996
6	500	10700	0.0604	16.5626
7	600	10600	0.0511	19.5618
8	700	10500	0.0430	23.2538
9	800	10400	0.0359	27.8590
10	900	10300	0.0297	33.6962



Description: The grid cells have increasing wake effects and decreasing fitness values. This case is the most probable.

PS: A grid cell with a low wake effect and a high fitness value will get a high probability to be selected. Contrary, a grid cell with a high wake effect and a low fitness value will have a low probability of being selected.

PD: A grid cell with low wake effects and high fitness value will have a low probability of being deleted and vice versa, a grid cell with high wake effects and low fitness values will have a high chance to be deleted.

Figure 19: Adjusting *trimton*-probabilities PS and PD for cases 3 & 4

After every individual is adjusted to the desired number of wind turbines, the genetic information of all individuals is then passed to the function *getRects*.

3.2.8. Get Grid-ID`s from Genetic Information (*getRects*)

The result of the previous function is a matrix of 0's and 1's, which represents the genetic information of all individuals. This matrix is recoded as a list of different wind farms with the particular grid coordinates and grid-ID's.

```
> mut1[,1:3]
      [,1] [,2] [,3]
 [1,]    0    0    0
 [2,]    0    0    1
 [3,]    0    0    0
 [4,]    0    0    1
 [5,]    0    0    0
 [6,]    1    1    1
 [7,]    0    0    0
 [8,]    0    0    0
 [9,]    0    0    0
[10,]    0    0    0
[11,]    0    0    0
[12,]    1    1    1
[13,]    0    0    0
[14,]    1    1    0
[15,]    0    1    0
[16,]    0    0    0
[17,]    0    0    1
[18,]    1    1    0
[19,]    1    0    0
[20,]    0    0    0
[21,]    0    0    0
[22,]    0    0    0
[23,]    0    0    0
[24,]    0    0    0
[25,]    0    1    0
[26,]    0    0    0
[27,]    1    0    1
[28,]    0    0    1
[29,]    1    0    1
[30,]    1    1    0
[31,]    0    0    0
[32,]    1    0    1
[33,]    0    1    0
[34,]    0    1    0
[35,]    0    0    0
[36,]    1    1    1
```

```
> getRects(mut1[,1:3],Grid)
[[1]]
ID        X       Y
15  6 4600404 2758495
23 12 4600404 2758405
27 14 4600044 2758315
31 18 4600404 2758315
34 19 4599954 2758225
44 27 4600134 2758135
46 29 4600314 2758135
47 30 4600404 2758135
51 32 4600044 2758045
55 36 4600404 2758045

[[2]]
ID        X       Y
15  6 4600404 2758495
23 12 4600404 2758405
27 14 4600044 2758315
28 15 4600134 2758315
31 18 4600404 2758315
42 25 4599954 2758135
47 30 4600404 2758135
52 33 4600134 2758045
53 34 4600224 2758045
55 36 4600404 2758045

[[3]]
ID        X       Y
11  2 4600044 2758495
13  4 4600224 2758495
15  6 4600404 2758495
23 12 4600404 2758405
30 17 4600314 2758315
44 27 4600134 2758135
45 28 4600224 2758135
46 29 4600314 2758135
51 32 4600044 2758045
55 36 4600404 2758045
```

R Output 9: Left: First 3 exemplary individuals after trimton-function - Right: Output of getRects-function for the exemplary 3 individuals

The columns of the matrix on the left represent the various individuals or wind farms of the population. The rows represent the discrete genes or grid cells of an individual. In this case, 36 grid cells are available whose centroids are each a potential location for a turbine. Grid cells with the value 1 are extracted for each individual and according to the row numbers; the stored values from the indexed data frame from chapter 3.2.1 are assigned to a new list, resulting in the output on the right side above. The colors signal the correspondence of individual genes with the value 1 to the resulting grid ID's and coordinates.
This function marks the last step of the program cycle and is activated starting with the second iteration. The algorithm then continues with evaluating the fitness of the new generation.

3.2.9. Termination Criteria

The algorithm stops when the amount of generations exceeds the input variable *iteration* or when a wind farm with an efficiency of 100% is found.

4. Results

To verify that the algorithm works, it is tested with a small rectangular area, where the optimal solutions are easy to find. The different methods of the algorithm are tested on this area to identify their pros and cons and to find the best configuration scheme.
With this optimal input configuration scheme, the algorithm is applied to a hypothetical case area that was widely used in the literature (Case 2: Reference Shape). (Mosetti, et al., 1994) (Grady & Hussaini, 2005) (Ituarte-Villarreal & Espiritu, 2011) (Ituarte-Villarreal & Espiritu, 2011) (Shakoor, et al., 2015)

Next, the terrain effects on an existing wind farm in Austria, an area in Styria (Case 3: Wind Farm "Tauern"), are tested at a considerable altitude of approximately 1800m.

The results of each case and the behavior of the algorithm are illustrated and discussed in each corresponding subchapter. The output illustrations of all cases consist of 8 similar plots (e.g. for Case 4.1.1 consider Figure 21 and Figure 22):

- Figure a) displays the best-found solution of a test case. The given area is colored in blue and the corresponding grid is overlaid in black. The points inside this grid represent the turbine locations of the best-found solution. The colors and the values below these points indicate their total wake effect in a traffic light scheme. Lower wake effects will result in green points and contrary higher wake effects will be colored in red. Below the best-found solution the minimum and average distance between all turbines of this solution are noted.
- Figure b) shows a heat map of the entire grid and gives an indication of how often a particular grid cell was selected during the entire optimization process. The values are calculated with an Inverse Distance Weighting method and plotted in a traffic light scheme in a similar manner as in Figure a). The more often a grid cell has been selected, the greener the cell becomes and the more rarely it has been chosen, the redder it is shown.
- Figure c) indicates the development of the efficiencies of all generations. The maximum efficiency rates are plotted in green, average rates in blue and minimum rates in red. With the exception of the case that terrain

effects are neglected, the efficiency rates are always directly proportional to the energy outputs and therefore to the fitness values of the individuals.

- Figure d) gives an overview of the population sizes during all generations. The amounts of individuals are counted after the fitness, selection and crossover function, although they are identical for the fitness and crossover function. Black points indicate the number of individuals after the fitness function, red points the number after the selection function, and green points the number after the crossover function. Thus, 3 points per generation are visible.
- Figure e) shows for all generations the selection percentages in the upper half and the number of used crossover parts in the lower part.
- Figure f) is a combination of Figure b) and the upper half of Figure e). The efficiencies for all generations are displayed and green horizontal lines are drawn for the generations where the selection percentage was greater than 75%.
- Figure g) is a combination of Figure b) and the lower half of Figure e). The efficiencies for all generations are displayed again and red horizontal lines are drawn for the generations where the number of crossover parts was greater than 2.
- Figure h) uses again the efficiency values of Figure b) and adds black horizontal lines for the generations in which a variable mutation rate was applied, instead of the fixed mutation rate.

All tests were conducted on a PC with the following specifications:

- Intel Core(TM) i7-2600 CPU @ 3.40 GHz
- 8GB RAM

The following R code shows an example of calling the *genAlgo* function, which starts an optimizing process with the given input values.

```
Result <- genAlgo(Polygon1=Polygon, n=15, SurfaceRoughness=0.3,
                  Rotor=30,fcrR=15, RotorHeight=60, referenceHeight=60,
                  iteration=100, Proportionality=0.2, mutr=0.08,
                  vdirspe = data.in, topograp="TRUE", elitism="TRUE", nelit=7,
                  selstate="FIX", crossPart1 = "EQU", trimForce="TRUE")
```

R Output 10: Exemplary call to *genAlgo*

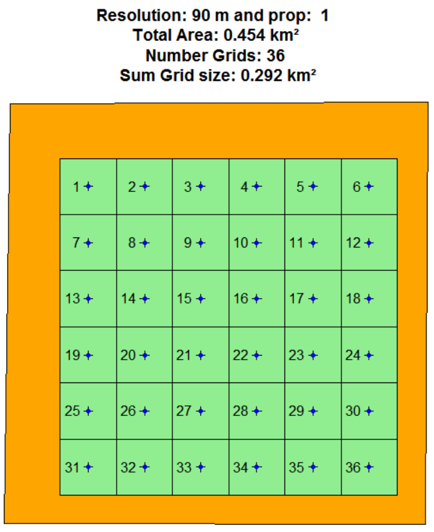
4.1. Case 1: Test Shape

4.1.1. Fixed Selection vs. Variable Selection

The following input values are used for all tests of Case 1. Exceptions are mentioned at the beginning of each subchapter.

Input Variable	Value
n	12
SurfaceRoughness	0.14
Rotor	30
fcrR	3
RotorHeight	60
referenceHeight	60
iteration	100
Proportionality	1
mutr	0.008
vdirspe	data.in
topograp	"FALSE"
elitism	"TRUE"
nelit	6
selstate	"FIX"
crossPart1	"EQU"
trimForce	"FALSE"

Output of *GridFilter*



The data frame (*data.in*) containing the wind speed information has the following inputs and the resulting wind rose illustrated below on the right.

Input Wind data frame

```
> data.in
  ws  wd  probab
  12  0     25
  12  0     25
  12  0     25
  12  0     25
```

Windrose of *data.in*

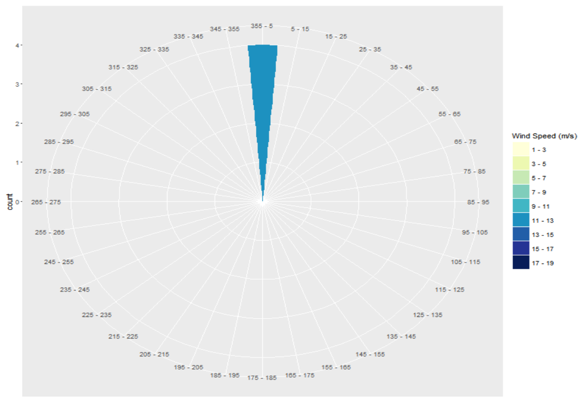
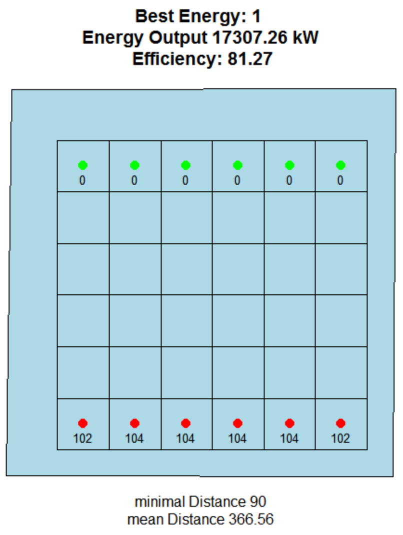


Figure 20: Input values for Case 1 - Test Shape

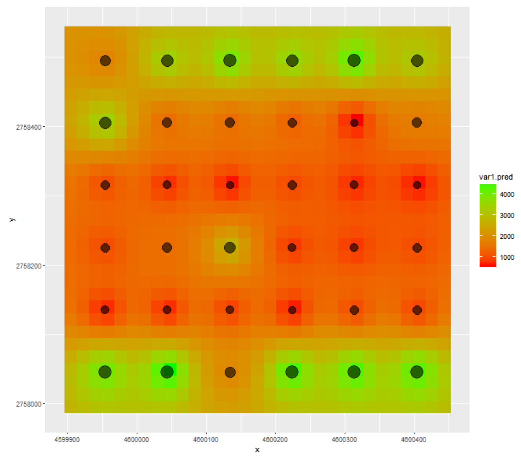
Fixed Selection

All input variables are as described on the previous page. This will mark the default of the input configurations for Case 1.

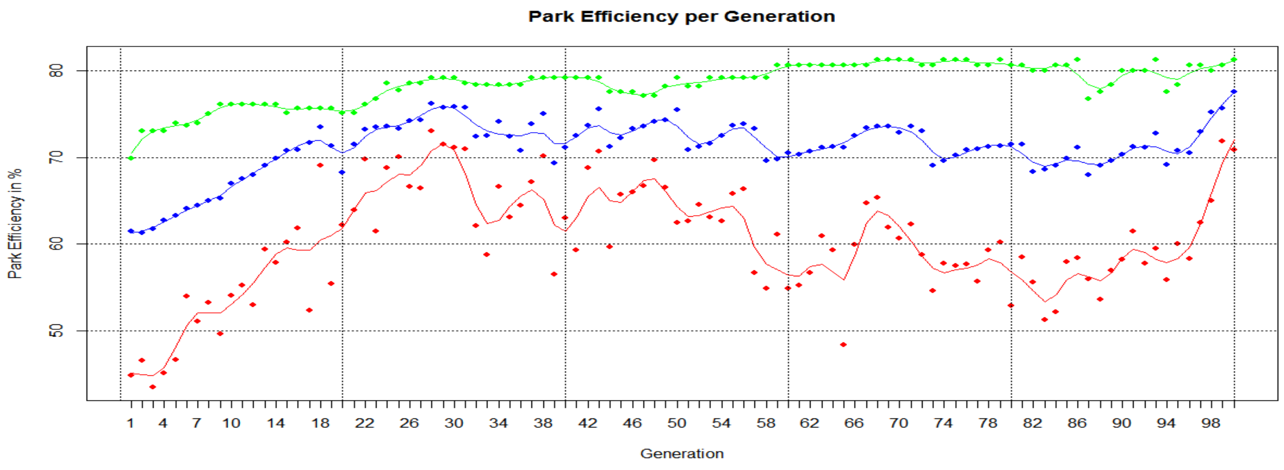
a



b



c



d

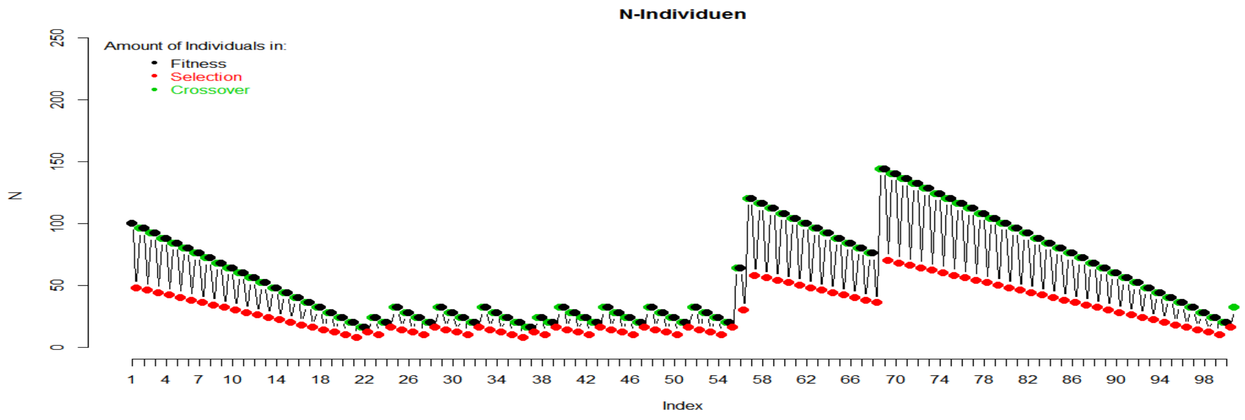
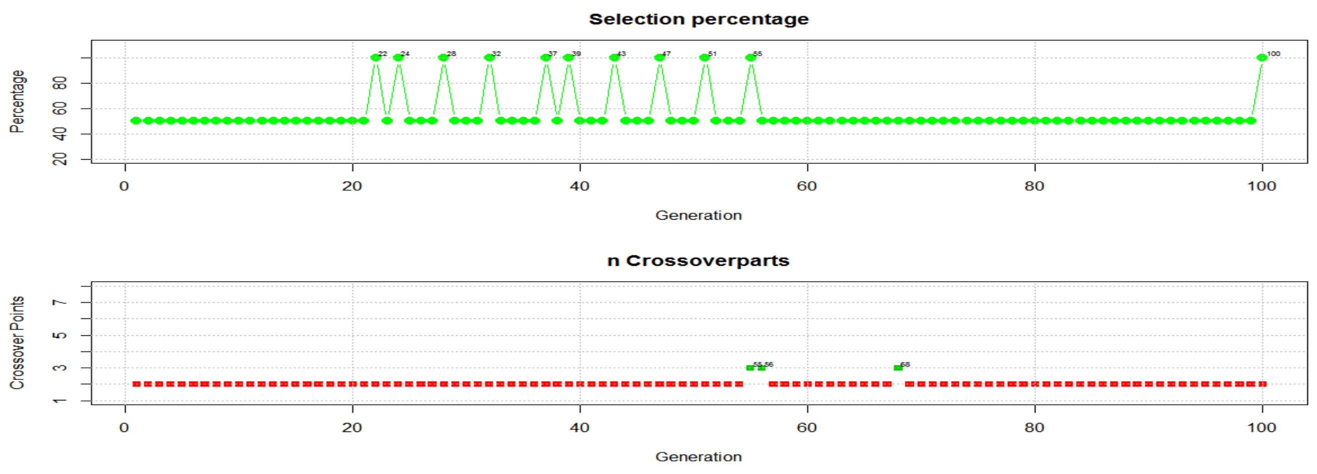
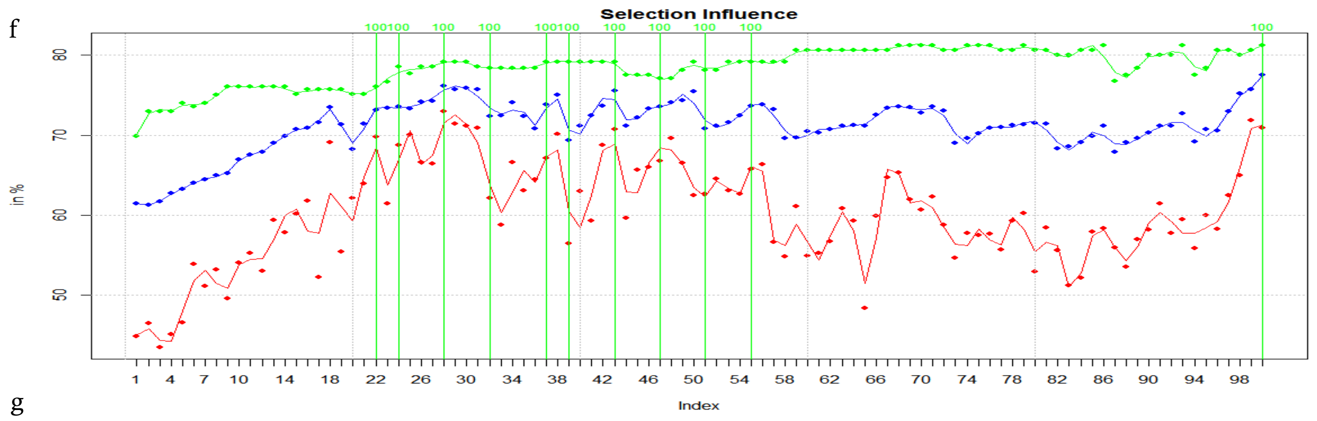


Figure 21: Results of Default Input: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

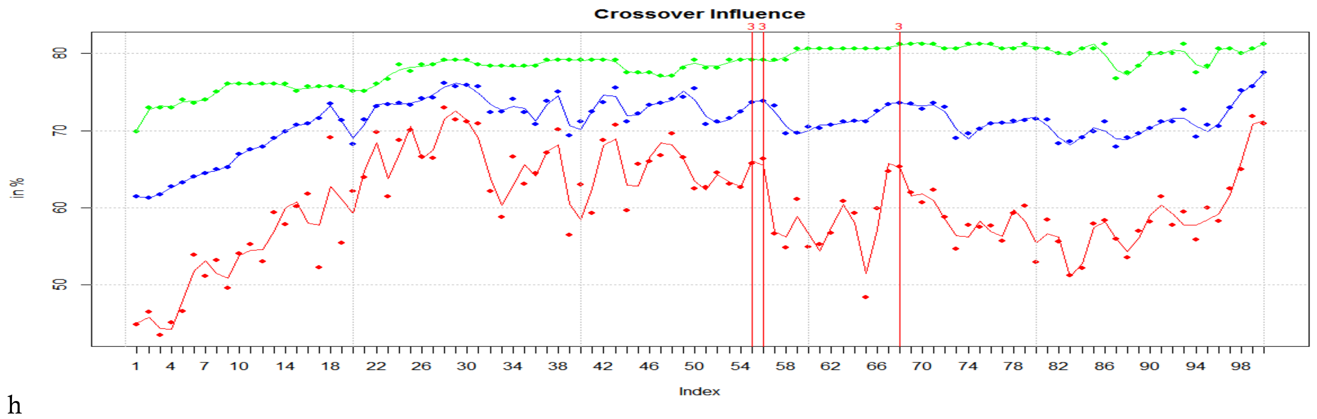
e



f



g



h

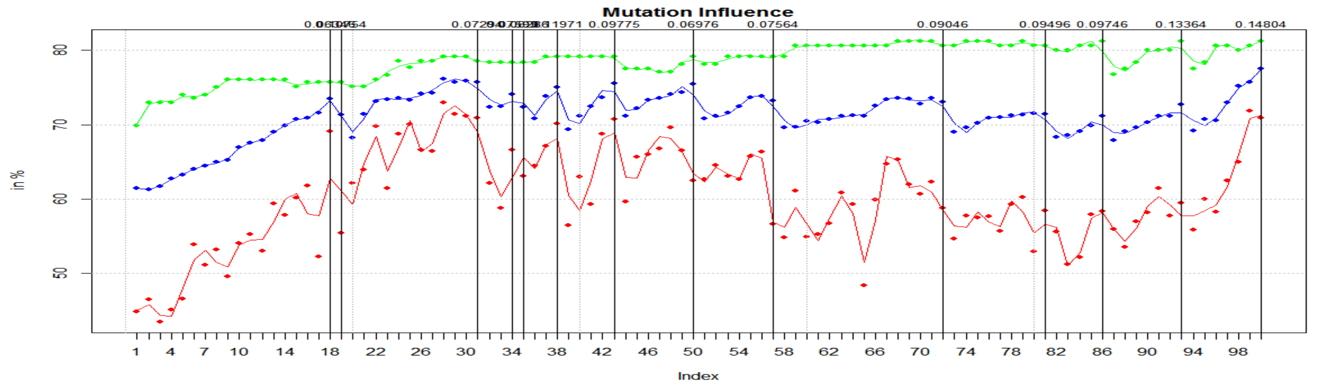


Figure 22: Results of Default Input: e) The selection percentage during the evolution f) High selection influence on evolution g) Crossover influence of 3 crossover parts h) Variable mutation rate influence

The result found in figure a) represents the best possible solution and therefore the global optimum of the genetic algorithm. The point color and the number below a turbine location indicate the total wake effect of this turbine. Green points indicate low wake effects, while red points indicate high wake effects.

The heat map in figure b) demonstrates how often a grid cell was selected during the entire optimization process and is calculated with an inverse distance weighting. The more often a grid cell is selected, the more the cell is colored green and the more rarely it is chosen, the more the color becomes red.

In figure c) wind farm efficiencies are plotted for all iterations. The maximum efficiencies are plotted in green, average rates in blue and minimum rates in red. In this test, the algorithm seems to converge at the initial stage until iteration t=30, and although the maximum remains nearly constant during this period, the average efficiencies increase almost constantly, similar to the minimum rates, although with higher fluctuations. It is also apparent that the algorithm has found the best possible solution in the 73 generation.

In figure d), the set of individuals in the populations is displayed. The amounts are counted after the fitness, selection and crossover function, although they are identical for the fitness and crossover function. A fixed selection rate of 50% and 2 crossover parts in the early states constantly decrease the populations by the 4 deleted individuals in the selection function. Once the amount of individuals is less than 20, all individuals are selected with a selection percentage of 100% and the crossover point rate is increased. As soon as the crossover point rate exceeds a value of 3, 3 crossover parts are used, leading to increased permutations and population sizes. As a result, the variance of the individual fitness values will also increase, which can lead to better solutions, since the algorithm would have more building blocks to construct a wind farm layout and would therefore have better exploration properties. This can be observed at the iteration t=59 at which a better solution was found shortly after the two-time use of 3 crossover parts.

The selection percentage and the amount of crossover parts for all iterations are presented in figure e). Figure f) shows the development of efficiencies and marks generations with green lines, in which the selection percentage is over 75%. Figure g) identifies iterations with more than 2 crossover parts with red lines and figure h) indicates where the variable mutation rate was used with black lines.

As noted above, the fitness of the populations or their efficacy increased while the number of individuals decreased consistently up to the ~20 generation. Then, as more than 3 duplicated individuals represented the actual best solution, the mutation rate was increased for the first time at the iteration t=19.

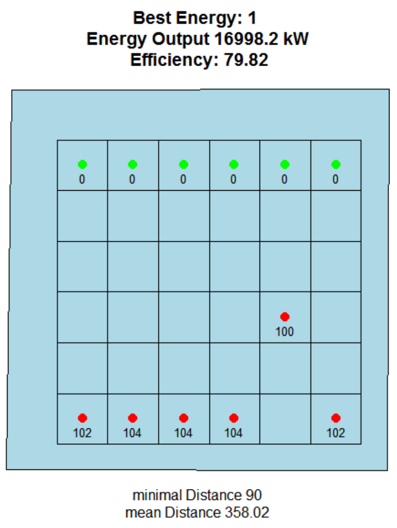
The amount of individuals was quite low during t=20 and t=60. As less than 20 individuals existed, the selection percentage was set to 100% and the crossover point rate was increased. This did occur 11 times, as can be seen in figure e) & f) and the crossover rate increased therefore in total by 0.33 (11*0.03). This was sufficient, that the crossover point exceeded 3 in t=55 and the population size increased rapidly. The algorithm used 3 crossover parts only 3 times in total, but the influence on the variance of the individuals was intense. Although the algorithm jumped back to 2 crossover parts at t=57 to t=67 and at t=69, the variance stayed considerable almost until the end.

As the populations in this method were quite small, the computational time needed was comparably low and took in total about half an hour.

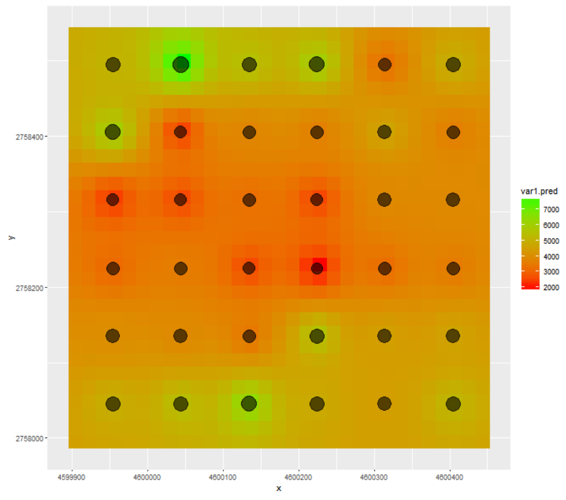
Variable Selection

Exception: The variable *selstate* is set to "VAR".

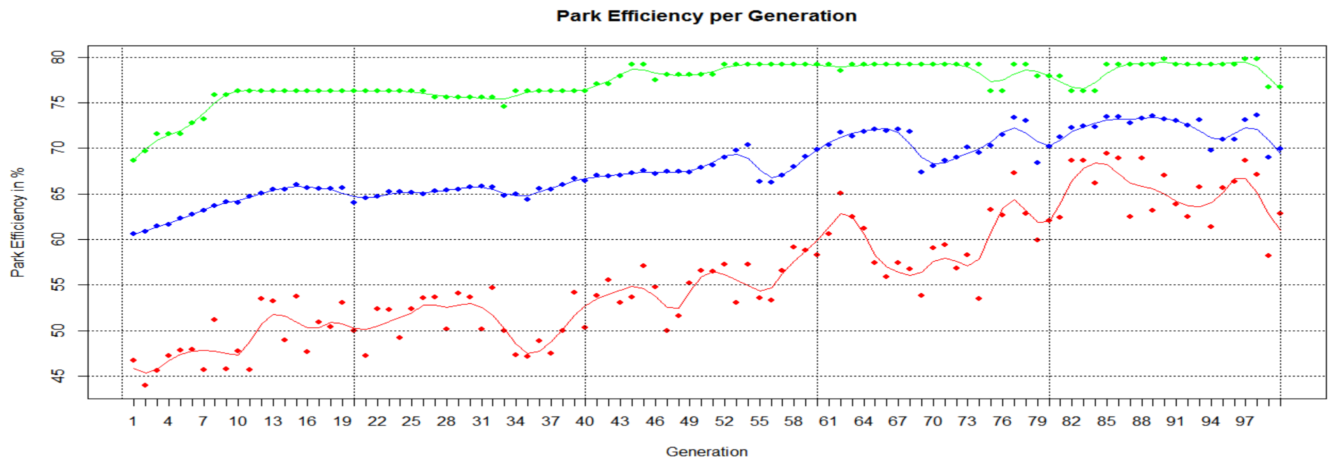
a



b



c



d

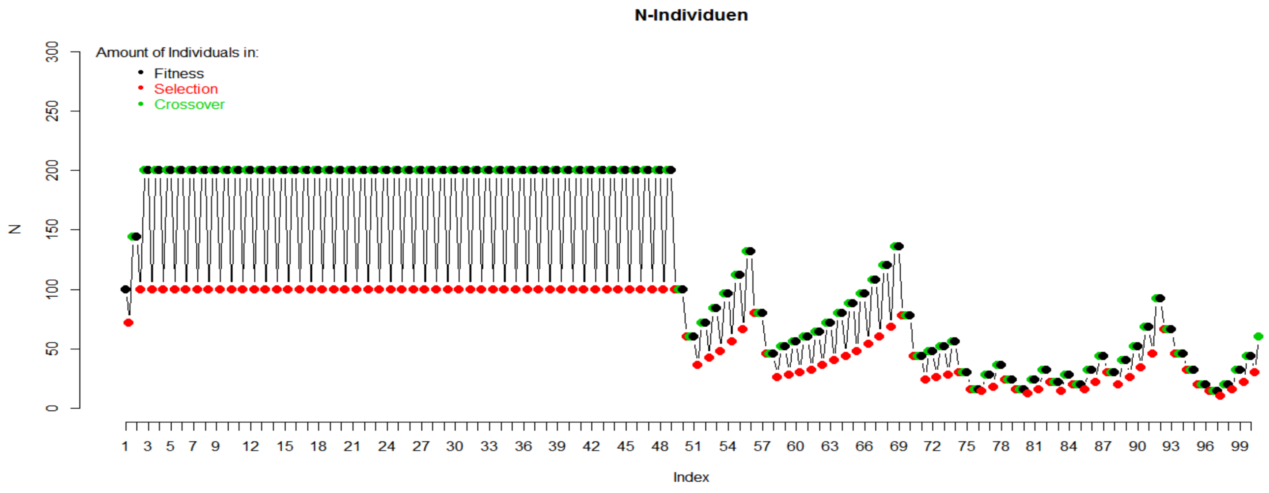
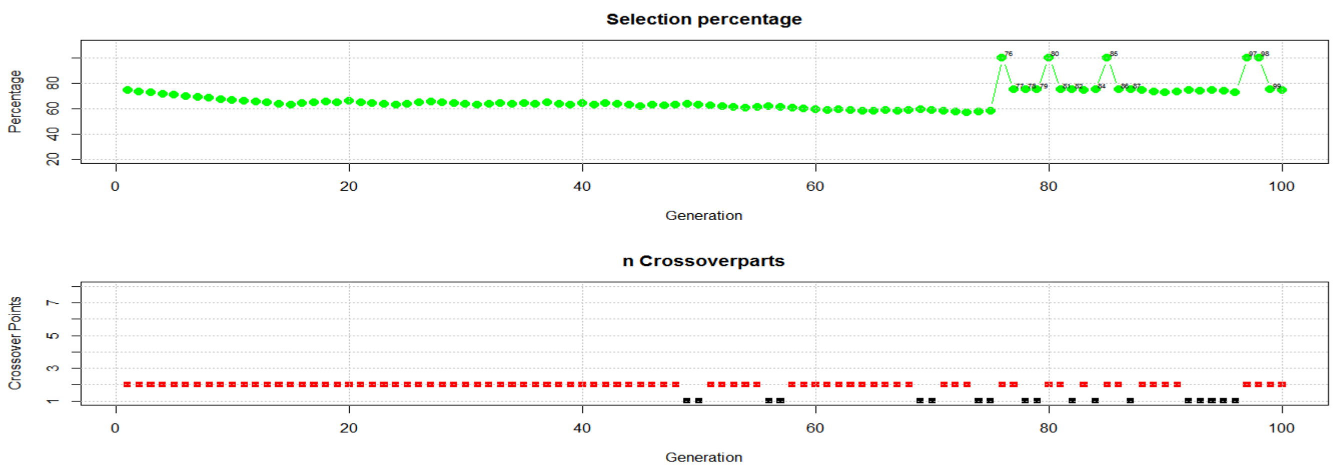
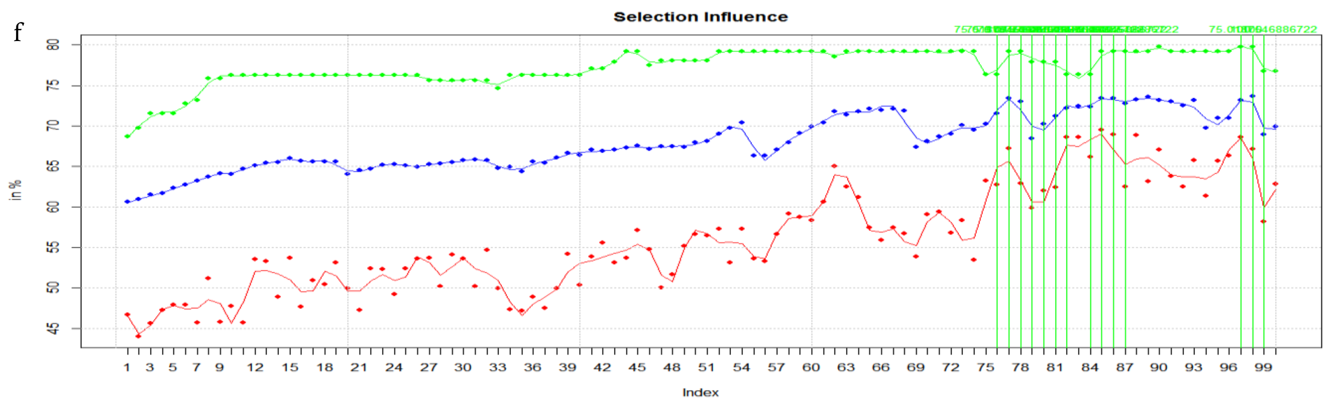


Figure 23: Results of Variable Selection: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

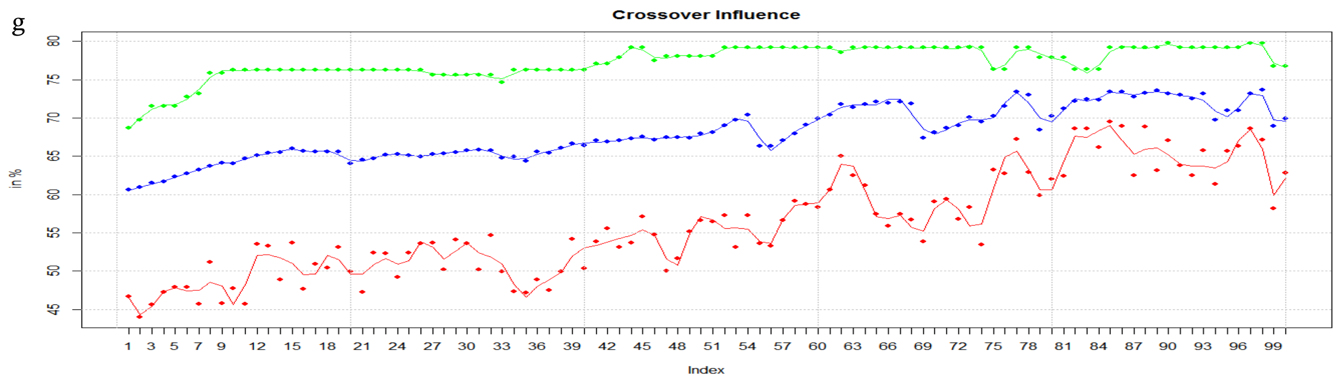
e



f



g



h

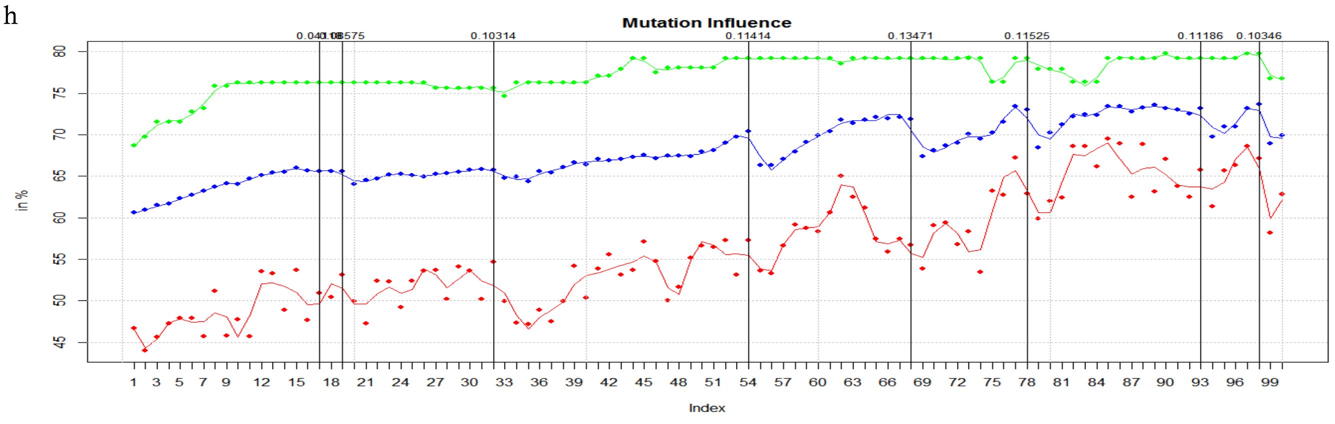


Figure 24: Results of Variable Selection: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The variable selection method did not lead to the best possible solution even if it came quite close and only 1 turbine would have to be moved 2 grid cells to the south. The heat map indicates that with this method the preference for good grid cells is not as strong as with the fixed selection. This can be attributed to higher numbers of individuals in the initial populations, i.e. to more slowly converging properties, and hence to more diverse and inferior possibilities that had to be evaluated.

The converging abilities of this method appear to be weaker than in the fixed selection method. On the one hand, the variance of the fitness values remained high up to the ~80th generation, and on the other hand, the algorithm even fell back several times on 1 crossover part. This has reduced the number of individuals and thus the variance in the populations so that the algorithm could only focus on a smaller number of currently best individuals.

The state with 1 crossover part leads to reduced population sizes, which are evident especially during the iterations t=70 to t=87, whereby the selection percentage had to be increased to 100% for 3 times as a result of too small population sizes.

Although the found solution is near the optimum, it took about twice as much computing time as the fixed selection method due to higher population sizes, between t=1 and t =50. During these iterations, 200 individuals were evaluated in the fitness function, which is by far the most time-consuming function of this algorithm. For these reasons given, the variable selection method is classified as weaker than the fixed selection with current settings. With more intelligent selection parameters which, among other things, increase or decrease the variable selection pressure relative to the fitness development or adapt the initial selection rate relative to the problem size, this method could probably be adapted reasonably.

4.1.2. Equal Crossover vs. Random Crossover

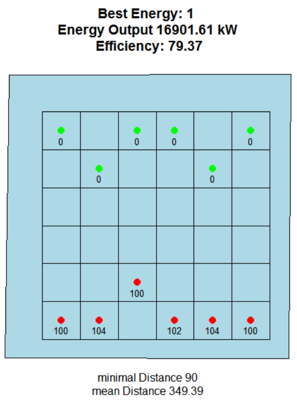
<u>Equal Crossover</u>

Equal Crossover (*crossPart1* = "EQU") is the default of the input configuration scheme. The outputs of the fixed selection method are used as reference.

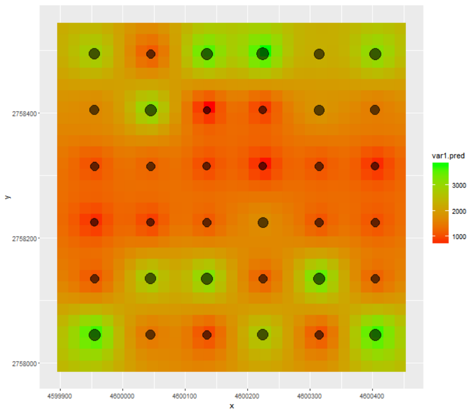
<u>Crossover Random</u>

Exception: The variable *crossPart1* is set to "RAN".

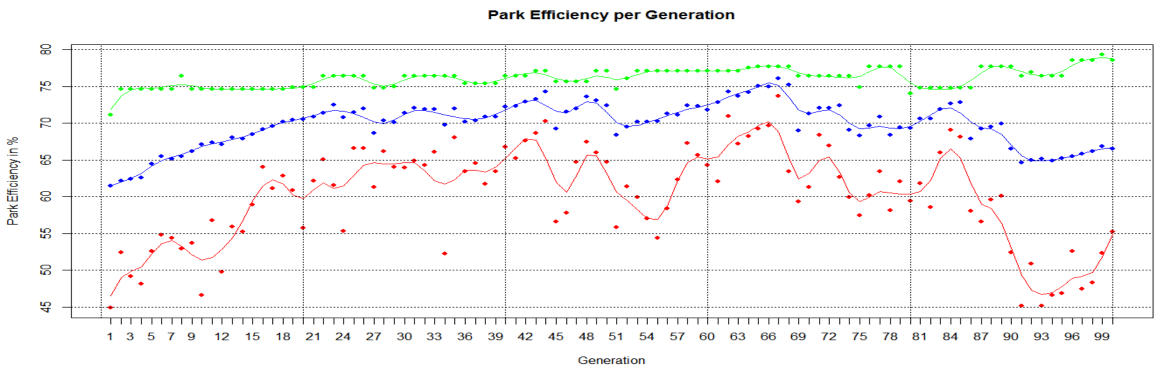
a



b



c



d

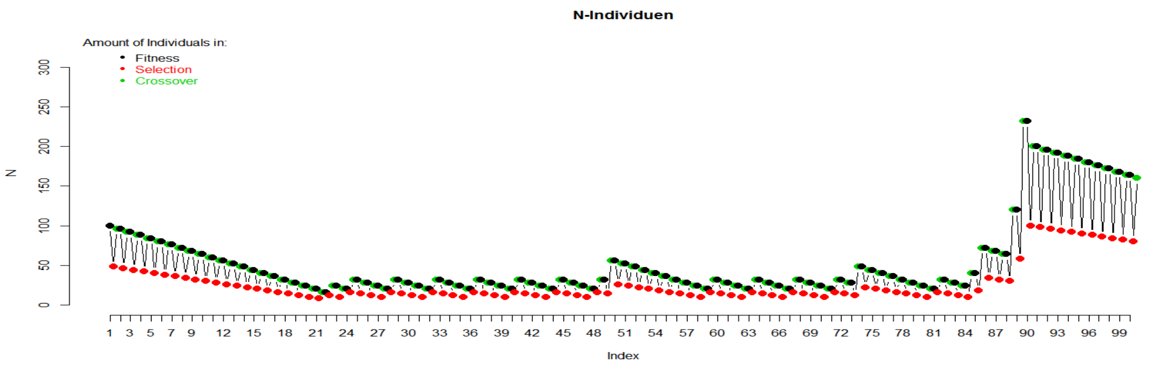
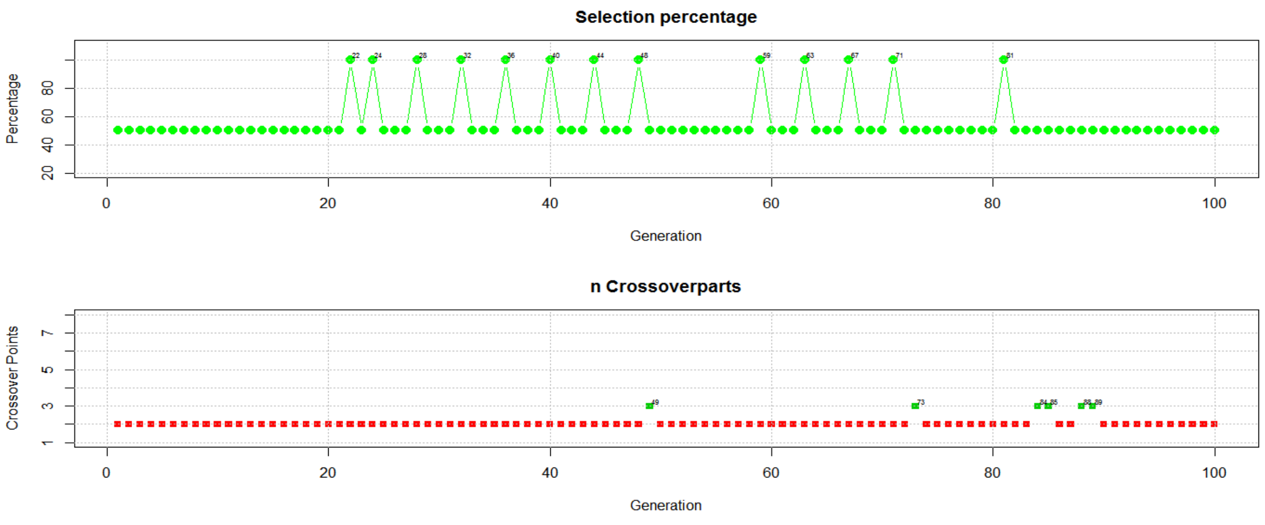
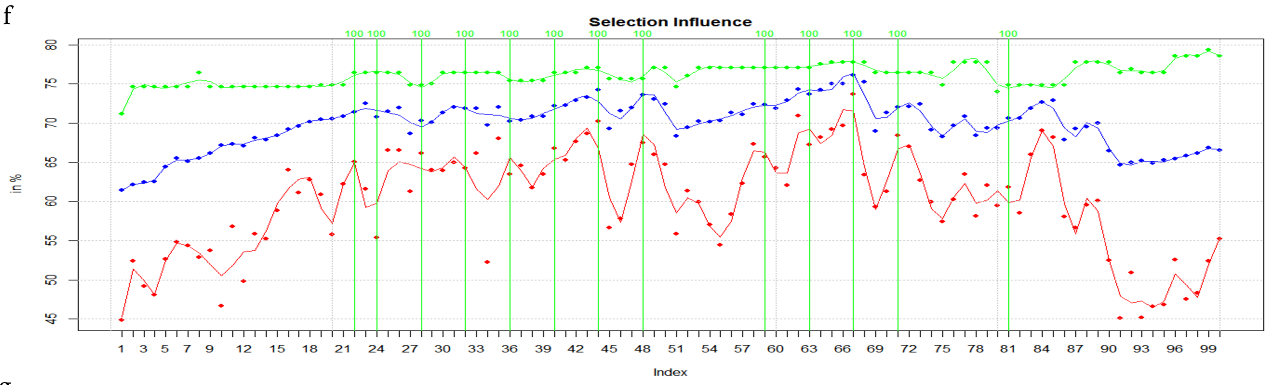


Figure 25: Results of Random Crossover: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) d) *d)* Amount of individuals after every evolutionary method

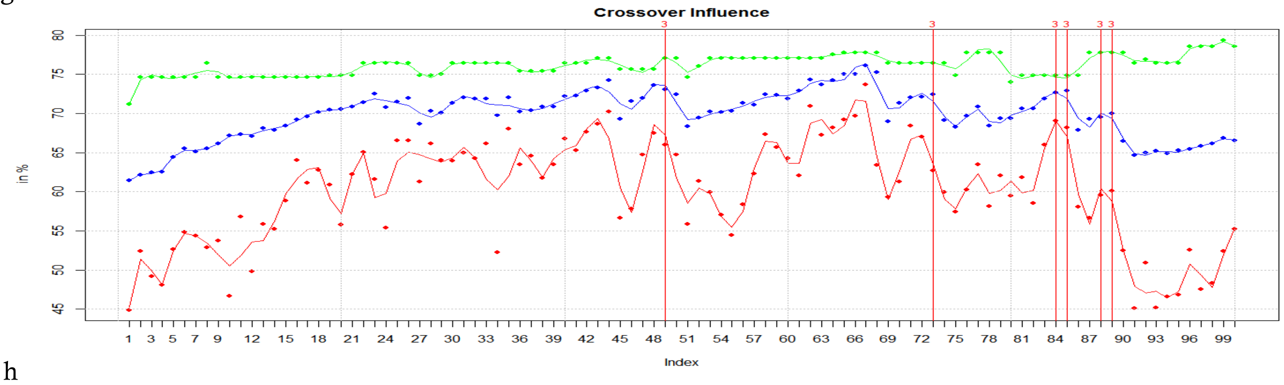
e



f



g



h

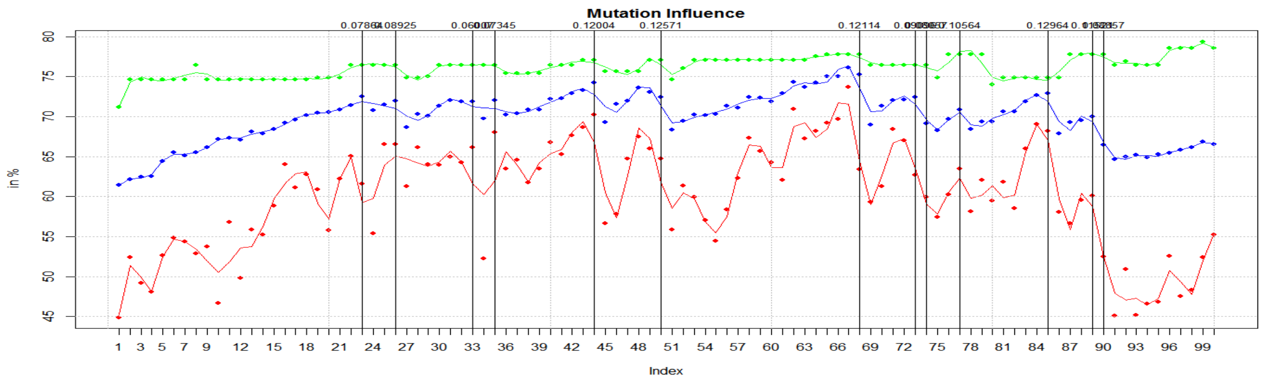


Figure 26: Results of Random Crossover: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The solution found with the random crossover method is worse than both previous methods and it was found close to the end of the optimization run, in the 99th generation. Although the heat map indicates strong preferences for certain grid cells, some of these often selected grid cells do not belong to the optimal wind farm layout which can be seen in Figure 21 *a*).

The evolution of the maximum fitness values is not as constantly increasing as the previous methods and remained more static, especially in the first 20th generations. The mean and minimum values also seem to increase slower than the equal crossover method.

The population sizes were overall quite small and the algorithm had to increase the selection percentage 13 times to 100%. This lead to 3 crossover parts at t=49, t=73 and 4 times during t=84 to t=89. The variance between the best and worst individuals increased then strongly with minimum fitness values, that are similar to the worst individual in the first generation and although the mean fitness values dropped also for about 5%, the algorithm was able to find better solutions after t=96.

Although this method was comparably fast and took about 20 minutes, it is considered weaker than the previous methods and harder to interpret, as no information of the length of all randomly divided crossover parts is available. Similar to the previous method, the random crossover technique could be implemented and used in a smart way. The algorithm could use random crossover, when the populations seem to be stuck in local optima, where more variance is required to explore the search space. With equal and constant crossover parts, the method is more predictable, as it cuts the genetic code always at the same locations and I assume that the converging abilities are therefore stronger. If this behavior does not lead to any further improvement, the crossover point locations could be randomly created or increased.

4.1.3. Elitism Inclusion vs. Elitism Exclusion

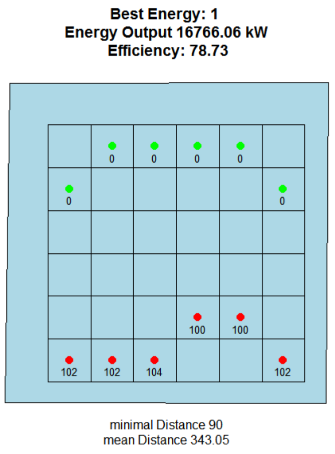
Elitism Included

Included elitism (*elitism* = "TRUE") is the default of the input configuration scheme. The outputs of the fixed selection method are used as reference.

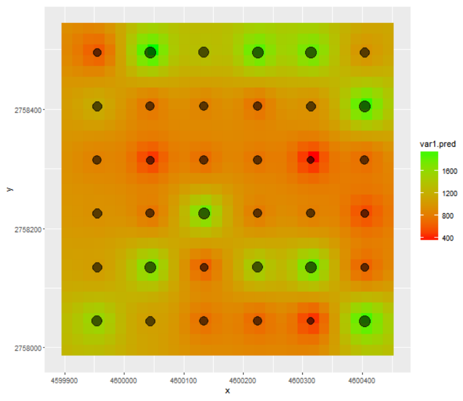
Elitism Excluded

Exception: The variable *elitism* is set to "FALSE"

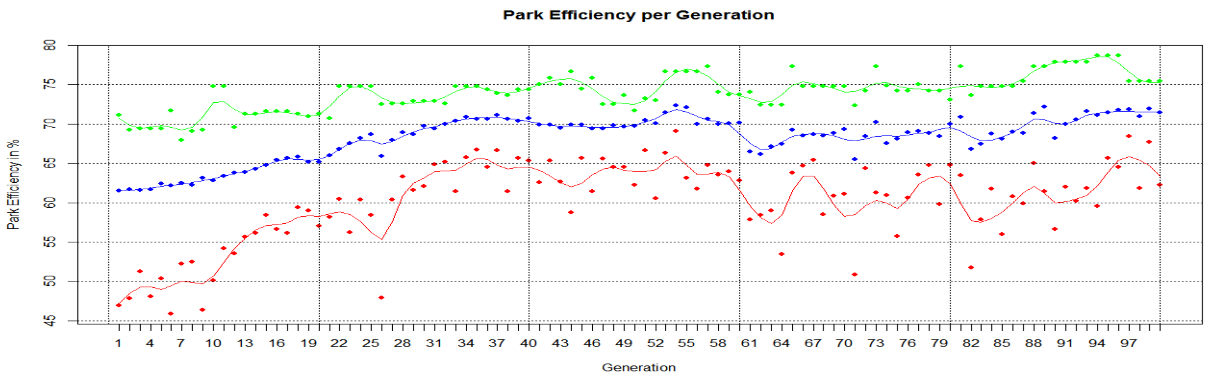
a



b



c



d

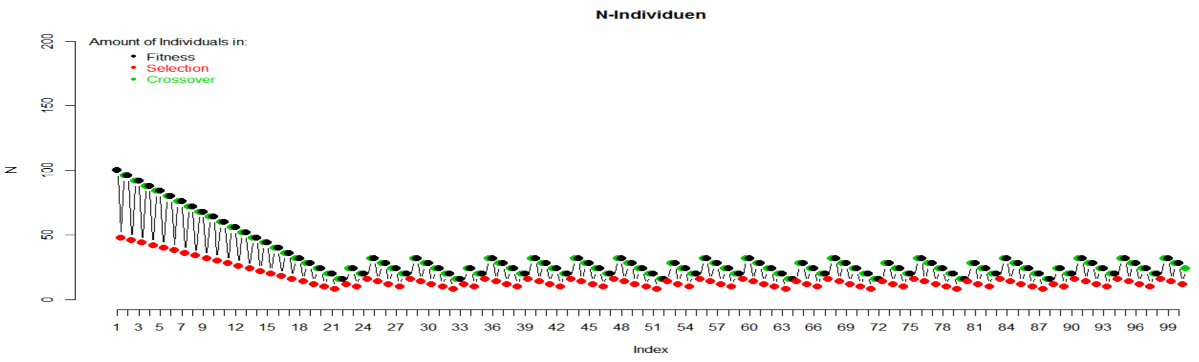
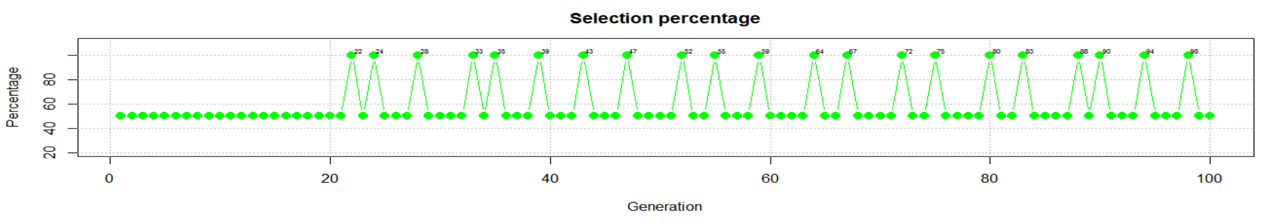
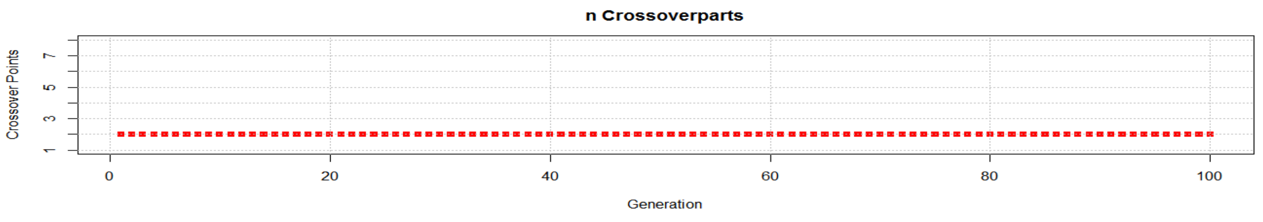


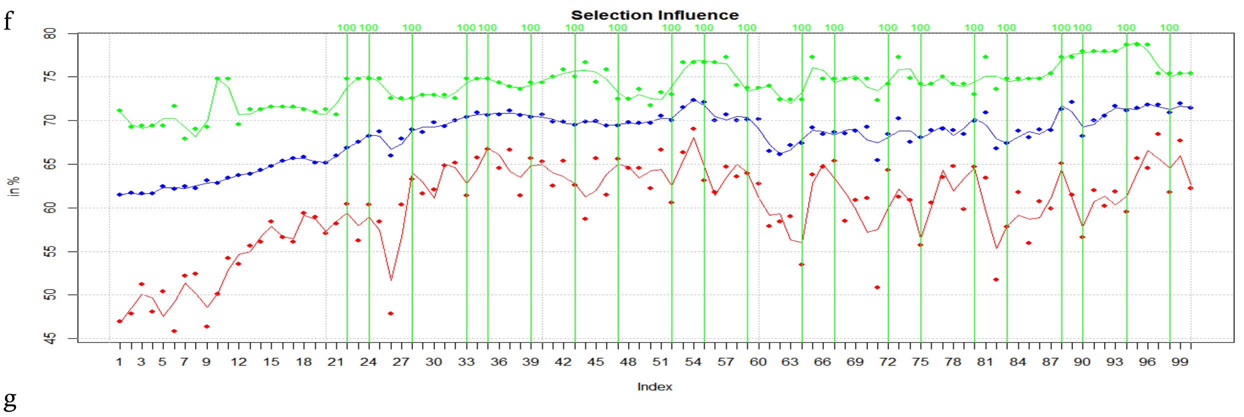
Figure 27: Results of Excluded Elitism: *a)* The best found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

e

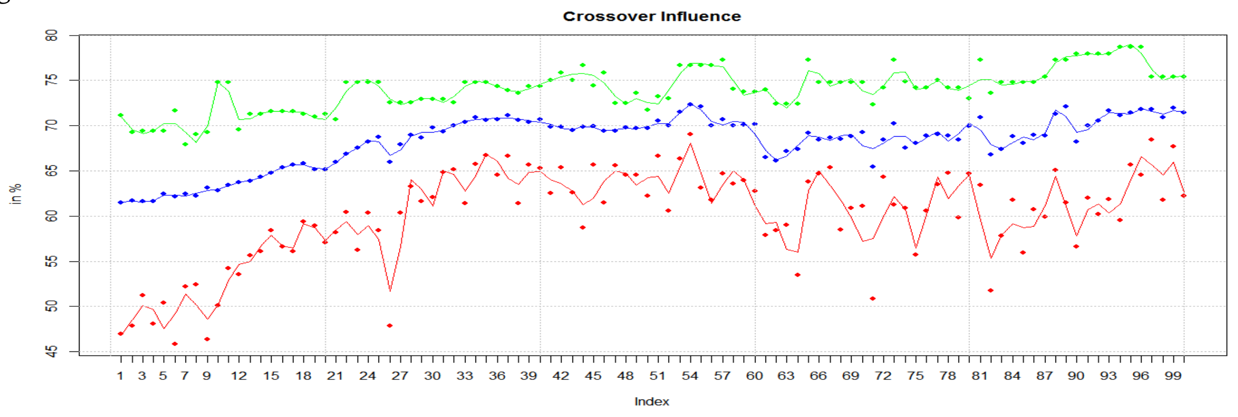




f



g



h

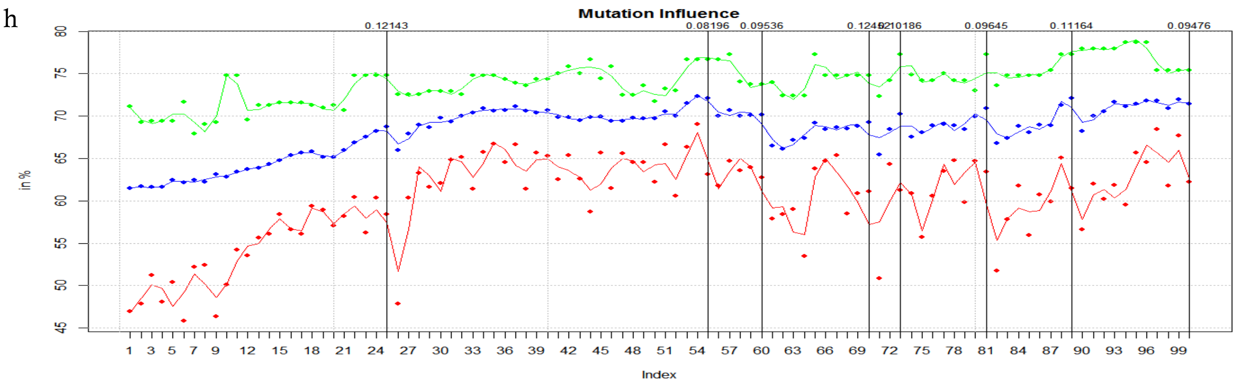


Figure 28: Results of Excluded Elitism: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The solution found without elitism is the worst solution of all test cases. 4 turbines would have to be displaced to achieve the best possible solution. The heat map shows strong preference for certain grid cells, where half of these cells do not belong to the optimal solution.

The maximum fitness values vary much more than with previous methods and even after the first generation, the maximum value dropped for about 3%. In the early stages, all other methods show an increase in maximum fitness values. Although the selection percentage had to be increased 21 times, which additionally increased the crossover point rate, it did not lead to an increase in the crossover parts. The random crossover results in comparison had a selection rate of 100% for 13 iterations only, and 3 crossover parts for 6 iterations. The fixed selection method required only 11 selection percentages of 100 and had 3 crossover parts for 3 generations. The higher fluctuations in the maximum fitness values of this method decreased the crossover point rate more often, while low population sizes below 20 individuals increased the crossover point rate. Because of these counterbalancing effects, the crossover parts were not able to change and remained constant at 2.

The variable mutation method was used 8 times, which is similar to the variable selection method. All other methods used the variable mutation method more often, which can be interpreted as superior convergence ability.

The population sizes in this method were very small and the optimization process took only 15 minutes. Although this method was the fastest, it was also the weakest.

4.1.4. Random Adjustment vs. Probabilistic Adjustment

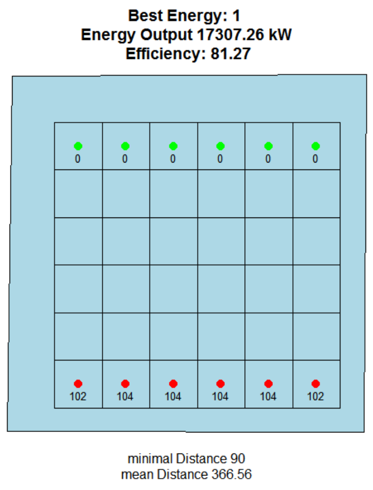
Random Adjustment

Random Adjustment (*trimForce* = "FALSE") is the default of the input configuration scheme and therefore identical to the outputs of the fixed selection method.

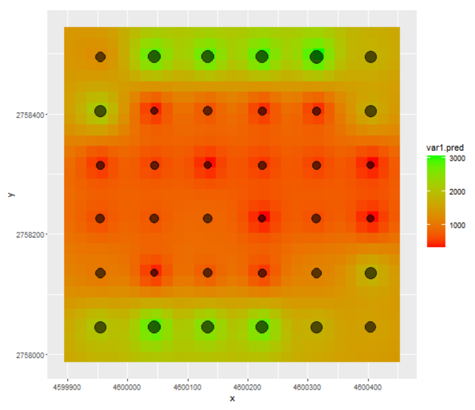
Probabilistic Adjustment

The variable *trimForce* is set to "TRUE".

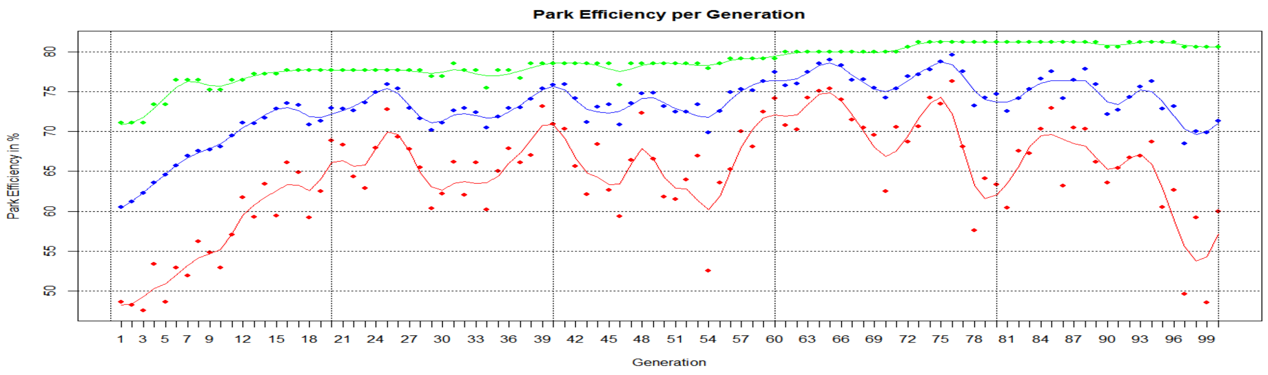
a



b



c



d

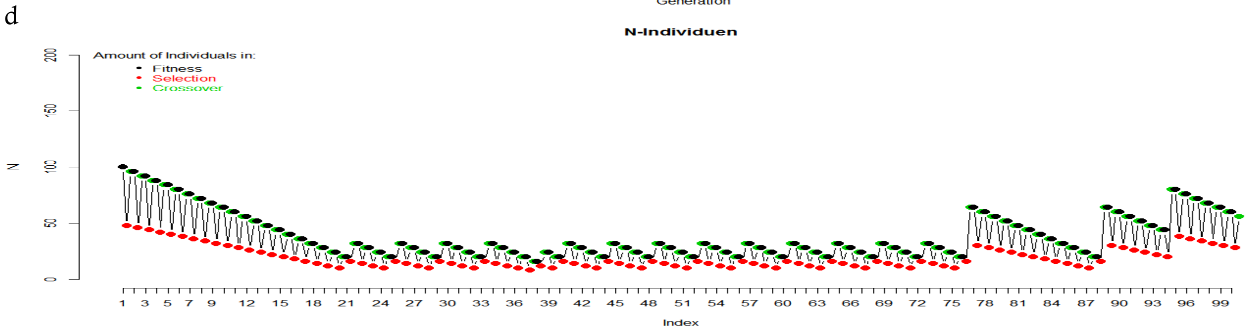
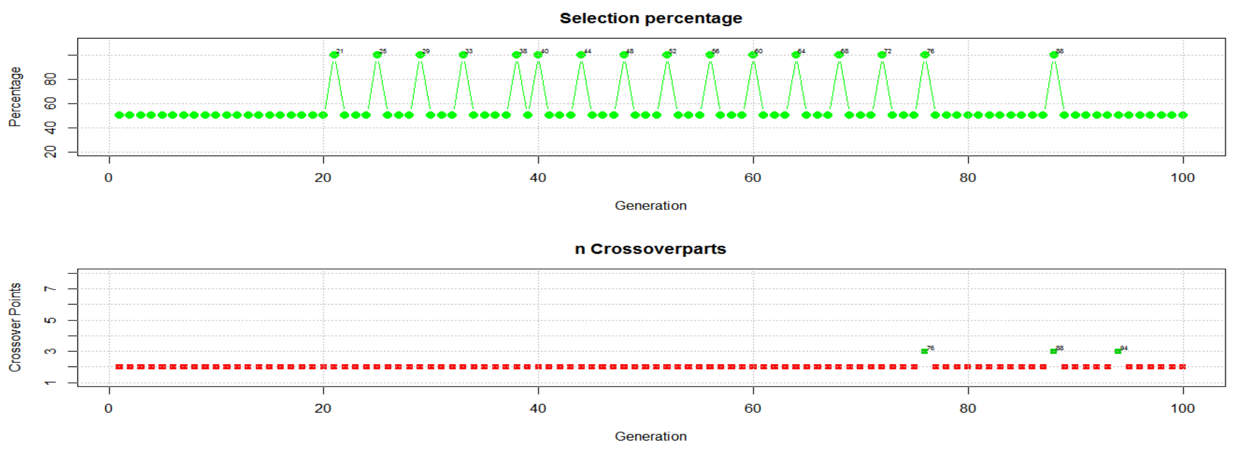
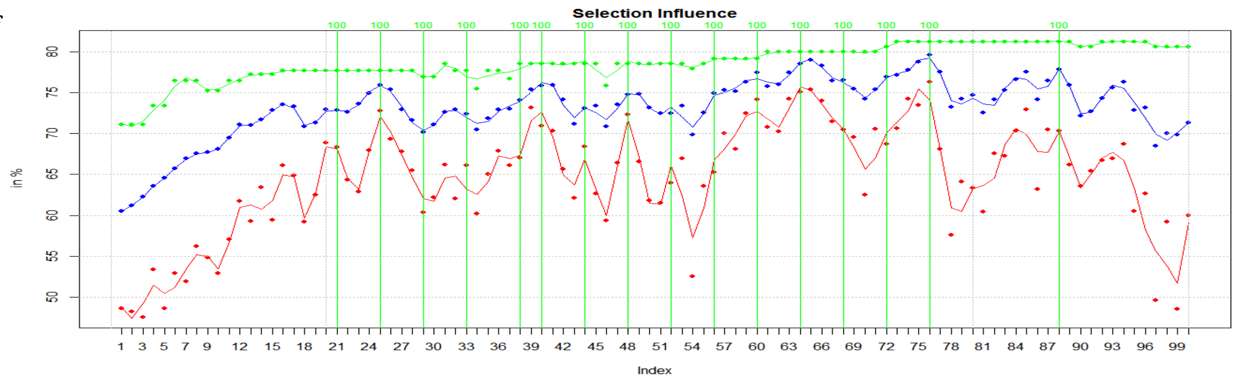


Figure 29: Results of Probabilistic Adjustment: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

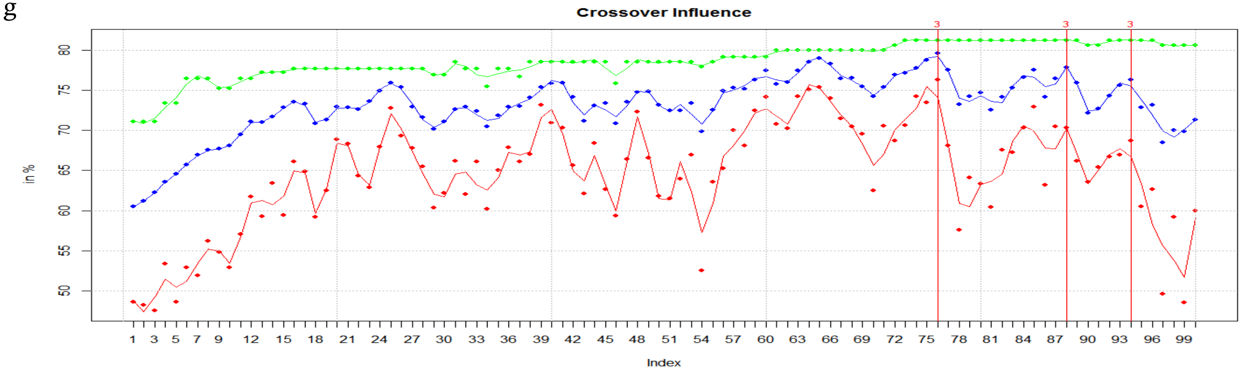
e



f



g



h

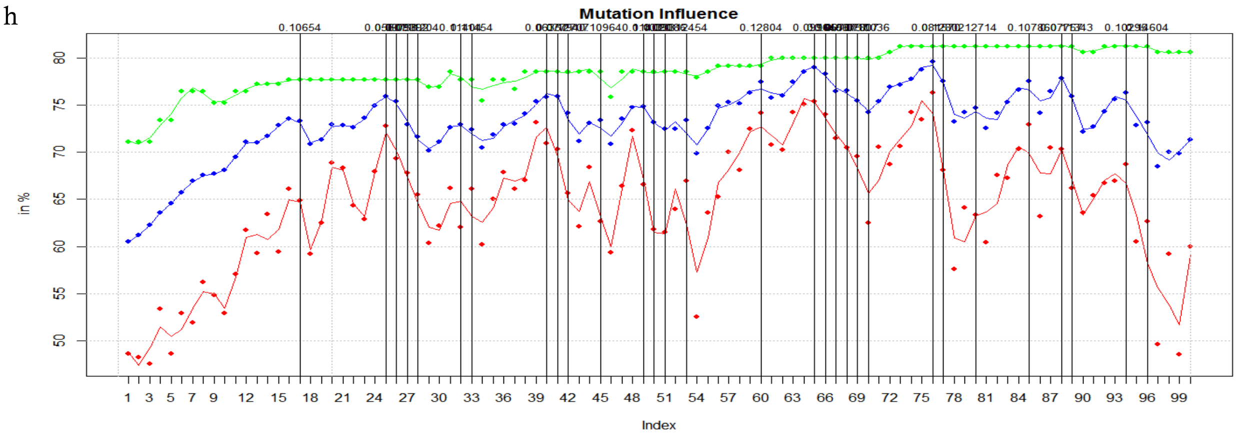
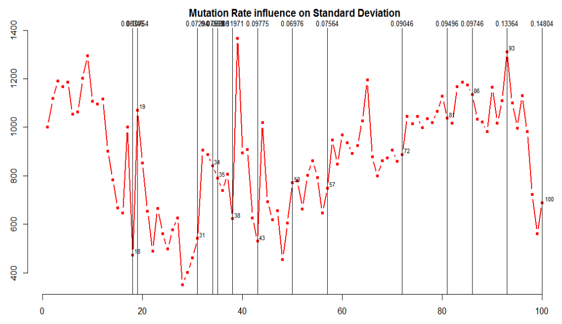


Figure 30: Results of Probabilistic Adjustment: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The result found by this method is the best possible solution and identical to the result from the fixed selection method. The heat map shows that 7 of the strongly preferred grid cells correspond to the best possible solution.

The maximum fitness values increased almost constantly and reached the global optimum in the 73 generation, similar to the results with default inputs. The average efficiency of this method, which is the only to reach almost 80% at t=76, is thus significantly higher than in the other methods. The exploitation qualities seem to be stronger with a probabilistic adjustment method compared to the random one, especially after the first ~20 generations. This can be seen in the standard deviations of the individual fitness values during the whole evolution process.

Random Adjustment (4.1.1)



Probabilistic Adjustment (4.1.4)

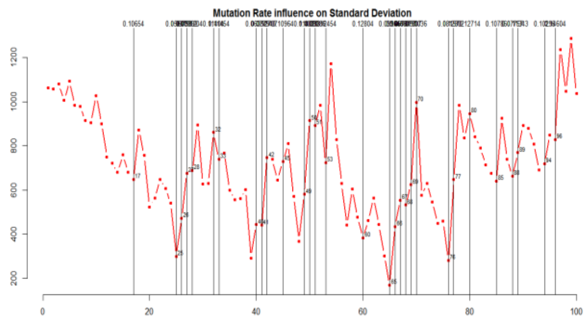


Figure 31: Standard deviations of populations with random adjustment on the left and probabilistic adjustment method on the right. Black lines indicate a variable mutation rate

The values with random adjustment range between 400 and 1400 while the probabilistic adjustment values range between 200 and 1200 and are considerably lower.

The population sizes were quite small and the algorithm had to use a selection percentage of 100% for 16 times in total. This resulted in 3 crossover parts for 3 times during the evolution at t=76, t=88 and t=94, which then lead to an increase in the variance of the individuals and population sizes.

The strong exploitation properties are also evident by the frequently used variable mutation rates. Since the algorithm only switches to a variable

mutation rate when 3 identical individuals represent the best solution, this method seemed to provide many identical best solutions that were not significantly affected by higher mutation rates. This can be seen in figure h), where several consecutive variable mutation rates were applied. Although this decreased the mean and the minimal fitness values, the maximum values remained unchanged. Further, the amount of identical best solutions had to be higher than 2, to trigger the variable mutation again.
As this method reached the best possible solution, has good exploiting and exploring abilities and was about 5 minutes faster than the fixed selection method, which also reached the best possible solution, this input configuration scheme is with current parameters considered to be the best one available.

The results of the pretests for Case 1 are shown in the following table. The values are displayed with a color traffic light system, with good values shown in green and bad values in red.

	Fixed Selection	Variable Selection	Crossover Random	Elitism Excluded	Probabilistic Adjustment
Energy Output in kW	17307,26	16998,2	16901,61	16766,06	17307,26
Efficiency Rate in %	81,27	79,82	79,37	78,73	81,27
Duration in min.	30	60	20	15	25
Minimal Turbine Distance	90	90	90	90	90
Mean Turbine Distance	366,56	358,02	349,39	343,05	366,56
Sum Crossover Parts	203	182	206	200	203
Amount of Variable Mutation	14	8	13	8	30
Amount 100% Selection	11	5	13	21	16

Table 5: Color graded results from the pretests for Case 1

It can be seen, that the algorithm worked best with the default inputs including the probabilistic adjustment method.

4.2. Case 2: Reference Shape

The optimization problem in Case 2 refers to a widely used wind farm layout problem. The considered wind farm has a total area of 2km x 2km and is divided into 100 squares with a resolution of 200m x 200m. The following input values were used in the literature.

Wind speed velocity	12 m/s
Wind Direction	Unidirectional uniform wind
Hub Height	60 m
Rotor Radius	40 m
Thrust Coefficient Ct	0.88
Surface Roughness	0.3 m
Air Density	1.2253 kg/m^3

Table 6: Parameters and characteristics of previous studies (Shakoor, et al., 2015)

Some of the best resulting wind farms in literature are shown in the following illustration.

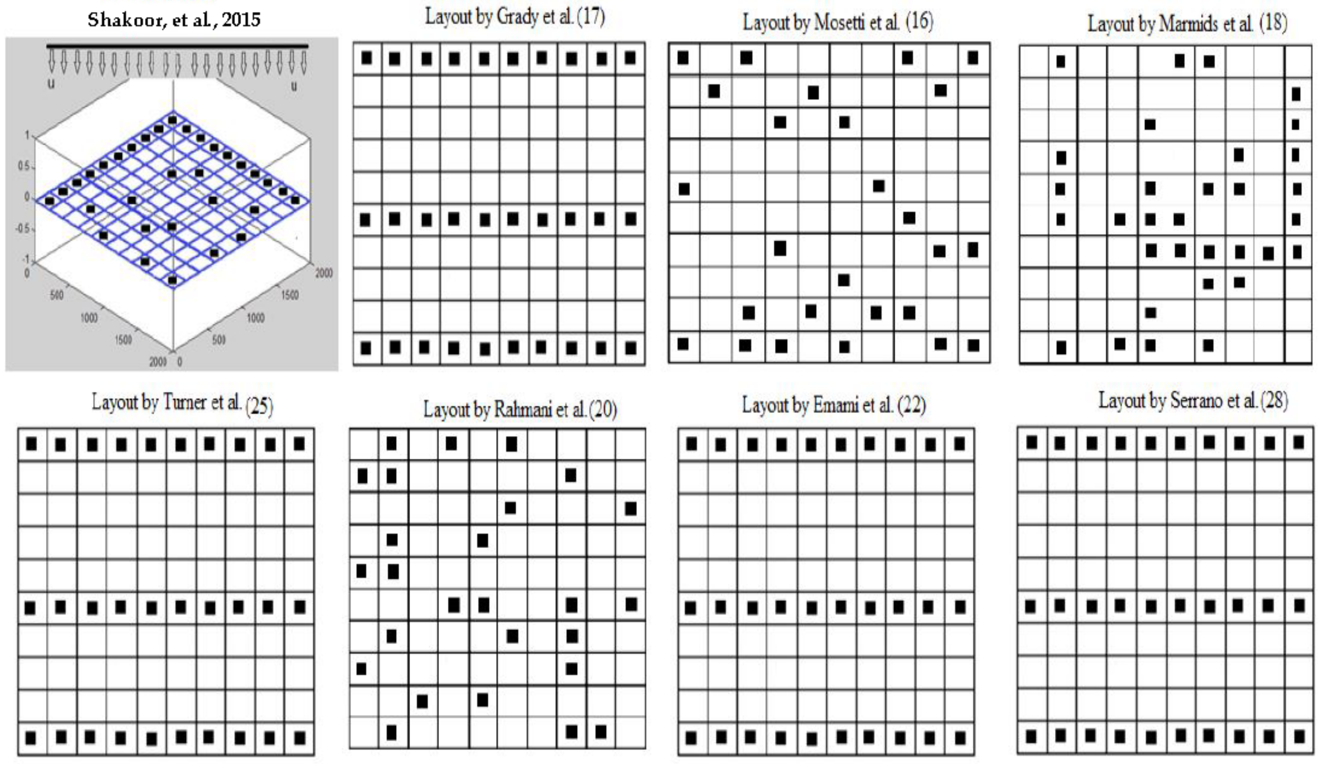

Shakoor, et al., 2015
Layout by Grady et al. (17)
Layout by Mosetti et al. (16)
Layout by Marmids et al. (18)
Layout by Turner et al. (25)
Layout by Rahmani et al. (20)
Layout by Emami et al. (22)
Layout by Serrano et al. (28)

Figure 32: Optimal layouts by earlier studies (Shakoor, et al., 2015)

Reference	Number of Turbines	Energy Output in kW	Efficiency rate in %
Shakoor, et al.	32	16251	97.70
Grady, et al.	30	14310	92.015
Rahmani, et al.	26	12819	95.11
Mosetti, et al.	26	12352	91.645

Table 7: Results of optimal layouts by earlier studies (Shakoor, et al., 2015)

The efficiency rate for a wind turbine used by the previous studies was 40% in comparison to the efficiency rate of the proposed method, which is 59.3% (Betz-value). For this reason, the energy values are not directly comparable, but also because some inconsistencies regarding the correct rotor radius were found in the work of Grady & Hussaini (2005), Ituarte-Villarreal & Espiritu (2011) and Shakoor (2015) as they obviously mistook a rotor diameter of 40m with a rotor radius of 40m. Grady & Hussaini (2005) used a turbine rotor radius of 20m in the formula for turbine power production as shown below:

$$P_p = \frac{40}{100} \times \frac{1}{2} \times 1.2 \times \pi \times 20^2 \times u_o{}^3 kW \qquad (3)$$

$$P_p = 0.3 \times u_0{}^3 kW \qquad (4)$$

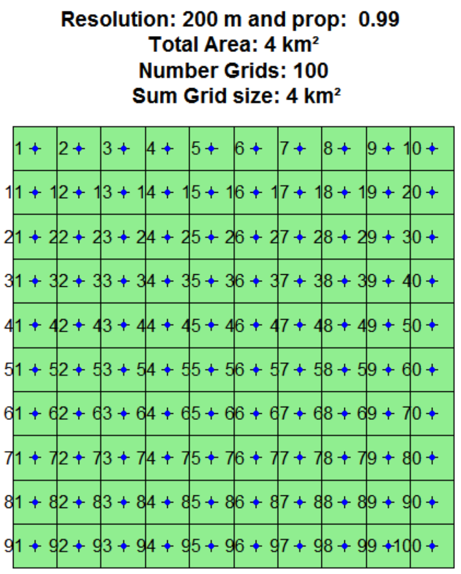
Figure 33: Formula for turbine power production according to Shakoor (2015) and Grady (2005)

If all values except the wind speed (u_o) of equation (3) are multiplied together, a value of 301.59W is obtained. Dividing this by 1000 yields 0.3kW, as in equation (4). This coefficient is still only valid for a rotor radius of 20m. Several other papers were found, including Saavedra-Moreno (2011) and Guirguis (2016) that used equation (4) as power production formula. The rotor radius used by previous studies had therefore to be 20m which would conform to the resulting energy outputs in Table 7. The tests for Case 2 were however done with a radius of 40m, resulting in higher energy outputs, lower efficiency rates and different layouts, as this problem was noticed after the tests were done.

4.2.1. Constant Wind Speed and Uniform Wind Direction

Input Variable	Value
n	30
SurfaceRoughness	0.3
Rotor	40
fcrR	5
RotorHeight	60
referenceHeight	60
iteration	100
Proportionality	0.99
mutr	0.008
vdirspe	data.in
topograp	"FALSE"
elitism	"TRUE"
nelit	7
selstate	"FIX"
crossPart1	"EQU"
trimForce	"TRUE"

Output of *GridFilter*



The data frame (*data.in*) containing the wind speed information has the following inputs and the resulting wind rose illustrated below on the right.

Input Wind data frame **Windrose of *data.in***

```
> data.in
  ws  wd  probab
  12  0     25
  12  0     25
  12  0     25
  12  0     25
```

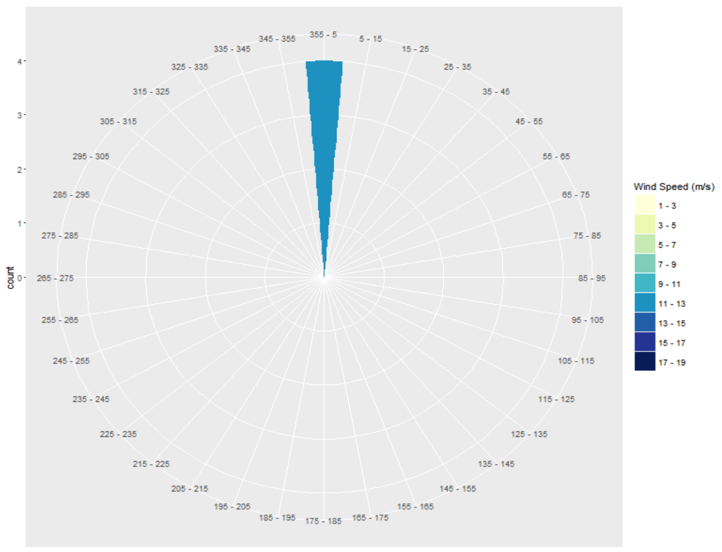
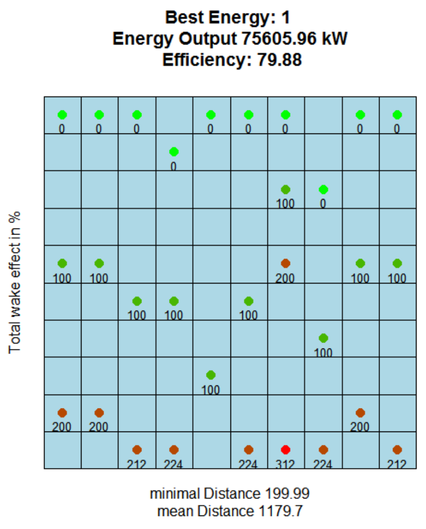
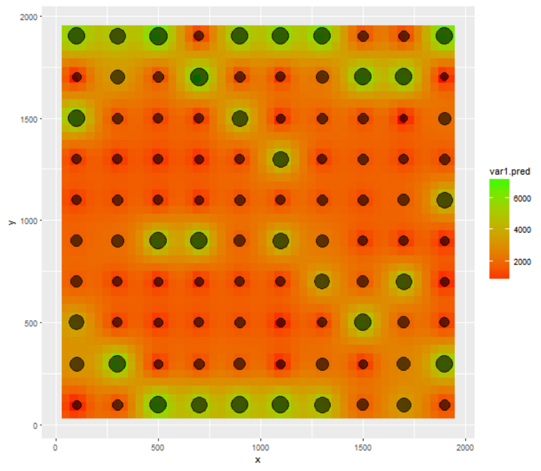


Figure 34: Input values for Case 2 - Reference Shape and uniform wind direction

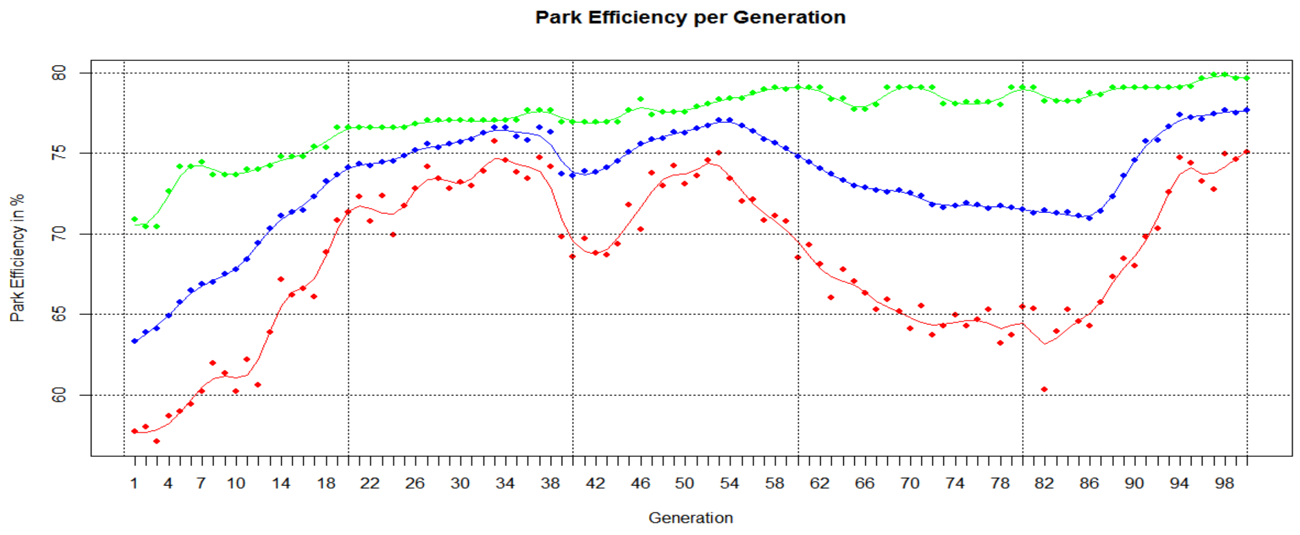
a



b



c



d

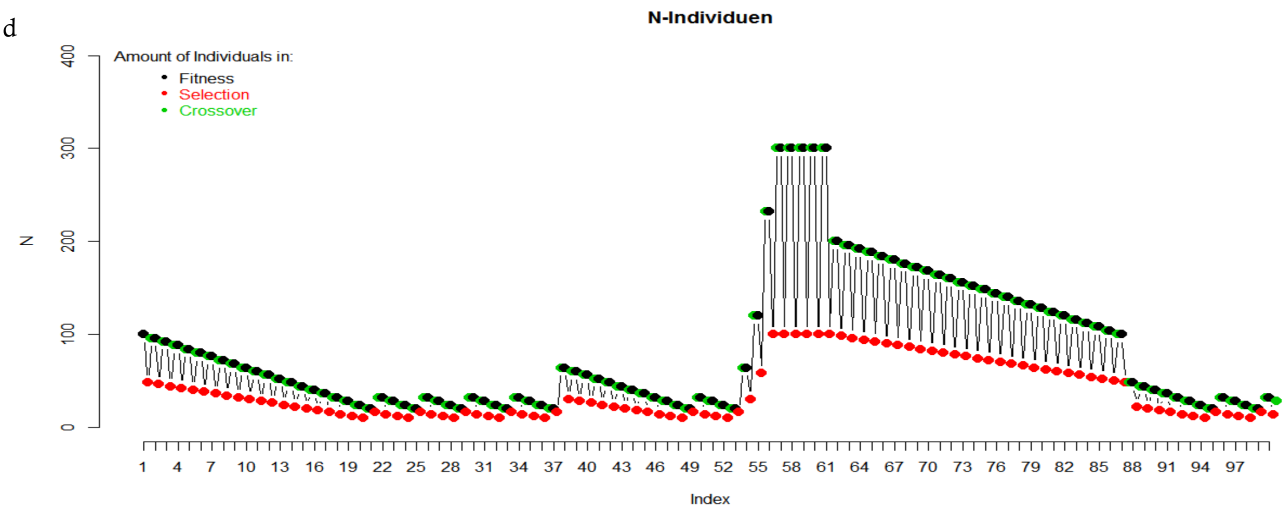
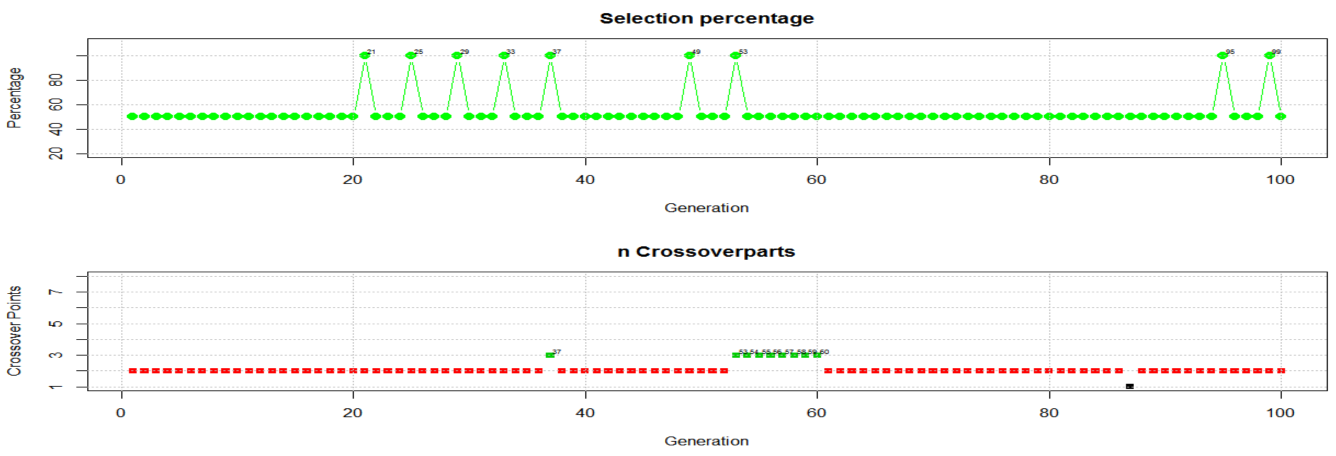
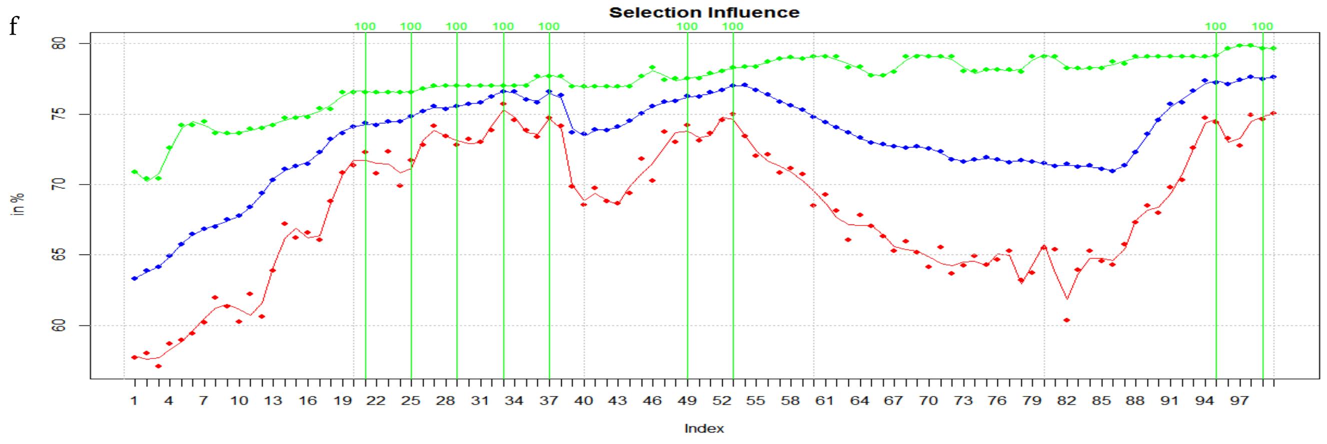


Figure 35: Results of Case 3.2.1.: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

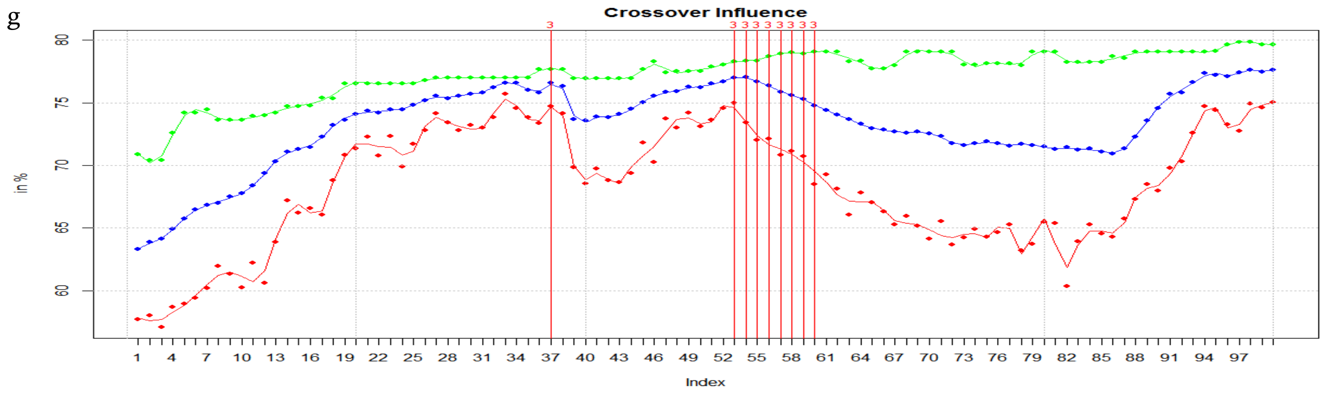
e



f



g



h

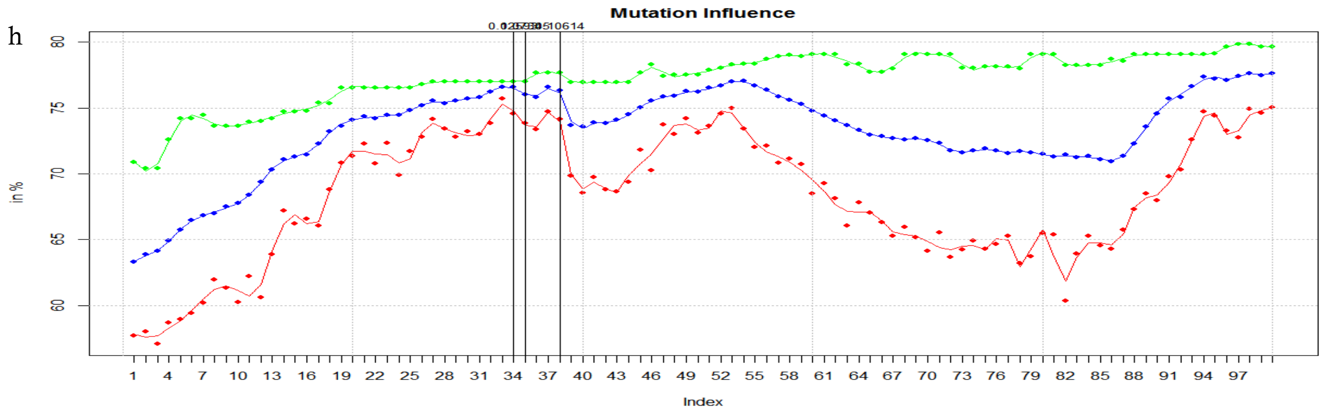


Figure 36: Results of Case 3.2.1.: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The best possible solution to this problem is shown in Figure 32 with the layouts from Grady, Turner, Emami and Serrano.
Shakoor achieved a layout with higher energy yield by rotating the squared area under consideration by 45 degrees, which can be seen in Figure 32. With this new orientation of the area, Shakoor was able to place 2 edges of the square facing north while the remaining layouts had only 1 edge of the square facing north. He could therefore place 19 turbines on locations without any influential wake effects, whereas the other layouts were restricted to a maximum of 10 turbines without any influential wake effects. Since the area in the proposed method is considered constant and cannot be rotated at will, the results obtained cannot be compared with the results achieved by Shakoor.
The solution found by this method did not reach the best possible solution, but it can be seen that the algorithm also tried to align 3 rows of 10 turbines. The second and third rows were placed with shorter distances to the turbines in front, as the higher rotor radius of 40m resulted in higher wake diameters.
The efficiency values of this run converged until about t=37, wherein the crossover parts were first increased to 3, which in combination with a higher mutation rate resulted for the next 10 generations in a strong increase of the variance of the fitness values. The algorithm then converged again until t=53, where the crossover parts were again increased to 3 for the following 8 generations. Interestingly, the average and minimum efficiency values decreased for these 8 generations, while the maximum values rose almost constantly. After the crossover part dropped to 1 in t=87, the variance of the efficiency values and the population size decreased considerably, and the algorithm then converged again to the end where it found the best solution.
As the variances after iteration 37 and especially after iteration 53 were quite big, the algorithm was not able to generate more than 3 identical best solutions

to activate the variable mutation rate. It was only used 3 times at t=34, 35 & 38. If the algorithm was able to develop further, for instance 200 iterations, then the variable mutation rate would probably have been activated more often, which could have lead to superior solutions as the following best solution after the same test with *iteration* = 300 shows:

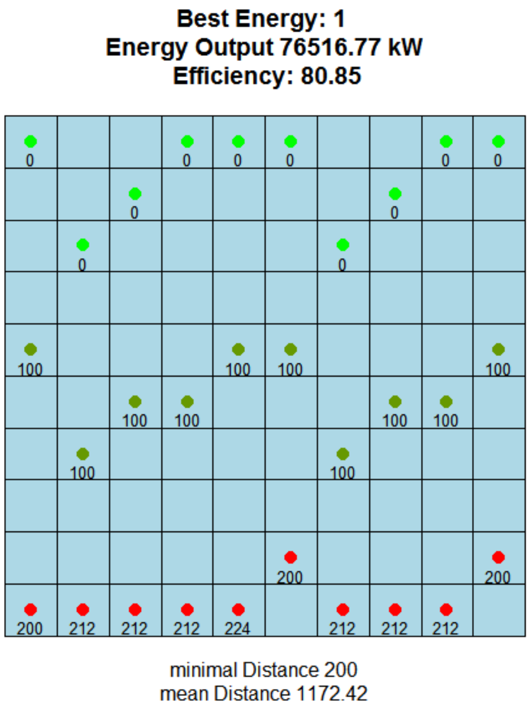


Figure 37: Resulting best individual for case 4.2.1 after 300 iterations

The expected energy output after 300 iterations is 910 kW and the efficiency is 0.97% higher. It can also be seen that the algorithm was able to place 3 turbines in each column, which is identical to the best possible solution of this problem, although the specific arrangement differs due to higher wake diameters. The best solution after 100 iterations had one column with 4 turbines and therefore higher wake effects. The optimization run for the case with 100 iterations took about one and a half hours, while the test with 300 iterations took about 5 hours.

The same test was also done with a rotor radius of 20m and 100 iterations to be able to compare the results to previous studies. The best resulting individual for this test can be seen on the following figure.

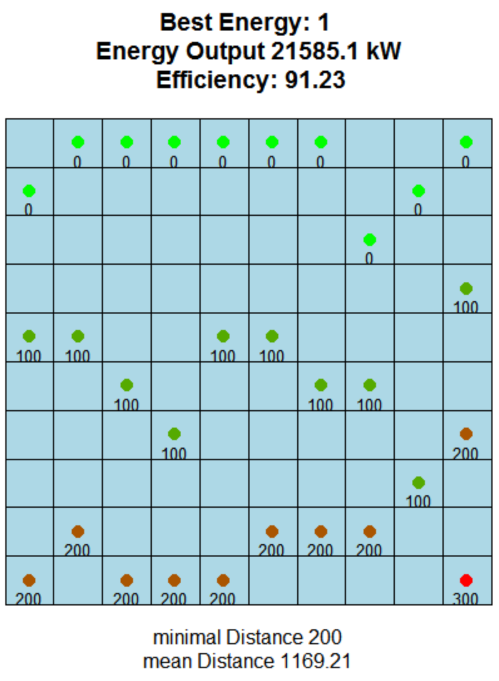


Figure 38: Resulting best individual for case 4.2.1 although with rotor radius of 20m

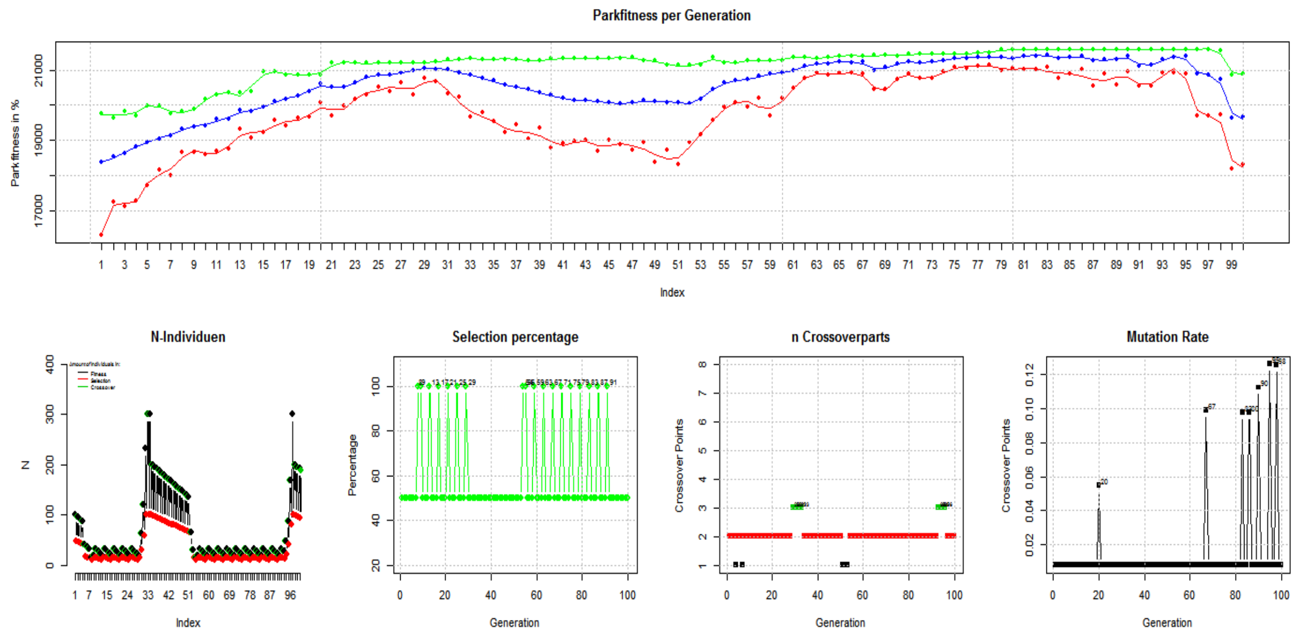


Figure 39: Evolutionary data for best individual of case 4.2.1 with rotor radius of 20m

If the energy output of this best individual is multiplied with an efficiency of 40% instead of 59.3% in the proposed method, the energy output decreases from 21585.1 kW to 14599.9 kW. This output is still higher than Grady's result although with a lower efficiency which is due to the differing power production formula in Figure 33. The value 0.3 is only valid for an efficiency rate of 40%, a rotor radius of 20m and an air density of 1.2, resulting in 0.30159 ($0.4*0.5*1.2*20^2*\pi$ / 1000 = 0.30159), which is then truncated to 0.3 and multiplied with the third power of the wind velocity. The proposed method has a default air density value of 1.225 resulting in a value of 0.30787 which is then multiplied by the third power of the wind speed. If the energy output of this best solution would be multiplied by 0.3 instead of 0.30787, the energy output would further decrease from 14599.9 kW to 14187.5 kW. This energy output is now comparable to the energy output of Grady in Table 7, which is the only one with 30 turbines. The energy output and efficiency rate of the proposed algorithm are about 0.8% lower compared to Grady's solution.

4.2.2. Constant Wind Speed and Variable Wind Direction

The wind farm is optimized with 4 equally distributed wind directions and constant wind speeds of 12m/s as shown below. Additional, the same input values of chapter 4.2.1 were used for this optimization run with 40m rotor radius.

Input Wind data frame | **Windrose of *data.in***

```
> data.in
  ws   wd    probab
  12    0      25
  12   90      25
  12  180      25
  12  270      25
```

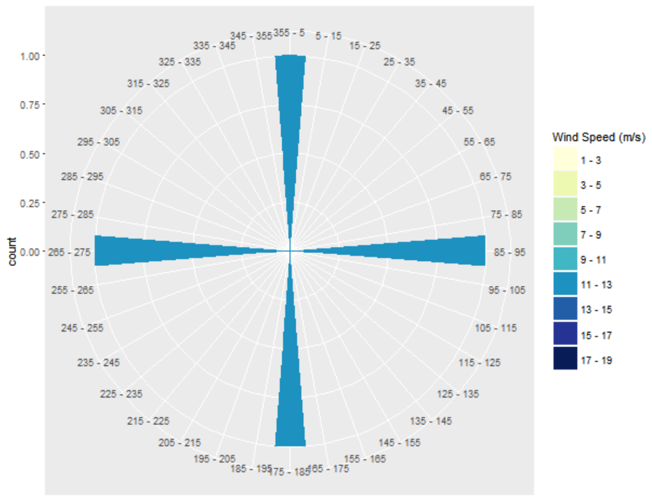
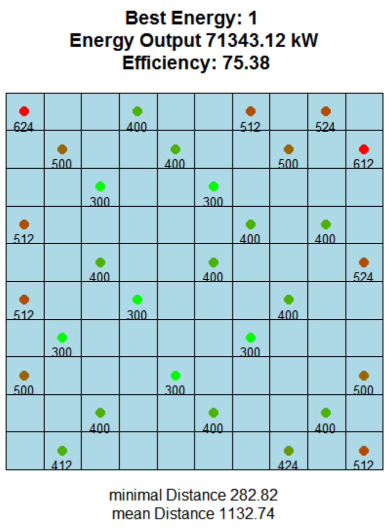
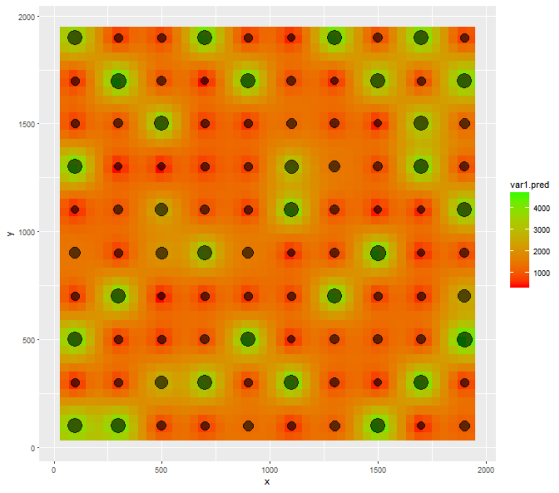


Figure 40: Input wind data for Case 2 - Reference Shape and multiple wind directions

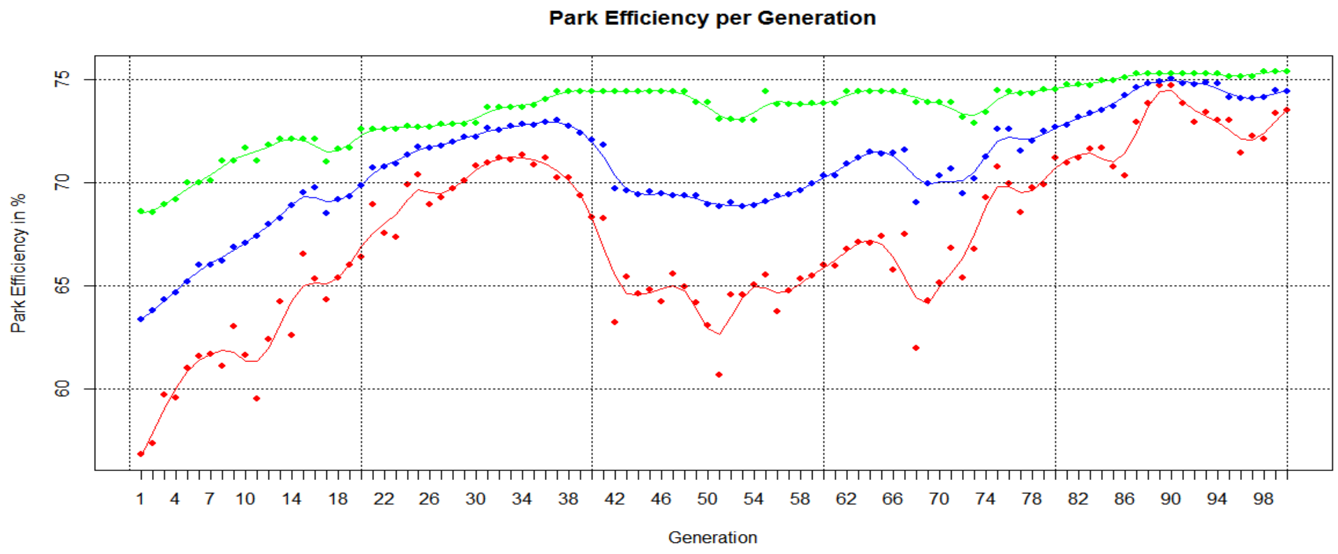
a



b



c



d

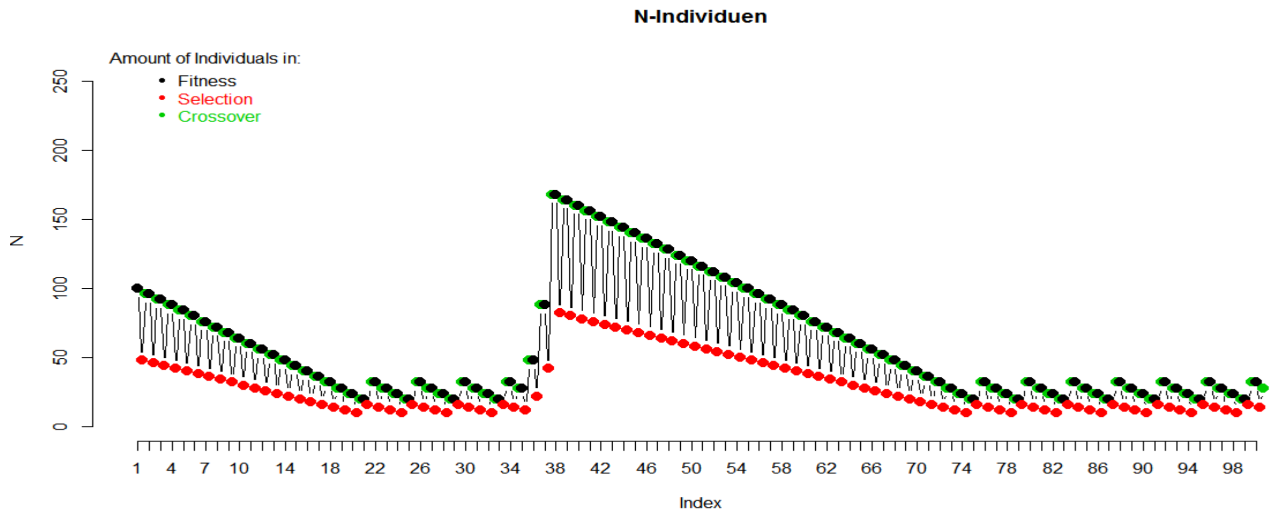
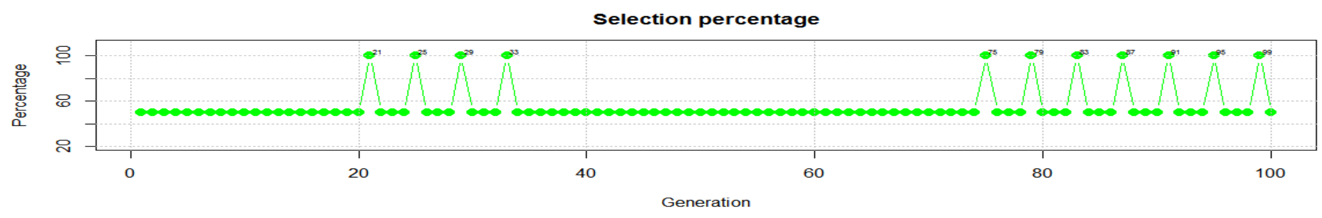
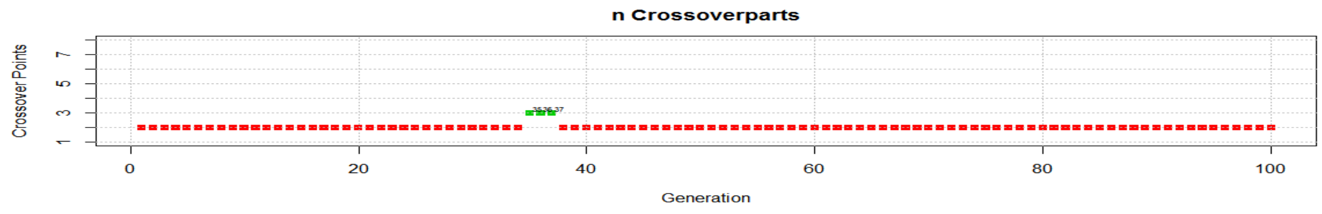


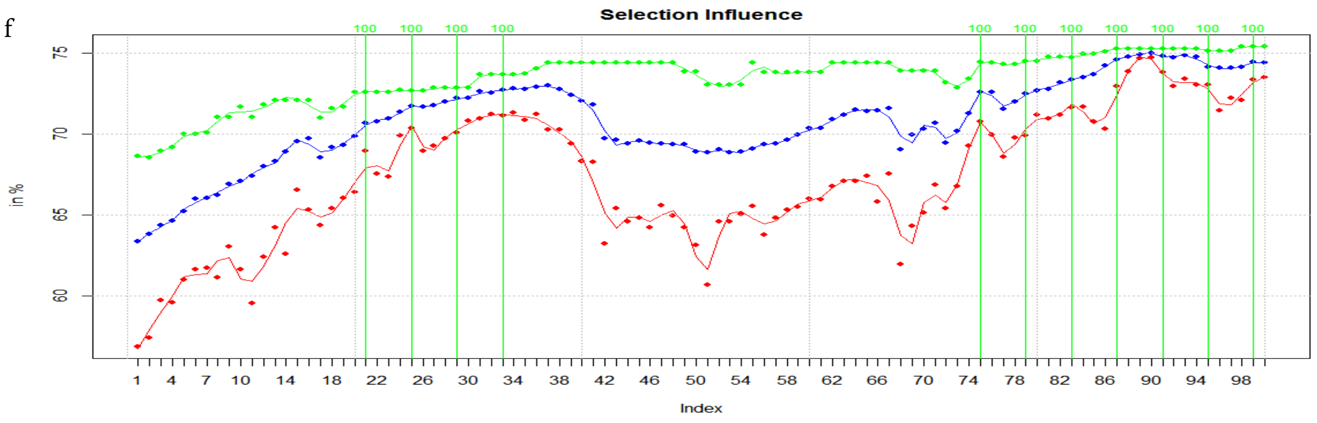
Figure 41: Results of Case 3.2.1.: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

e

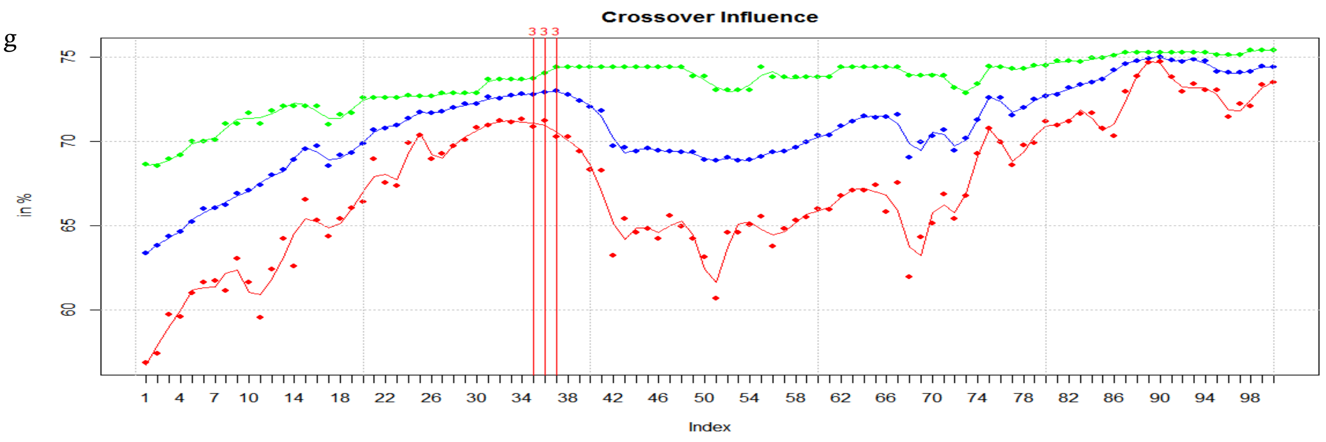




f



g



h

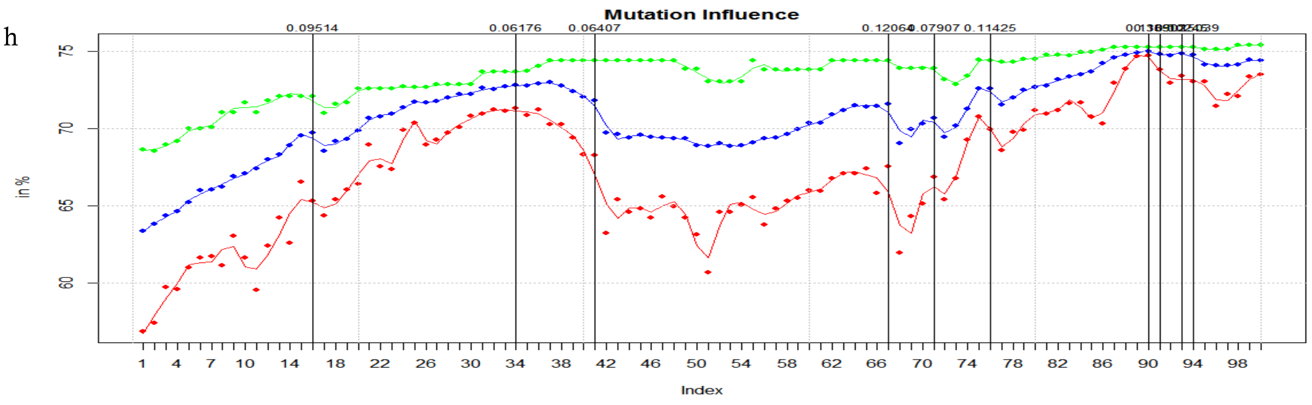


Figure 42: Results of Case 3.2.1.: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

As layouts with a single wind direction are quite easy to understand and probably also easier and faster to optimize by hand, correct interpretation and optimization of layouts with several wind directions becomes more complex. Since in this example wind has come uniformly from all four directions of the sky, no grid cell with a turbine should have a directly adjacent grid cell with a turbine. The best-found solution shows such a layout that is also more spread over the whole area and not as regularly arranged as for a layout with single wind direction. Nevertheless, with the given problem and the resulting layout, it is not so easy to determine visually whether a better result would be possible or not.

The efficiency values increased and converged until t=37, where the crossover parts increased to 3 for 3 times in a row, which then increased the variance of the fitness values and the population sizes. The variance remained high up to the ~70th generation, where it began to decline again due to low population numbers. The peak of this decrease is at t=90, where the maximum, mean and minimum value were quite close together. Although the variation values between t=80 and t=100 were quite low, the algorithm was able to find several better individuals during this time.

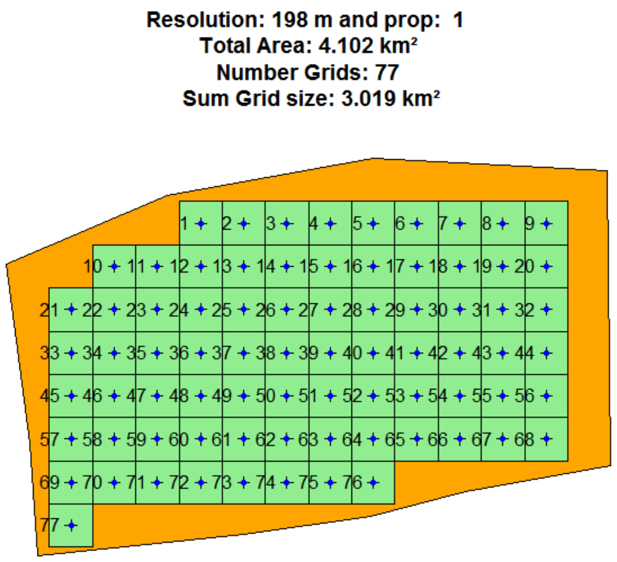
The calculation time for this test was the highest with about 5 hours in total, since 4 wind directions had to be evaluated for each wind farm compared to one single wind direction in the other tests.

The previous tests and most of the research in literature took a rectangular shape as area to be optimized, although the real space available on which a wind farm is actually to be built, will most likely not be only rectangular. The available area can be restricted, among other things by property rights, nature protection or infrastructure. The algorithm is tested in the following chapter with an irregular area, in which the terrain effect model is investigated.

4.3. Case 3: Wind Farm "Tauern"

Input Variable	Value
n	14
SurfaceRoughness	0.3
Rotor	33
fcrR	6
RotorHeight	60
referenceHeight	60
iteration	100
Proportionality	1
mutr	0.008
vdirspe	data.in
topograp	"FALSE"
elitism	"TRUE"
nelit	7
selstate	"FIX"
crossPart1	"EQU"
trimForce	"TRUE"

Output of *GridFilter*



The data frame (*data.in*) containing the wind speed information has the following inputs and the resulting wind rose illustrated below on the right.

Input Wind data frame

```
> data.in
  ws   wd   probab
  12   0       25
  12   0       25
  12   0       25
  12   0       25
```

Windrose of *data.in*

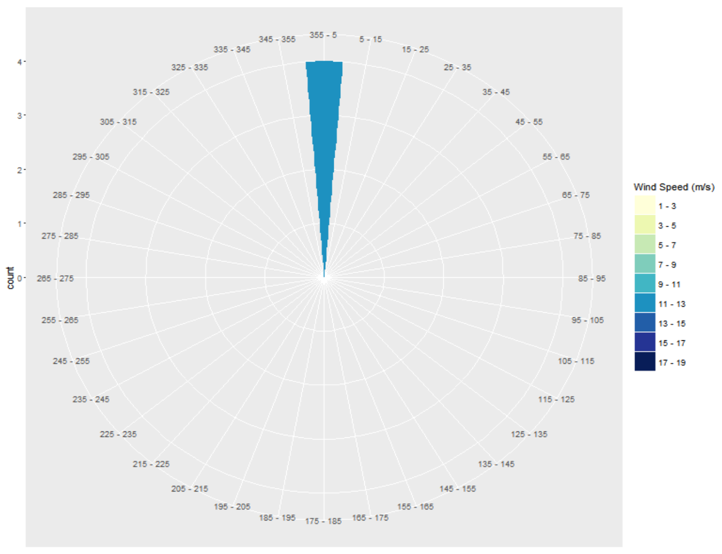
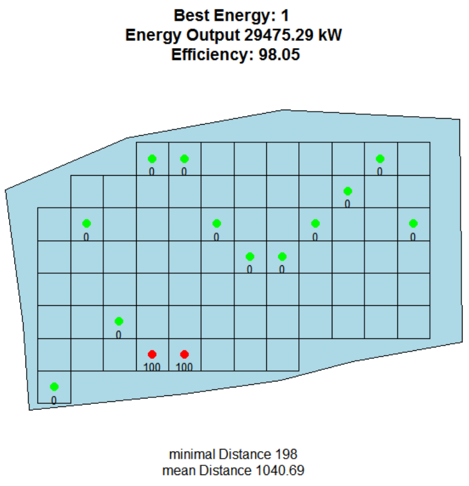


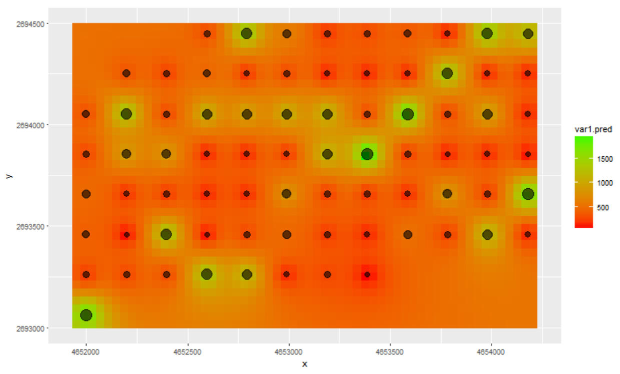
Figure 43: Input values for Case 3 - Wind Farm "Tauern"

4.3.1. Terrain Effect Model Excluded

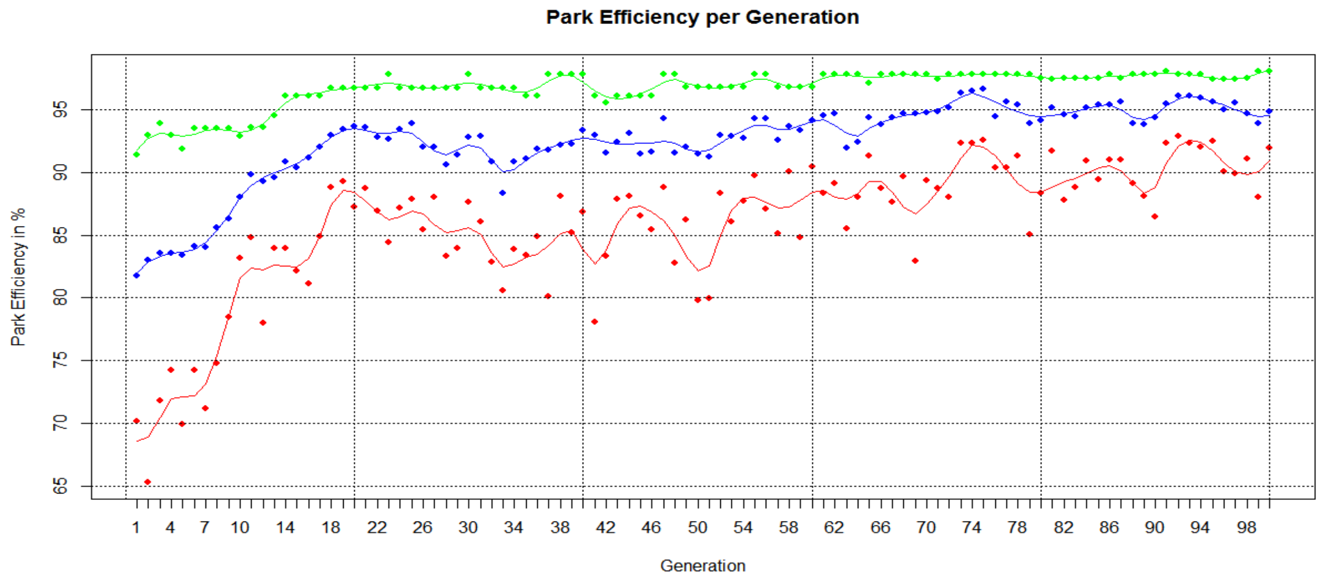
a



b



c



d

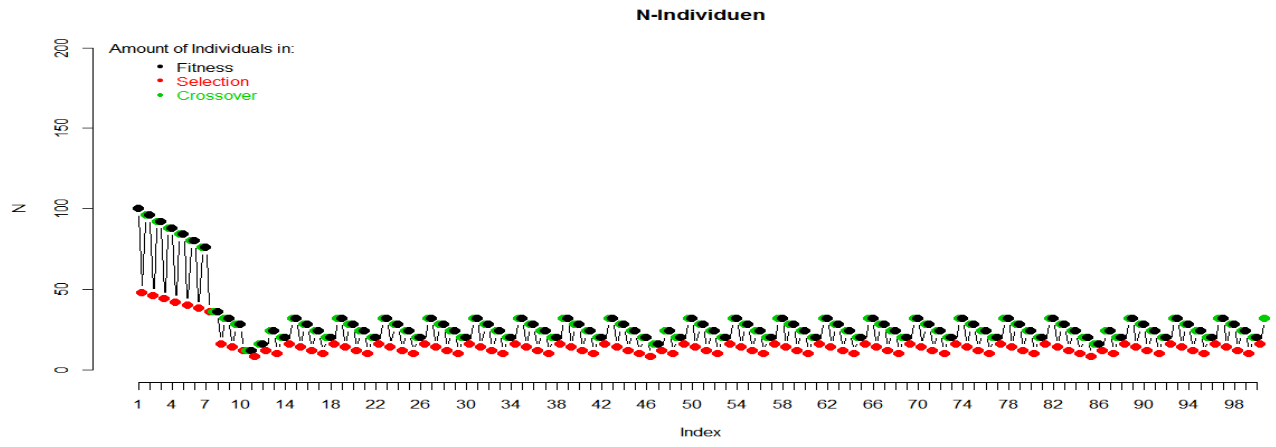
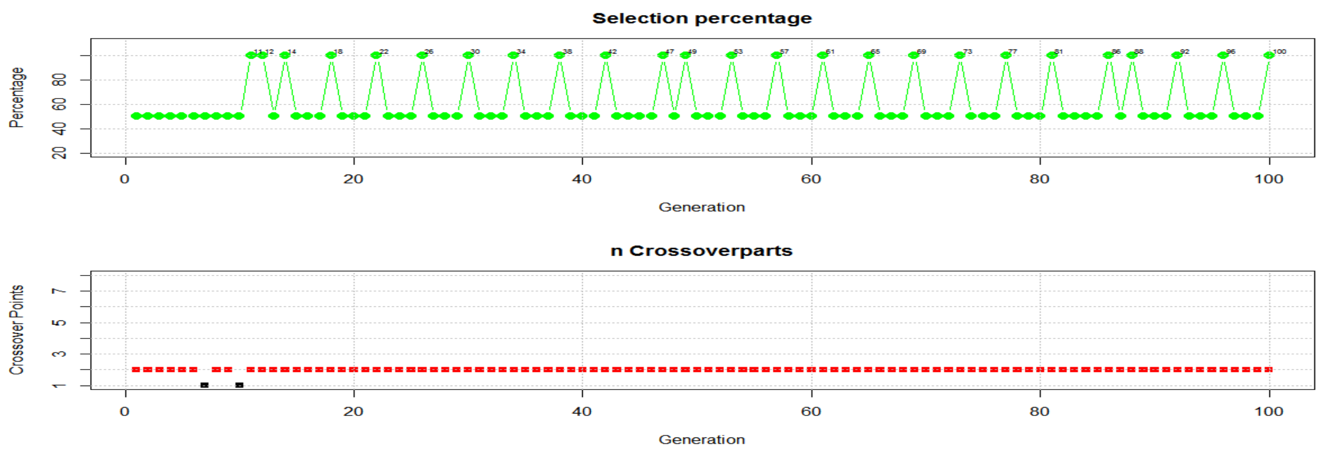
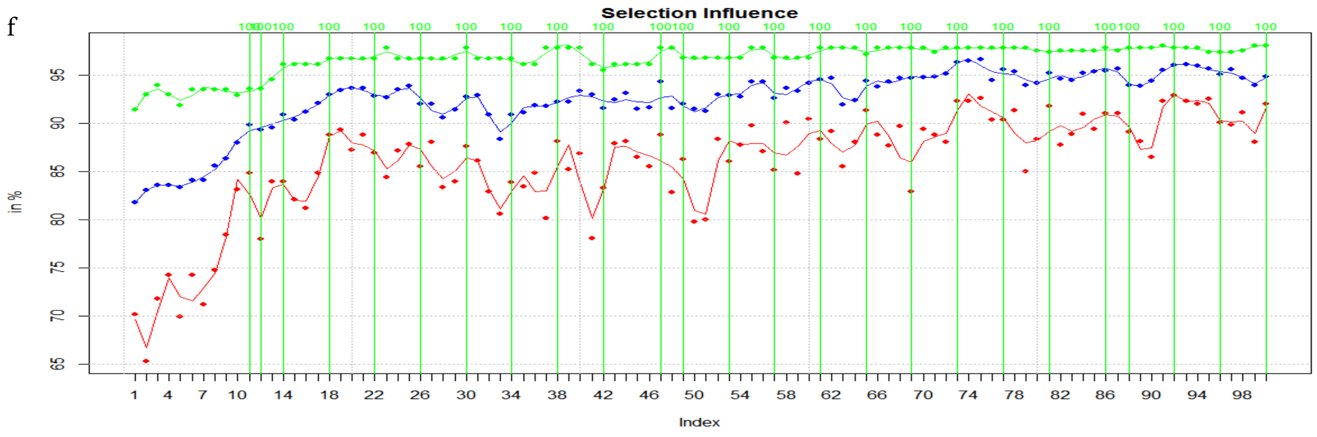


Figure 44: Results of Case 3.3.1.: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (exactly proportional to energy yield) *d)* Amount of individuals after every evolutionary method

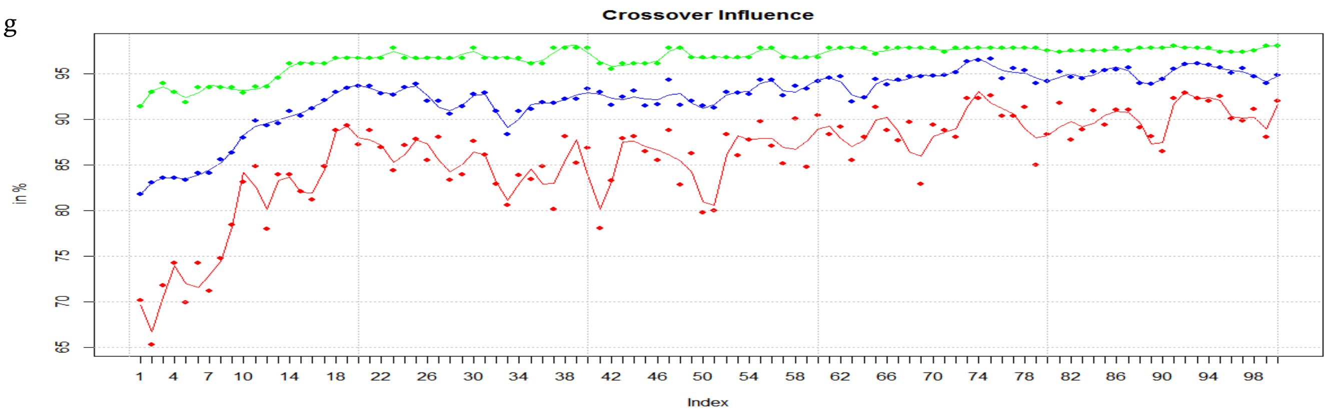
e



f



g



h

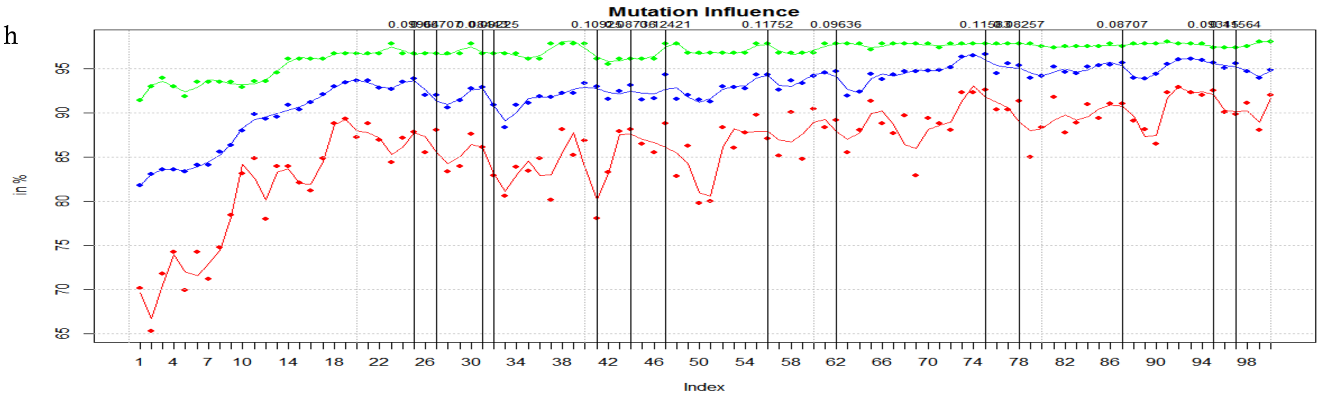


Figure 45: Results of Case 3.3.1.: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The best-found individual can be regarded as an optimal solution, as 12 of the 14 wind turbines can be aligned next to each other without any wake influences and only 2 turbines have to be placed in the wake of other turbines. Those 2 wake influenced turbines are placed with a maximum possible distance to the influencing turbines, which is good in terms of wind speed recovery.

Although the algorithm jumped back to 1 crossover part in the early stages which lead to a decrease in the population size and in the spreading of the individual fitness values, an optimal solution was already found at t=23. The amounts of individuals in the populations were quite low during the whole evolution process and the selection percentage had to be increased several times, which did not lead to an increase in the crossover parts. Besides, 2 crossover parts were sufficient to solve this problem and although the variable mutation rate was activated 14 times, it was not needed to reach the optimum at t=23. This optimization run took about 15 minutes. The best-found individual is displayed above an aerial image of the real wind farm, even though the wind inputs for this optimization run were pure assumptions and do not represent real conditions, making a direct comparison difficult.

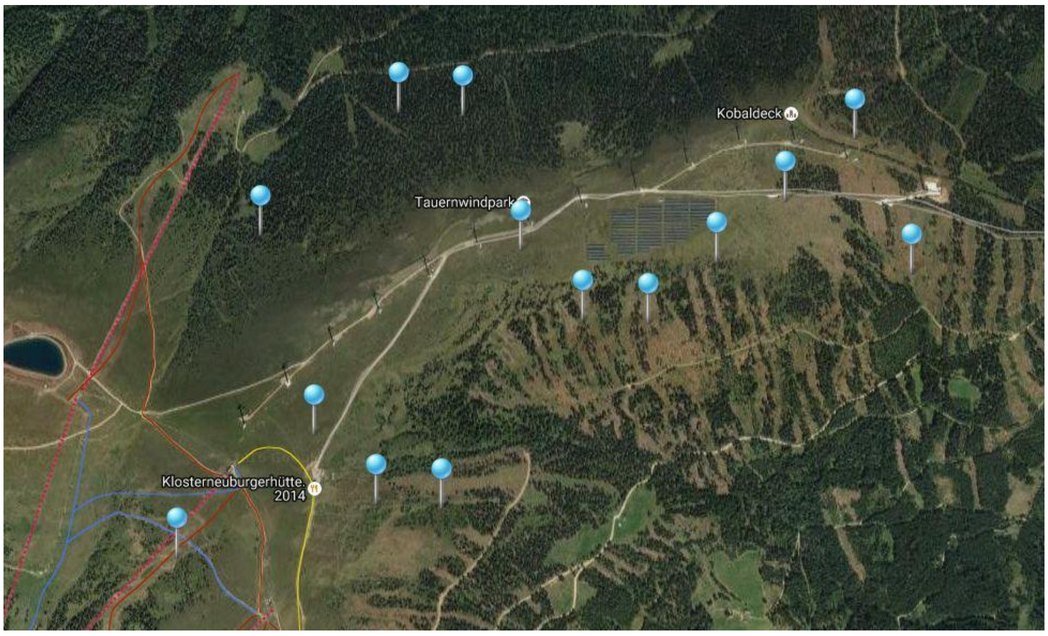
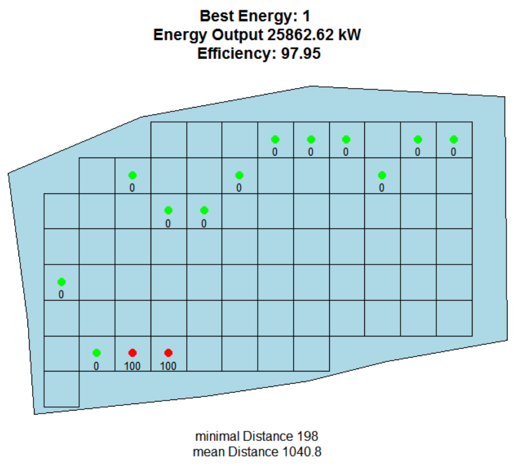


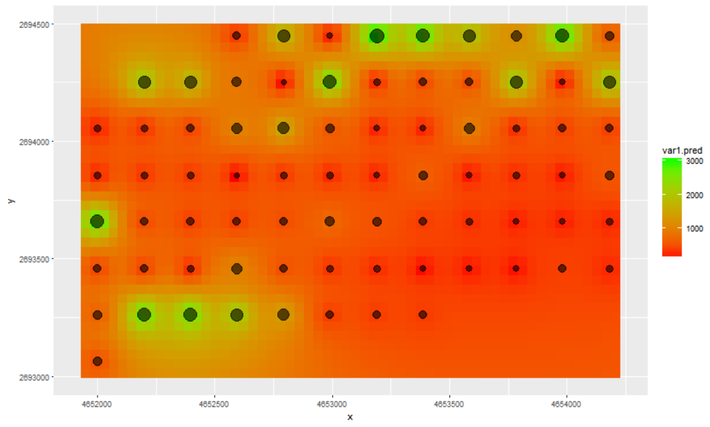
Figure 46: Best-found solution of wind farm "Tauern" neglecting terrain effects

4.3.2. Terrain Effect Model Included

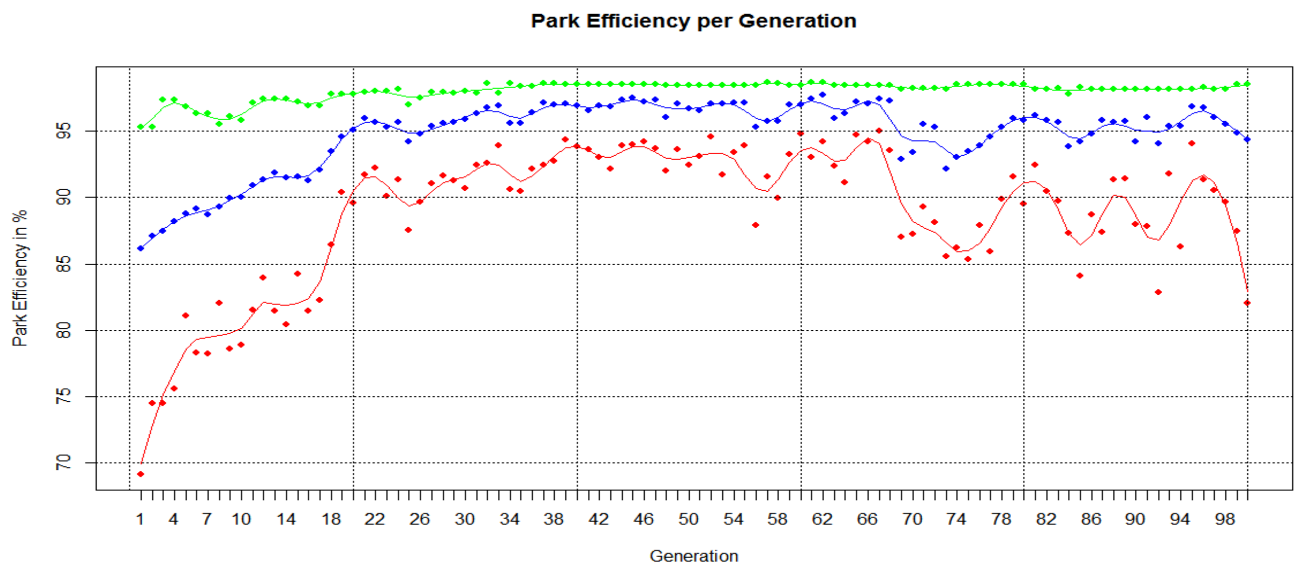
a



b



c



d

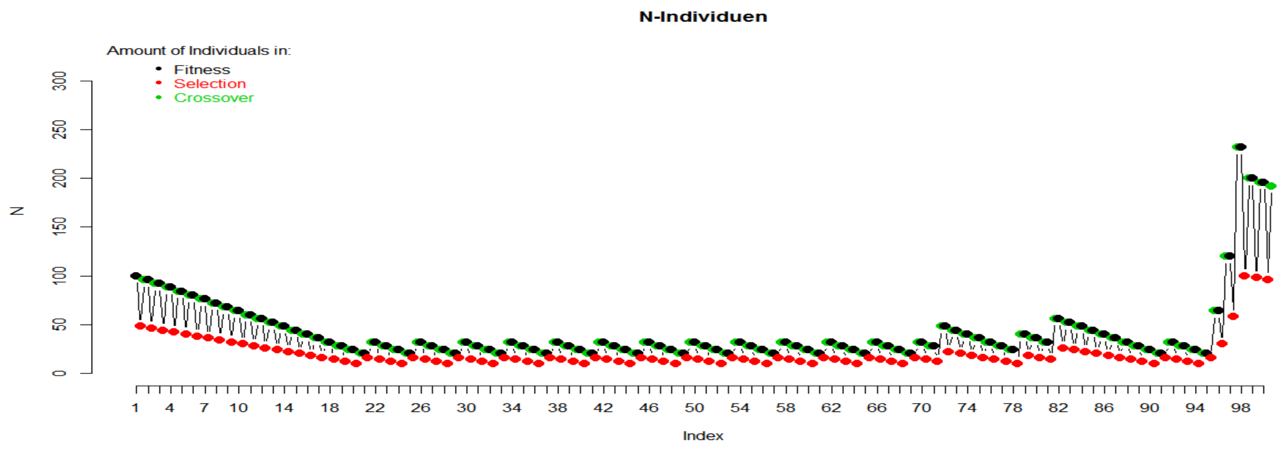
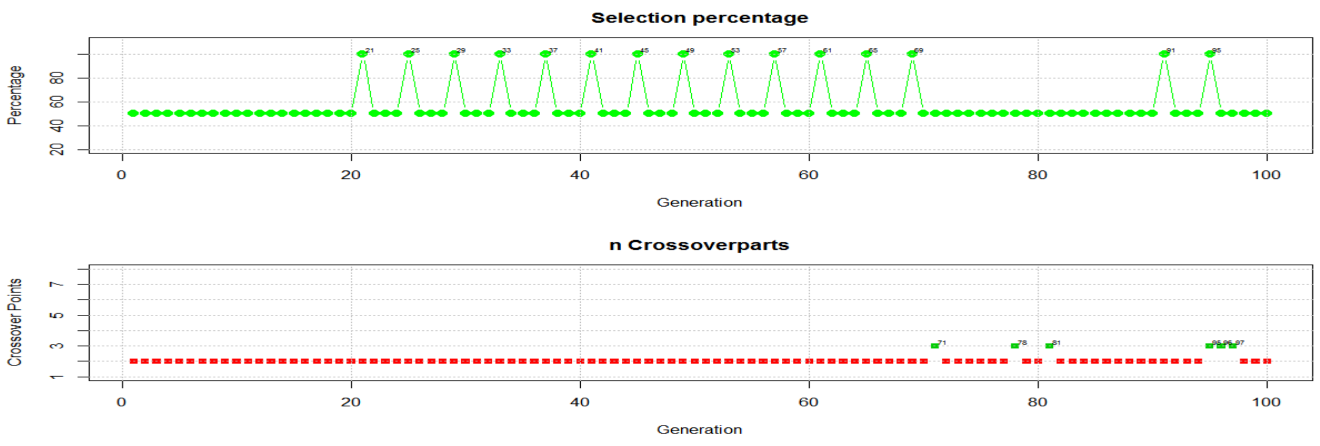
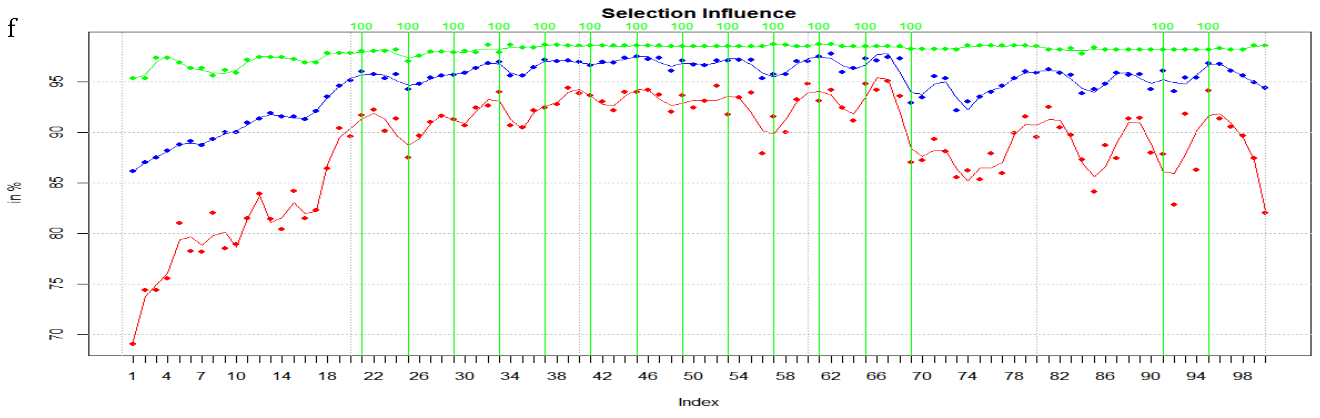


Figure 47: Results of Case 3.3.2.: *a)* The best-found solution of the algorithm *b)* Interpolated heat map of all used individuals *c)* Evolution of wind farm efficiencies (not proportional to energy yield) *d)* Amount of individuals after every evolutionary method

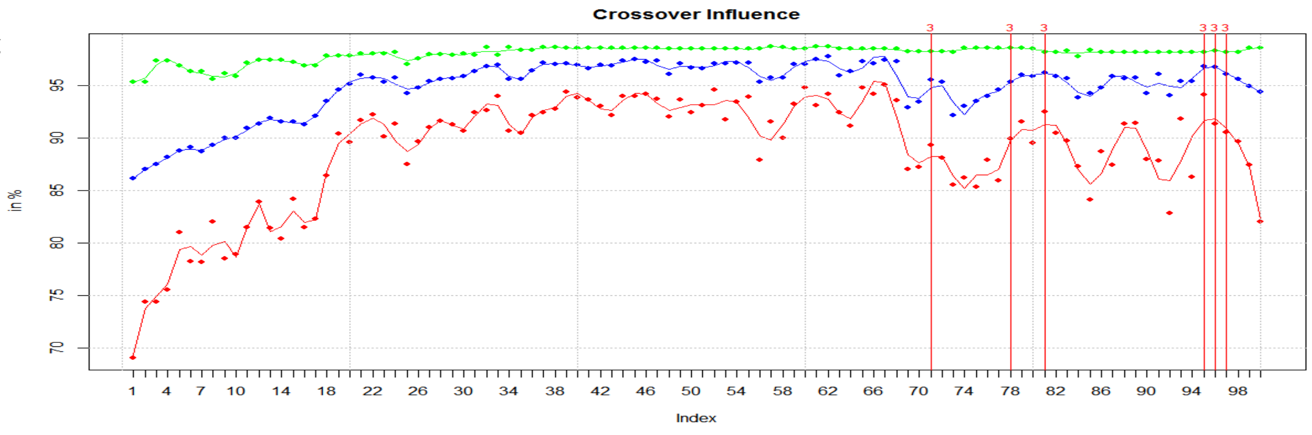
e



f



g



h

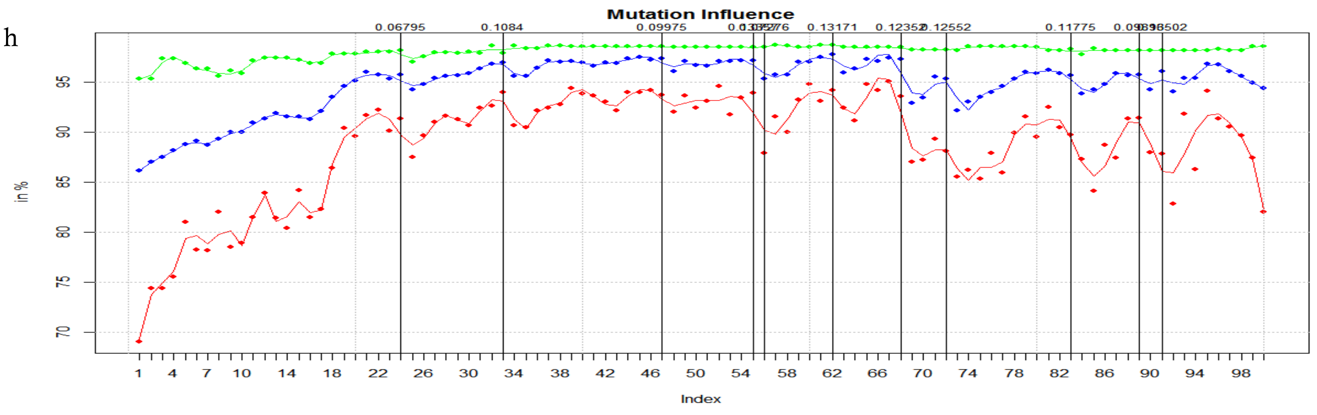


Figure 48: Results of Case 3.3.2.: *e)* The selection percentage during the evolution *f)* High selection influence on evolution *g)* Crossover influence of 3 crossover parts *h)* Variable mutation rate influence

The best-found solution of this method can only be correctly interpreted, when the terrain effects are known, but it is obvious that the energy output of the method with terrain effects (25862.62 kW) is about 12% lower than without terrain effects (29475.29 kW), although their efficiency rates are quite similar. The layout of the best individual of this method shows again 12 uninfluenced turbines and 2 wake-influenced turbines, but with a smaller distance to the wake-inducing turbines. Without terrain effects, both influenced turbines were 6 grid cells or 1188 (6*6*33=1188) meters away. Considering the terrain effects, one influenced turbine was 5 grid cells or 990m apart and the other one 4 grid cells or 792m. One possible reason for this is visible on the following illustration of the elevation and the resulting wind speed multipliers for this area.

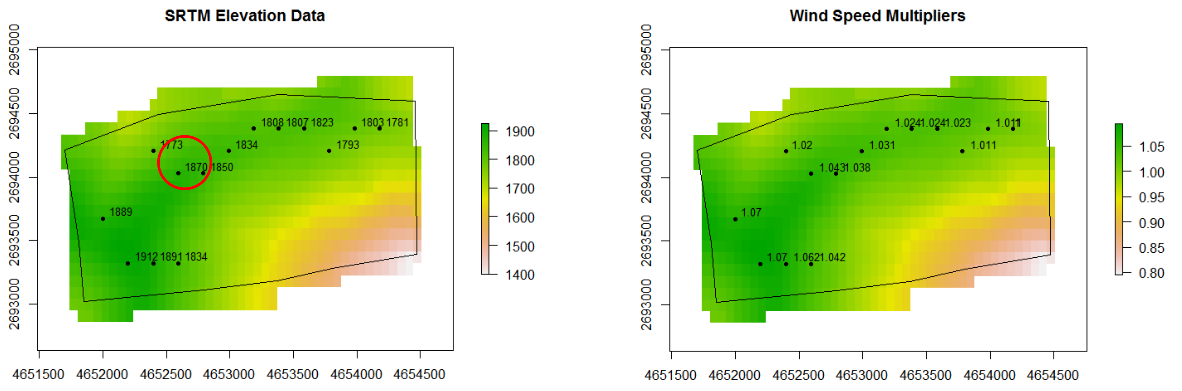


Figure 49: Elevation values and wind speed multipliers for the best individual of Case 3.3.2

The two wake-inducing wind turbines marked in the red circle are located near a hillside and have therefore multiplier values above 1. Also all other turbine locations have multiplier values greater than 1 and are positioned close to the dark-green area which indicates higher elevation. The best-found solution of the method without terrain effects, illustrated below, shows 2 turbine locations with multiplier values below 1. It can clearly be seen that the terrain properties were ignored and the turbines are not concentrated in dark green areas.

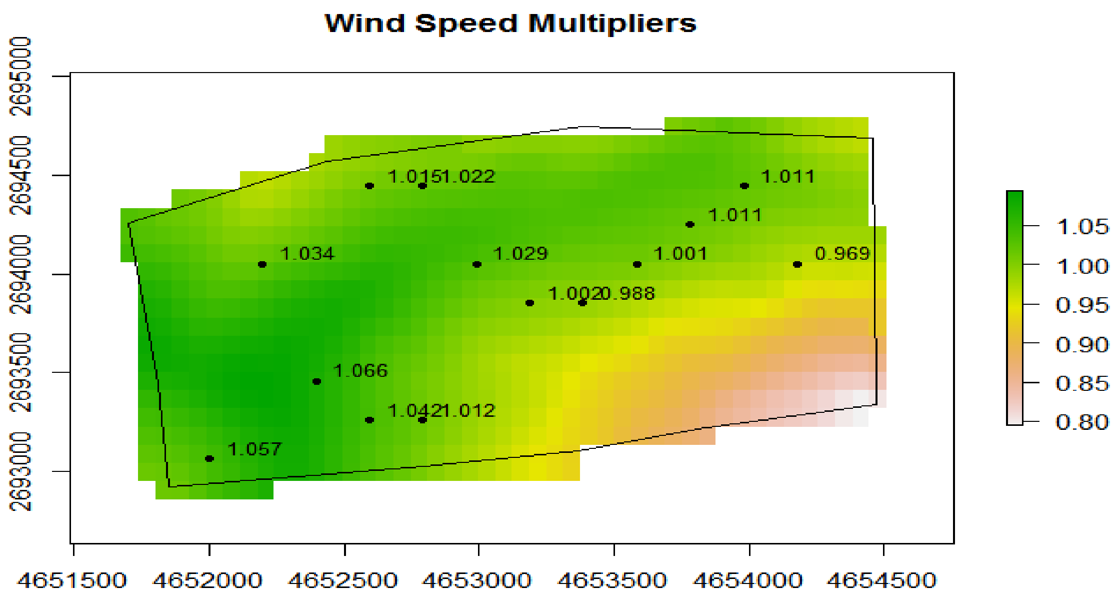


Figure 50: Wind speed multiplier values for Case 3.3.1

Although higher elevations have a positive effect on the expected energy output due to the wind speed multiplier method, they also have a negative effect as they decrease the air density as shown below.

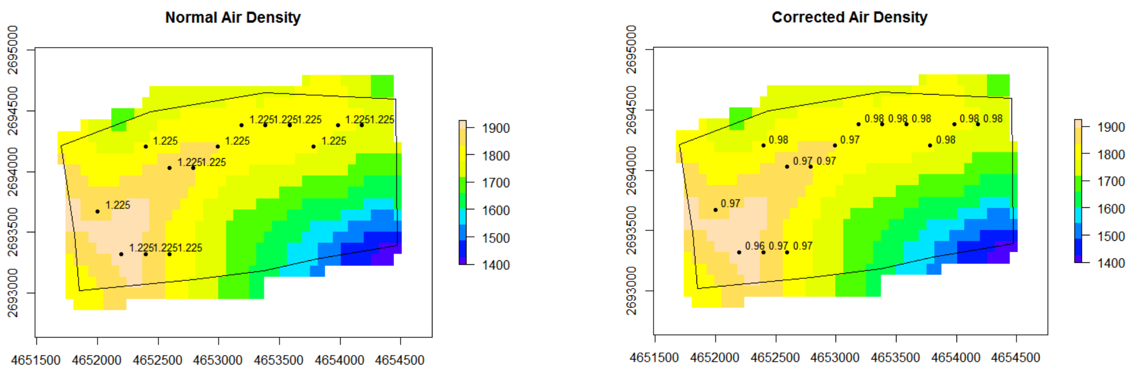


Figure 51: Normal air density and corrected air density for best individual of Case 3.3.2

The default air density is shown on the left with a value of 1.225 for each turbine, which refers to sea level. As this wind farm is situated at an elevation of about 1700m, the corrected air density values drop to about 0.98 shown on the right, which reduces the expected energy output of the wind farm by about 20%.

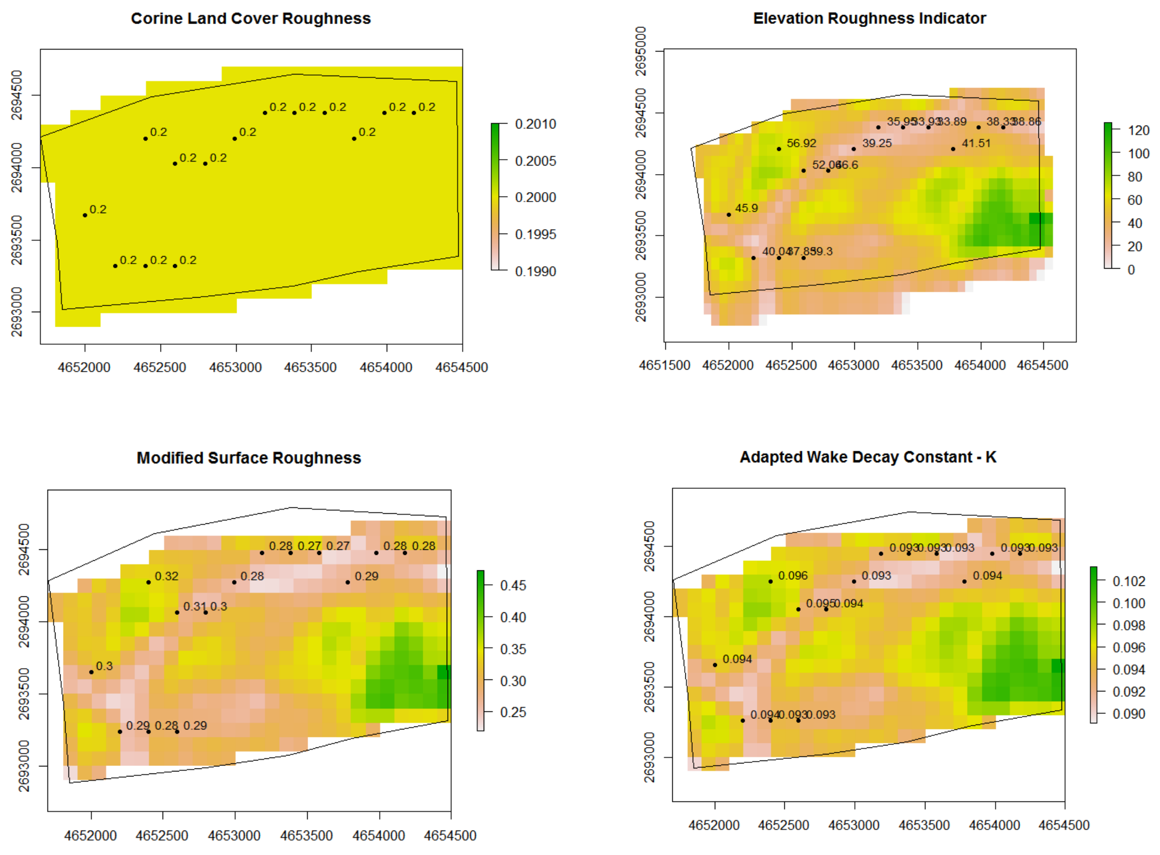

Corine Land Cover Roughness
Elevation Roughness Indicator
Modified Surface Roughness
Adapted Wake Decay Constant - K

Figure 52: *Top left*) Corine Land Cover Surface Roughness - *Top right*) Elevation roughness indicator - *Bottom left*) Modified surface roughness - *Bottom right*) Adapted wake decay constant for best individual of Case 3.3.2

The two roughness values on the upper half of the graph are used to calculate the modified surface roughness at the bottom left, which is then used to calculate the wake decay constants at the lower right.

The higher the modified surface roughness, the higher is the adapted wake decay rate. In the event that the wind reference height is lower than the turbine hub height, the wind speed increases with increasing surface roughness. If the reference height is higher than the hub height, the wind speed decreases with increasing roughness.

Furthermore, the higher the adapted wake decay value, the higher the wake diameter and the lower the wind speed deficit inside the wake. The turbines in this best individual are located in grid cells with low modified surface roughness values, resulting in low adapted wake decay rates. This means, that the wakes will not spread so quickly but at the same time the wind speeds inside the wakes will be lower. As the impact of the wake decay value on the spread of a wake with increasing distance is considerably stronger to the wind velocity recovery, the algorithm preferred grid cells with low roughness values and therefore low wake decay values with minor wake diameters. The best individual of this method is plotted on an aerial image of the real wind farm.

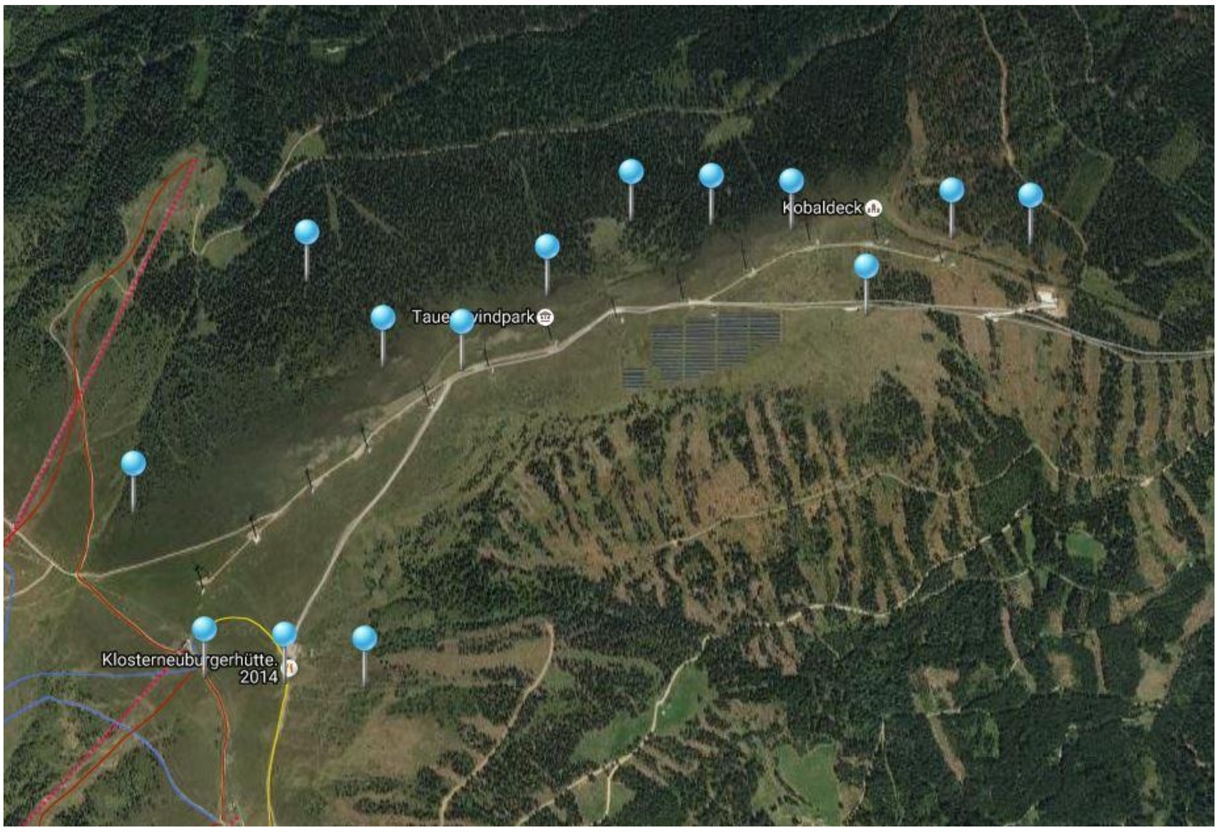


Figure 53: Best-found solution of wind farm "Tauern" including terrain effects

Although the input values, except the rotor radius, of this optimization run were pure assumptions, the layout of the proposed method is comparable to the real wind farm in terms of selected turbine locations, gaps between the turbines and general arrangement. The algorithm could be further tested with a slightly smaller grid resolution so that at least 14 grid cells were placed horizontally and the algorithm could thus place all turbines side by side. For this purpose, the number of wind directions should be increased since a wind farm with an efficiency rate of 100% would stop the algorithm irrespective of whether a better layout with 100% efficiency but with higher expected energy output would exist.

When terrain effects are taken into account, the efficiency rates do not correspond linearly to the expected energy outputs of a wind farm, since an entire wind farm can be exposed to higher wake influences but can achieve higher wind speeds with higher located wind turbines and can achieve thus higher energy yields. This can be seen in the best-found solution regarding efficiency for this method, shown below, which differs from the best-found solution regarding energy output in Figure 47 a).

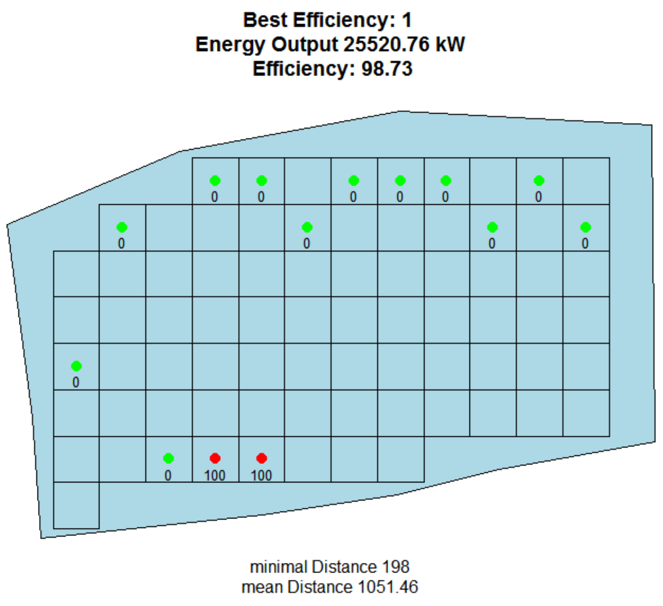


Figure 54: Best-found solution regarding efficiency of wind farm "Tauern" including terrain effects

The energy output of the wind farm with highest efficiency rate of 98.73% is 25520 kW and therefore about 340 kW lower than the wind farm with highest energy output of 25862 kW, achieved with a lower efficiency rate of 97.95%.

5. Discussion & Conclusions

The test runs show that the proposed algorithm works as expected and is able to optimize a given wind farm layout problem. Some of the improvements to previous methods and some of the major drawbacks of the proposed method should be further discussed in this chapter.

It was intended to build a practical tool that could optimize wind farms under real conditions and with real data. Therefore, the shape of the wind farm in the proposed method can have any form and size as compared to many previous studies limited to the optimization of a rectangular area. The algorithm would also function with multi-polygons or with polygons that contain holes although this has not been sufficiently tested. The following example shows however a best-found solution for a multi-polygon area with random input values.

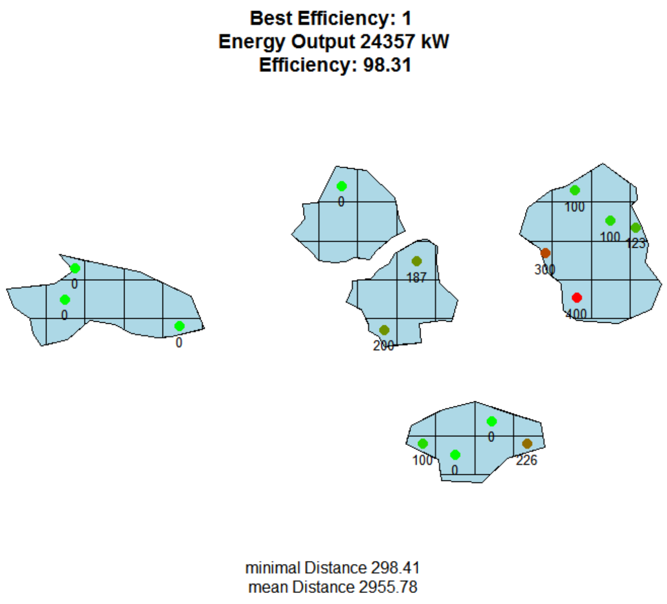


Figure 55: Exemplary best-found solution of multi-polygon area

Several restrictions on the area are therefore possible, which could take into account e.g. nature conservation, property rights, infrastructure, development and their corresponding minimum distances. If the considered wind farm is situated somewhere in Europe then a terrain effect model is available, which includes real geodata. The terrain model uses an elevation raster and a land cover raster to adapt wind speeds to higher elevations, to adjust the air density at a certain altitude, to calculate a surface roughness according to the land cover type and elevation roughness and to adapt the wake decay parameter according to the surface roughness. As described in chapter 4.3.2, the consideration of terrain effects had a significant influence on the expected energy production and layout of a wind farm. The wind speed multiplier method attempts to estimate local wind conditions according to the orography of the terrain, resulting in heterogeneous wind speeds across the surface. Locations at higher altitudes will get higher wind speed estimations but similarly lower air density values. The height thus influences the expected energy output in both directions, although higher wind speeds and hence heights will still be preferred, as their influence on wind power production is exponential. The adapted surface roughness and wake decay value will change the wake diameter and the wind velocity deficit. As the required input values for the adapted values are extracted exactly at the location of a turbine, they are not considered highly accurate. For a more realistic representation, a roughness and a wake decay value should be evaluated over the entire distance of the mutually influencing turbines, as this would also be the distance over which the wake would spread and the wind velocity would be influenced.

The procedure of the proposed method includes many random effects, as the crossover method divides the genetic information, representing available grid cells, and creates possible permutations. It does this, without considering the number of turbines in each crossover part, resulting in wind farms with different total turbine counts. The mutation function also modifies several random genes or grid cells, which may further influence the number of turbines per wind farm. The *trimton* function adjusts the wind farms to the required sum of turbines, with either a random or a probabilistic approach. If the proposed method and, in particular, the calculation of the fitness values were to be adapted so that the efficiency of the wind farm or the cost per unit of energy would be optimized instead of the energy output, the function *trimton* could have been omitted. If the overall profit of a wind farm is maximized with an additional cost function for turbines, the algorithm could optimize the number of turbines and the layout of the wind farm. This could reduce some of the present random effects. Other random effects are inherent in the roulette-wheel selection method, which randomly selects a certain number of individuals from the population, although proportionally to their fitness score. Even with activated elitism, which increases the fitness values of n-elitist individuals by a certain factor, there is still no guarantee that the elitist individuals will be selected. This uncertainty is increased when the number of crossover parts is higher than 2, which increases the number of possible permutations and thus the number of individuals in the populations. If, for example, 3 crossover parts and 100 individuals or 50 parental groups are present in the current population, a total of 400 ($50*2^3 = 400$) permutations would be possible, with a maximum of 300 permutations permitted in the crossover method. 100 excessive permutations must therefore be cleared, which consequently will not be examined in the fitness function and represent a loss of information. After the fitness function evaluated the remaining 300 individuals, the selection function selects a certain amount, depending on the two available selection methods, but at maximum 100 individuals. This would in this case mean a loss of 66.6% of the population. Independent of the chosen selection method, the selection pressure and occurring random effects rise with increasing crossover parts, although it is under the current settings very unlikely that an optimization run would reach crossover parts above 3. The present algorithm will divide the given area at maximum in 3 subareas which are then recombined with the crossover method. If a single wind turbine in one of these subareas would have to be displaced to achieve the global optimum, then probably only a random mutation would have the potential to achieve this. Alternatively, the determination of crossover point locations could be done randomly, rather than at equal intervals, or the number of crossover parts could be increased so that the resulting subareas would include fewer turbines. This in turn would increase the amount of individuals, the selection pressure and therefore the occurring random effects.

One of the major drawbacks of genetic algorithms is the correct configuration and parameterization of all relevant methods and inputs, such as selection, crossover and mutation. Some of the parameters in this method were defined through particular formulas, while some were pure assumptions. Therefore, a supervisory control mechanism, which could assess the complexity of the problem and the development of the populations, would be a desirable complement. In addition to the set of initial individuals, the mechanism should also estimate the time needed for the current problem and automatically set the number of iterations. Likewise, the reduction or increase in the crossover point rate and the selection percentage could be calculated dynamically according to the development of the population's fitness, although the necessary formulas are still missing as for now only constant values are used.

The convergence of the populations to a certain optimum results in several duplicates during evolution. In the development stage of this algorithm, I tried to delete these duplicates in order to increase time efficiency. However, this has proven to be an inefficient design decision, since the development of the fitness values from then on appeared to be too stochastic. Convergence to a certain optimum means that more and more individuals are similar or even identical over time. Genetic code segments with high fitness values are then strongly represented and can therefore reproduce more descendants. The search space will be concentrated around these fittest individuals and will be analyzed more precisely by crossover and mutation. The duplicates are therefore necessary for the genetic algorithm, although they appear to be inefficient, since the proposed method has to calculate duplicates repeatedly. The occurring random effects and the fact, that the algorithm will produce several identical best individuals over time lead to the variable mutation rate method accompanied by a very low default mutation rate of 0.8%. Duplicated fitness evaluations could be prevented by storing layouts with the corresponding output values for further reference.

The time efficiency of the present method is hardly suitable for larger wind farms with approximately 1000 turbines and several wind directions. The algorithm would need days if not weeks for the problem on a regular computer without any guarantee of finding the best possible solution. Even though the best solution to the problem would be found, there may still be an even better solution that was outside the search range of the genetic algorithm, since the algorithm divides the search space into a smaller and finite set of possibilities.

The genetic algorithm could be combined with a subsequent local search method that further optimizes the specific placement of the turbines. Since the turbines of this genetic algorithm can only be placed in the centroid of a grid cell, the resulting best solutions can be considered as good starting solutions for further and more exact optimizations. A subsequent local search method, which can place turbines anywhere in the grid cell and on the considered area, would make the layout arrangement more flexible, which would lead to more irregular layouts and probably higher energy outputs.

Further possible improvement of the proposed method includes a wind speed-dependent power curve for turbines. The thrust coefficient (C_T) of the proposed method is also a constant of 0.88 although it is similarly dependent on the wind speed and should therefore be adapted. To derive a correct estimation of local wind speeds, the algorithm could be further improved so that a raster of wind conditions could be given as input. A raster either with the average wind speed or with the form and scale parameters of the Weibull distribution function per grid cell could be useful. The energy output calculated with the mean wind speed as in the present study results in a lower energy output than when a Weibull distribution of the wind velocities would be used. The energy output with a Weibull distribution would also be more accurate and should therefore be implemented. As Chen, et al. (2013) pointed out, a wind farm with different hub heights can lead to better results and decreased costs per generated power unit. Differing rotor radii could also have a positive effect on the energy results, but I have not found an approach that has implemented this idea. Currently, the implementation of the algorithm is not optimized. To some extent, the method of vectorization has been applied to improve speed, but there is still a lot of potential through program code optimizations – and by parallel computing. A better implementation of the algorithm in combination with better computational hardware, a monitoring and parameter-evaluating mechanism to exploit and explore the search space depending on the development or stagnation of the fitness values could improve the results due to higher population sizes and consequently the evaluation of a larger solution set.

6. List of References

Afanasyeva, S. et al., 2013. *Optimization of wind farm design taking into account uncertainty in input parameters.* Finland: Lappeenranta University of Technology.

Cavcar, M., 2005. In: *The International Standard Atmosphere (ISA).* Eskisehir: Anadolu University, p. 2.

Chen, Y., Li, H. & Song, Q., 2013. *Wind farm layout optimization using genetic algorithm with different hub height wind turbines.* Kingsville: Texas A&M University.

Chowdhury, S., Zhang, J., Messac, A. & Castillo, L., 2011. In: E. Ltd., ed. *Unrestricted wind farm layout optimization (UWFLO): Investigating key factors influencing the maximum power generation.* s.l.:s.n., p. 19.

Diaz-Gomez, P. A. & Hougen, D. F., 2007. *Initial Population for Genetic Algorithms: A Metric Approach.* Norman, Oklahoma, USA: University of Oklahoma.

EEA, 2016. *Corine Land Cover 2006 raster data.* [Online] Available at: http://www.eea.europa.eu/data-and-maps/data/clc-2006-raster-3
[Accessed 12 2016].

Global Wind Energy Council, 2015. *Annual Market Update.* Brussel: Global Wind Energy Council.

Global Wind Energy Council, 2016. *Global Wind Statistics 2015.* [Online] Available at: http://www.gwec.net/wp-content/uploads/vip/GWEC-PRstats-2015_LR.pdf
[Accessed 20 12 2016].

Grady, S. & Hussaini, M., 2005. In: *Placement of wind turbines using genetic algorithms.* s.l.:s.n.

Guirguis, D., Romero, D. A. & Amon, C. H., 2016. In: U. o. Toronto, ed. *Toward efficient optimization of wind farm layouts: Utilizing exact gradient information.* Canada: s.n.

Hakenesch, P., 2016. In: *Aerodynamik des Flugzeugs.* s.l.:s.n., pp. 7-10.

Hidasi-Neto, J., 2014. *GridFilter Function: Intersect Shape with a Grid and Exclude Cells Based on Area Coverage.* [Online] Available at: http://rfunctions.blogspot.co.at/2014/12/gridfilter-intersect-grid-with-shape.html
[Accessed 2016].

Hirsch, P., 2012. In: *Naturanaloge Optimierungsverfahren und Modellierung.* Wien: Universität für Bodenkultur, pp. 74 -76.

International Renewable Energy Agency, 2012. Wind Power. In: *Renewable Energy Technologies: Cost Analysis Series.* Bonn: IRENA, pp. 4 - 6.

Ituarte-Villarreal, C. M. & Espiritu, J. F., 2011. In: T. U. o. T. a. E. Paso, ed. *Wind turbine placement in a wind farm using a viral based optimization.* El Paso: s.n.

Kalmikov, A. & Dykes, K., 2011. In: *Wind Power Fundamentals.* Massachusetts: MIT Wind Energy Group, pp. 12 - 13.

Katic, I., Hojstrup, J. & Jensen, N. O., 1987. In: *A Simple Model for Cluster Efficiency.* Rome: A. Raguzzi, pp. 407 - 408.

Kruse, R. & Held, P., 2013. Evolutionäre Algorithmen - Kodierung, Fitness, Selektion. In: I. f. W. u. Sprachverarbeitung, ed. Magdeburg: Otto-von-Guericke-Universität Magdeburg.

Kusiak & Song, 2009. In: *Design of wind farm layout for maximum wind energy capture.* Iowa City: University of Iowa, p. 687.

Liersch, D.-I. J., 2012. In: K. W. E. GmbH, ed. *Technik von Windkraftanlagen.* Berlin: s.n., p. 17.

Mehmeti, Q. et al., 2014. In: T. U. Dortmund, ed. *Genetische Algorithmen: Wirtschaftsmathematisches Projekt zur Numerik im WS 2013/14.* Dortmund: s.n.

Mosetti, G., Poloni, C. & Diviacco, B., 1994. In: J. o. W. E. a. I. Aredoynamics, ed. *Optimization of wind turbine positioning in large windfarms by means of a genetic algorithm.* s.l.:s.n.

Pelikan, M., Goldberg, D. E. & Cantu-Paz, E., 2000. *Bayesian Optimization Algorithm, Population Sizing, and Time to Convergence.* Las Vegas: U.S. Department of Energy.

Renkema, D. J., 2007. In: *Validation of wind turbine wake models.* Delft: Delft University of Technology, pp. 6 - 7.

Saavedra-Moreno, B. et al., 2011. In: *Seeding evolutionary algorithms with heuristics for optimal wind turbines.* Madrid: Universidad de Alcalá, pp. 2840-2841.

Samorani, M., 2013. In: *The Wind Farm Layout Optimization Problem.* Berlin: Springer, pp. 23 - 27.

Schmelmer, R., 2013. In: H. f. A. Wissenschaften, ed. *Wirtschaftlichkeitsanalyse einer vertikalen Kleinwindkraftanlage.* Landshut: s.n., p. 20.

Scholl, P., 2002. *Genetische Algorithmen – Erweiterungen und Analyse.* s.l.:s.n.

Shakoor, R., Hassan, M. Y., Raheem, A. & Rasheed, N., 2015. In: I. J. o. S. a. Technology, ed. *The Modelling of Wind Farm Layout Optimization for the Reduction of Wake Losses.* Adyar: s.n.

Shakoor, R., Hassan, M. Y., Raheem, A. & Wu, Y.-K., 2015. In: E. Ltd., ed. *Wake effect modeling: A review of wind farm layout optimization using Jensen's model.* s.l.:s.n., p. 1058.

Solle, C., 2011. In: C. z. Kiel, ed. *Organisationsformen der Windenergieerzeugung durch Landwirte – Eine betriebswirtschaftliche Analyse.* Kiel: s.n., p. 17.

Wang, L., Singh, C. & Kusiak, A., 2010. Application of Computational Intelligence. In: *Wind Power Systems.* Berlin Heidelberg: Springer, pp. 64-75.

Wendy, W., 2015. In: T. Wien, ed. *Genetic Algorithms: A Tutorial.* Wien: s.n., pp. 1 - 4.

Williams, G. M., 2014. In: *Wind Farm Layout Optimization Problem by Modified Genetic Algorithm.* Oklahoma : Oklahoma State University, p. 2.

Williams, G. M., 2014. In: *Wind Farm Layout Optimization Problem by Modified Genetic Algorithm.* Oklahoma: Oklahoma State University, p. 6.

Yang, Zhang, Sun & Zhang, 2015. *Optimal Wind Turbines Micrositing in Onshore Wind Farms Using Fuzzy Genetic Algorithm.* Shenyang: Hindawi Publishing Corporation.

Zahoransky, 2010. *Energietechnik: Systeme zur Energieumwandlung. Kompaktwissen für Studium und Beruf.* Wiesbaden: Vieweg und Teubner.

Zhang, P. Y., 2013. In: *Topics in Wind Farm Layout Optimization: Analytical Wake Models, Noise Propagation and Energy Production.* Toronto: University of Toronto, pp. 5-6.

7. List of Figures

8. List of R-Codes

9. List of Tables

Appendix

```
############### R - CODE ###################################
##########################################################
############### Required Inputs #############
ProjLAEA <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
              +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"

## To include terrain effects, the paths for the Corine Land Cover raster
## and legend must be changed on the following pages: 131, 137 and 139.

################ Input a Polygon
## Use your own Shapefile
# dns = "C:/.../Directory_of_Shapefile"
# AreaConsidered <- readOGR(dsn=dns, layer="NameOfShapeFile"); plot(AreaConsidered)
# AreaConsidered <-  spTransform(AreaConsidered, CRSobj = crs(ProjLAEA));

## Or create a random Polygon
Pol <- Polygon(rbind(c(0, 0), c(0, 2000), c(2000, 2000), c(2000, 0)))
Pols <- Polygons(list(Pol),1)
AreaConsidered <- SpatialPolygons(list(Pols)); rm(Pol,Pols)
proj4string(AreaConsidered) <- CRS(ProjLAEA)
plot(AreaConsidered,axes=T)

################ Initialize a random Wind data set
data.in <- structure(list(ws =  c(12,12,12,12),
                        wd = c(0,0,0,0)), .Names = c ("ws", "wd"),
                   row.names = c(NA, 4L), class = "data.frame")

##########################################################
############### Genetic Algorithm functions #############

############### Calculate the euclidian distance
euc.dist          <- function(x,y,...) {  round(sqrt(sum((x - y) ^ 2)),4) }
############### Calculate distances of a triangle
PointToLine2      <- function(x,y,k, ...) {
  if (x[1]-y[1] == 0 | x[2]-y[2]==0) {
    distc <- euc.dist(x,y)
    yellow <- rbind(c(y,x,x,distc,distc,0));
    colnames(yellow) <- c("Ax","Ay","Bx","By","Cx","Cy","Laenge_C",
                          "Laenge_B","Laenge_A");
    invisible(yellow);
  }
  if (is.matrix(y) == FALSE) {y <- as.matrix(rbind(y))};
  if (is.matrix(x) == FALSE) {x<- as.matrix(rbind(x))};
  if (k == "N"|k =="S") {C1 <- as.matrix(cbind(y[1,1],x[1,2]));}
  c <- euc.dist(x,y);
  a <- euc.dist(x,C1);
  b <- euc.dist(C1,y);
  dist1 <- c(c,b,a)
  blue <- rbind(c(y,x,C1,dist1));
  colnames(blue) <- c("Ax","Ay","Bx","By","Cx","Cy","Laenge_C",
                    "Laenge_B","Laenge_A");
  invisible (blue)
}
############### Calculate angles of a triangle
WinkelCalc        <- function(Aa,Bb,Cc, ...) {
  if (length(Aa)!=2|class(Aa)!="numeric"  |
       length(Bb)!=2|class(Bb)!="numeric" |
       length(Cc)!=2|class(Cc)!="numeric"){
           warning("Input is not numeric or doesnt contain two values")
  }
  AB <- Bb-Aa;  AC <- Cc-Aa;
  BA <- Aa-Bb;  BC <- Cc-Bb;
  CA <- Aa-Cc;  CB <- Bb-Cc;
  alpha <- acos(sum(AB*AC) / (sqrt(sum(AB * AB)) * sqrt(sum(AC * AC))))*(180/pi);
  betha <- acos(sum(BA*BC)/(sqrt(sum(BA*BA))*sqrt(sum(BC*BC))))*(180/pi);
  gamma <- acos(sum(CA*CB) / (sqrt(sum(CA * CA)) * sqrt(sum(CB * CB))))*(180/pi)
  if (trunc(alpha+betha+gamma) >= 179 | ceiling(alpha+betha+gamma) <= 180) {
    winkel <- rbind(alpha,betha,gamma)
    invisible(winkel)
  }
}
############### Calculate all distances and angles for one turbine
VekWinkelCalc     <- function(t,o,p, wkl, distanz, polYgon,...) {
  p = "N"
  datalist = list(); datalist1 = list();
  WKA_akt <- c(x = t[o,1], y = t[o,2]);
  xynew1 <- subset.matrix(x = t, subset = (t[o,1] != t[,1])&(t[o,2] < t[,2]));
  xyvorne1 <- subset.matrix(x = t, subset = (t[o,1] == t[,1])&(t[o,2] < t[,2]));
  len2 = length(xynew1[,1])
  len1 = length(xyvorne1[,1])
  if ((len2 + len1 != 0)) {
    if (len2!=0) {
      for (i in 1:len2){
        P2LFu <- PointToLine2(WKA_akt, xynew1[i,], p);
        winkel <- t(WinkelCalc(xynew1[i,],WKA_akt, P2LFu[5:6]));
        data <- t(as.matrix(c(as.numeric(P2LFu), as.numeric(winkel))));
        colN <- t(as.matrix(c(colnames(P2LFu),colnames(winkel))));
        datalist[[i]] <- data;
        datalun   <-   data.frame(matrix(unlist(datalist),   nrow=(length(datalist)),
                    byrow = TRUE));
        colnames(datalun) <- colN;
        datalun <- subset.data.frame(datalun,subset = datalun$alpha<wkl &
                                        datalun$Laenge_B<distanz);
      }
      if (nrow(datalun)==0){remove(datalun)}
    };
    if (len1!=0) {
      for (k in 1:len1){
        P2LFu1 <- PointToLine2(WKA_akt, xyvorne1[k,], p);
        P2LFu2 <- cbind(P2LFu1, alpha=0,betha=0,gamma=0);
        colN1 <- t(as.matrix(c(colnames(P2LFu2))));
        datalist1[[k]] <- P2LFu2
        datalun1   <-data.frame(matrix(unlist(datalist1),   nrow=(length(datalist1)),
                          byrow = TRUE));
        colnames(datalun1) <- colN1;
        datalun1 <- subset.data.frame(datalun1,subset = datalun1$Laenge_B<distanz)
      }
      if (nrow(datalun1) == 0) {remove(datalun1)}
    }
    if (exists("datalun") && exists("datalun1")) {
      DataLun3 <- rbind(datalun, datalun1);
    }
    if (exists("datalun") && !exists("datalun1")) {
      DataLun3 <- datalun;
    }
    if (exists("datalun1") && !exists("datalun")) {
      DataLun3 <- datalun1;
    }
    if (!exists("datalun") && !exists("datalun1")) {
      dfm3 <- data.frame(matrix(data = c(0,0,t[o,1],t[o,2],
                                         (rep(0,8))),nrow = 1, ncol = 12));
      colnames(dfm3) <- c("Ax","Ay","Bx","By","Cx","Cy","Laenge_C",
                          "Laenge_B","Laenge_A","alpha","betha","gamma");
      DataLun3 <- dfm3;
    }
  } else  {
    dfm4 <- data.frame(matrix(data = c(0,0,t[o,1],t[o,2],(rep(0,8))),
                        nrow = 1, ncol = 12));
    colnames(dfm4) <- c("Ax","Ay","Bx","By","Cx","Cy","Laenge_C",
                        "Laenge_B","Laenge_A","alpha","betha","gamma");
    DataLun3 <- dfm4;
  }
  DataLun3
  return(DataLun3);
}
############### Find all the potential influencing turbines
InfluPoints       <- function(t, wnkl, dist,polYgon,dirct,...){
  pointList <- list();
  for (i in 1:(length(t[,1]))) {
    ee11 <- VekWinkelCalc(t, i, dirct, wnkl, dist, polYgon);
    if (nrow(ee11) != 0){
      ee11[,13] <- as.character(dirct); colnames(ee11)[13] <- "Windrichtung";
      ee11[14] <- i;  colnames(ee11)[14] <- "Punkt_id"
      pointList[[i]] <- ee11;
    } else {
      abd <- data.frame(matrix(data = ee11[13,],nrow = 1, ncol = 12));
      abd[13] <- dirct[i]; colnames(abd) <- c(colnames(ee11), "Windrichtung")
      abd[14] <- i; colnames(abd)[14] <- "Punkt_id";
      pointList[[i]] <- abd;
    }
  }
  invisible(pointList)
}
############### Calculate air density, pressure and temperature according to height
BaroHoehe         <- function(data, height, po=101325, ro=1.225) {
  if ((ncol(data))==1) {
    ph = po * exp(-data * 0.0001252);  names(ph) <- "ph"
    rh = ro * exp(-data * 0.0001252); names(rh) <- "rh"
    Th <- 288.15 - ((6.5 * data)/1000); names(Th) <- "tempK"
  } else {
    ph = po * exp(-data[,height] * 0.0001252);
    rh = ro * exp(-data[,height] * 0.0001252);
    Th <- 288.15 - ((6.5 * data[,height])/1000);
  }
  if (class(data)!= "data.frame") {
    data <- as.data.frame(data)
  }
  Celsius = Th - 273.15; names(Celsius) <- "tempC"
  data$ph <- ph
  data$rh <- rh
  data$tempK <- Th
  data$tempC <- Celsius
  colnames(data) <- names(data)
  return(data)
}
############### Plot a Windrose
plot.windrose     <- function(data,spd,dir,spdres = 2,dirres = 30,spdmin = 1,
                          spdmax = 30, spdseq = NULL,palette = "YlGnBu",
                          countmax = NA,debug = 0){
  require(ggplot2);   require(RColorBrewer)
  if (is.numeric(spd) & is.numeric(dir)){
    data <- data.frame(spd = spd,
                       dir = dir)
    spd = "spd"
    dir = "dir"
  } else if (exists("data")){
  }
  n.in <- NROW(data)
  dnu <- (is.na(data[[spd]]) | is.na(data[[dir]]))
  data[[spd]][dnu] <- NA
  data[[dir]][dnu] <- NA
  if (missing(spdseq)){
    spdseq <- seq(spdmin,spdmax,spdres)
  } else {
    if (debug >0){
      cat("Using custom speed bins \n")
    }
  }
  n.spd.seq <- length(spdseq)
  n.colors.in.range <- n.spd.seq - 1
  spd.colors <- colorRampPalette(brewer.pal(min(max(3,n.colors.in.range),
                                              min(9,n.colors.in.range)),
                                            palette))(n.colors.in.range)
    if (max(data[[spd]],na.rm = TRUE) > spdmax){
    spd.breaks <- c(spdseq,max(data[[spd]],na.rm = TRUE))
    spd.labels <- c(paste(c(spdseq[1:n.spd.seq-1]),
                          '-',c(spdseq[2:n.spd.seq])),
                    paste(spdmax,"-",max(data[[spd]],na.rm = TRUE)))
    spd.colors <- c(spd.colors, "grey50")
  } else{
    spd.breaks <- spdseq
    spd.labels <- paste(c(spdseq[1:n.spd.seq-1]),
                        '-',c(spdseq[2:n.spd.seq]))
  }
  data$spd.binned <- cut(x = data[[spd]],
                         breaks = spd.breaks,
                         labels = spd.labels,
                         ordered_result = TRUE)
  dir.breaks <- c(-dirres/2,seq(dirres/2, 360-dirres/2, by = dirres),
                  360+dirres/2)
  dir.labels <- c(paste(360-dirres/2,"-",dirres/2),
                  paste(seq(dirres/2, 360-3*dirres/2, by = dirres),
                        "-",seq(3*dirres/2, 360-dirres/2, by = dirres)),
                  paste(360-dirres/2,"-",dirres/2))
  dir.binned <- cut(data[[dir]],breaks = dir.breaks,ordered_result = TRUE)
  levels(dir.binned) <- dir.labels
  data$dir.binned <- dir.binned
  p.windrose <- ggplot(data = data, aes(x = dir.binned, fill = spd.binned)) +
    geom_bar() + scale_x_discrete(drop = FALSE, labels = waiver()) +
    coord_polar(start = -((dirres/2)/360) * 2*pi) +
    scale_fill_manual(name = "Wind Speed (m/s)", values = spd.colors,
                      drop = FALSE) + theme(axis.title.x = element_blank())
  if (!is.na(countmax)){ p.windrose <- p.windrose + ylim(c(0,countmax))}
  print(p.windrose)
  return(p.windrose)
}
############### Calculate the expected energy output of a windfarm
calculateEn       <- function(sel, referenceHeight, RotorHeight,SurfaceRoughness,

windraster,wnkl,distanz,polygon1,resol,RotorR,dirSpeed,srtm_crop,
                              topograp,cclRaster){
  require(data.table);  require(maptools);     library(calibrate)
  PlotCalc="FALSE"
  sel1 = sel[,2:3];
  cT <- 0.88;
```

```
air_rh <- 1.225;
k = 0.075;
windpo <- raster::extract(x= windraster, y = as.matrix((sel1)),
                          buffer=resol*1, small=T,fun= mean,na.rm=T);
if (topograp == "TRUE") {
  orogr1 <- calc(srtm_crop, function(x) {x/(cellStats(srtm_crop,mean,na.rm=T))})
  orogrnum <- raster::extract(x= orogr1, y = as.matrix((sel1)),
                              buffer=resol*2, small=T,fun= mean,na.rm=T);
  windpo <- windpo * orogrnum
  heightWind <- raster::extract(x= srtm_crop, y = as.matrix((sel1)), small=T,fun=
                  max,na.rm=T);
  if (PlotCalc=="TRUE"){
    par(mfrow=c(1,1))
    plot(srtm_crop, main="SRTM Elevation Data");
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round(heightWind,0),cex=0.8);
    plot(polygon1,add=T)
    plot(orogr1, main="Wind Speed Multipliers");
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round(windpo,3),cex=0.8);
    plot(polygon1,add=T)
  }
  HeighttoBaro <- matrix(heightWind);
  colnames(HeighttoBaro) <- "HeighttoBaro"
  air_dt <- BaroHoehe(matrix(HeighttoBaro),HeighttoBaro)
  air_rh <- as.numeric(air_dt$rh);
  if (PlotCalc=="TRUE"){
    par(mfrow=c(1,1))
    plot(srtm_crop, main="Normal Air Density",col=topo.colors(10));
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = rep(1.225,nrow(sel1)),cex=0.8);
    plot(polygon1,add=T)
    plot(srtm_crop, main="Corrected Air Density",col=topo.colors(10));
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round(air_dt$rh,2),cex=0.8);plot(polygon1,add=T)
  }
  SurfaceRoughness0 <- raster::extract(x= cclRaster, y = as.matrix((sel1)),
                                       buffer=resol*2, small=T,fun= mean,
                                        na.rm=T);
  SurfaceRoughness1 <- raster::extract(x=terrain(srtm_crop,"roughness"),
                                       y = as.matrix((sel1)),
                                      buffer=resol*2, small=T,fun=
                                       mean,na.rm=T);
  SurfaceRoughness <-SurfaceRoughness*
                     (1+(SurfaceRoughness1/max(res(srtm_crop))));
  elrouind <- terrain(srtm_crop,"roughness")
  elrouindn <- resample(elrouind,cclRaster,method="ngb")
  modSurf <- overlay(x = cclRaster,y = elrouindn,
                     fun=function(x,y){return(x*(1+y/max(res(srtm_crop))))})
  if (PlotCalc=="TRUE"){
    par(mfrow=c(1,1)); cexa=0.9
    plot(cclRaster, main="Corine Land Cover Roughness");
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round(SurfaceRoughness0,2),cex=cexa);
    plot(polygon1,add=T)
    plot(x=terrain(srtm_crop,"roughness",neighbors = 4),
    main="Elevation Roughness Indicator");
    points(sel1$X,sel1$Y,pch=20); textxy(sel1$X,sel1$Y,
    labs = round((SurfaceRoughness1),2),cex=cexa); plot(polygon1,add=T)
    plot(modSurf, main="Modified Surface Roughness");
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round((SurfaceRoughness),2),cex=cexa);
    plot(polygon1,add=T)
  }
  k = 0.5/(log(RotorHeight/SurfaceRoughness))
  if (PlotCalc=="TRUE"){
    par(mfrow=c(1,1)); cexa=0.9
    plot(x=terrain(srtm_crop,"roughness",neighbors = 4),
         main="Adapted Wake Decay Constant - K");
    points(sel1$X,sel1$Y,pch=20);
    textxy(sel1$X,sel1$Y,labs = round((k),3),cex=cexa);
    plot(polygon1,add=T)
  }
}
alllist <- vector("list",nrow(dirSpeed))
for (index in 1:nrow(dirSpeed)) {
  xyBgldMa <- as.matrix((sel1));
  pointWind <- windpo * dirSpeed$ws[index]
  pointWind <- pointWind*((RotorHeight/referenceHeight)^SurfaceRoughness);
  pointWind[is.na(pointWind)] <- 0;
  angle <- -dirSpeed$wd[index];
  if (PlotCalc == "TRUE"){
    par(mfrow=c(1,2))
    plot(polygon1, main="Shape at angle 0");
    points(xyBgldMa[,1],xyBgldMa[,2],pch=20)
    textxy(xyBgldMa[,1],xyBgldMa[,2], labs = dimnames(xyBgldMa)[[1]],cex=0.8)
    Polygon3 = elide(polygon1, rotate=angle, center=apply(bbox(polygon1), 1,
    mean));
    plot(Polygon3, main=c("Shape at angle:", (-1*angle)))
    mtext(paste("Direction: ", index, "\nfrom total: ", nrow(dirSpeed)),
    side = 1)
  }
  xyBgldMa <- SpatialPoints(coordinates(xyBgldMa))
  xyBgldMa <- elide(xyBgldMa, rotate=angle, center=apply(bbox(polygon1), 1,
  mean));
  xyBgldMa <- coordinates(xyBgldMa)
  if (PlotCalc == "TRUE"){
    points(xyBgldMa, col="red",pch=20)
  }
  DatFram <- data.table(cbind(pointWind,xyBgldMa));
  colnames(DatFram)=c("Windmittel","X","Y");
  BgleInf <- InfluPoints(xyBgldMa, wnkl, distanz, Polygon, dirct = angle)
  windlist = vector("list",length(sel1[,1]))
  dfAll <- do.call("rbind",BgleInf) ;
  maxpo <- max(dfAll$Punkt_id)
  for (i in 1:maxpo){
    windlist[[i]] <- dplyr::filter(dplyr::select(dplyr::tbl_df(dfAll),
                                   Punkt_id,Ax,Ay,Bx,By,Laenge_B,
                                   Laenge_A,alpha,Windrichtung),
                                   Punkt_id==i)
    windlist[[i]]$Windmean <- DatFram[[1]][i];
  }
  windlist <- do.call("rbind", windlist);
  windlist$RotorR <- as.numeric(RotorR);
  lnro = nrow(windlist); windlist$WakeR <- 0; windlist$Rotorflaeche <- 0
  for (i in 1:lnro){
    RotD <- as.numeric(windlist[i,]$RotorR)
    if (windlist[i,]$Laenge_B != 0) {
      if (topograp=="TRUE"){
        windlist[i,]$WakeR = (((RotD * 2) + (2*k[windlist[i,]$Punkt_id]*
                              (as.numeric(windlist[i,]$Laenge_B))))/2)[1]
      } else {
        windlist[i,]$WakeR = (((RotD * 2) +
                         (2*k*(as.numeric(windlist[i,]$Laenge_B))))/2)[1]
        }
      } else {
        windlist[i,]$WakeR = 0
      }
      windlist[i,]$Rotorflaeche = (RotD^2) *pi
    };
    windlist$A_ov <- 0; windlist$AbschatInProz <- 0
    for (o in 1:lnro){
      Rotorf <- as.numeric(windlist[o,]$RotorR)
      leA <- windlist[o,]$Laenge_A
      wakr <- windlist[o,]$WakeR;
      if (windlist[o,]$Laenge_B == 0) {
        windlist[o,]$A_ov <- 0;
      } else {
        if ((wakr - Rotorf) >= leA && leA >= 0) {
          windlist[o,]$A_ov <- as.numeric(windlist[o,]$RotorR^2)*pi;
        }
        if (round((wakr + Rotorf),2) <= round(leA,2)) {
          windlist[o,]$A_ov <- 0
        }
        if ((wakr - Rotorf) <= leA && leA <= (wakr+Rotorf))  {
          windlist[o,]$A_ov <-  (Rotorf^2 * round(acos((Rotorf^2 - wakr^2 + leA^2) /
                                 (2*leA * Rotorf)),4)) +
                                (wakr^2  *  round(acos((wakr^2  -  Rotorf^2  +  leA^2)  /
                                 (2*leA * wakr)),4)) -
                                ((1/2)*sqrt( round((Rotorf + wakr + leA),6) *
                                 round((-Rotorf + wakr + leA ),6) *
                                round((Rotorf  -  wakr  +  leA),6)  *  round((Rotorf  +
                                 wakr - leA),6) ))
        }
      }
 if (windlist[o,]$A_ov != 0) {
        windlist[o,]$AbschatInProz <- round(((as.numeric(windlist[o,]$A_ov)/
                                              windlist[o,]$Rotorflaeche)*100), 2);
      } else {
        windlist[o,]$AbschatInProz <- 0;
      }
    }
    windlist$V_red <- 0
    for (p in 1:lnro) {
      RotrR <- windlist[p,]$RotorR
      a <- 1- sqrt(1-cT)
      s <- (windlist[p,]$Laenge_B/RotrR);
      if (topograp=="TRUE"){
        b <- (1 + (k[windlist[p,]$Punkt_id]*s))^2;
      } else {
        b <- (1 + (k*s))^2;
      }
      aov <- (windlist[p,]$A_ov / windlist[p,]$Rotorflaeche);
      windlist[p,]$V_red <- (aov *(a / b))
      ve <- windlist[p,]$Windmean * windlist[p,]$V_red;
      windlist[p,]$V_red <- ve
    }
    maPi = max(windlist$Punkt_id)
    windlist$V_i <- 0; windlist$TotAbschProz <- 0; windlist$V_New <- 0;
    windlist$Rect_ID <- 0
    for (z in 1:maPi) {
      windlist[windlist$Punkt_id==z,]$V_i <-
                                  sqrt(sum(windlist[windlist$Punkt_id==z,]$V_red^2))
      windlist[windlist$Punkt_id==z,]$TotAbschProz <-
                                   sum(windlist[windlist$Punkt_id==z,]$AbschatInProz)
      windlist[windlist$Punkt_id==z,]$V_New <-
```

```
                              windlist[windlist$Punkt_id==z,]$Windmean -
                              windlist[windlist$Punkt_id==z,]$V_i
      windlist[windlist$Punkt_id==z,]$Rect_ID <-  sel[z,1]
    }
    windlist2 <- dplyr::select(windlist,-alpha,-AbschatInProz,-Rotorflaeche,-
                             V_red,-V_i)
    windlist1 <- split(windlist2, duplicated(windlist2$Punkt_id))$'FALSE'
    EneOutRed <- sum(0.593 * (1/2) * air_rh * (windlist1$V_New ^ 3) *
                    ((as.numeric(windlist1$RotorR)^2)*pi),na.rm=T)/1000;
    EneOutFul <- sum(0.593 * (1/2) * air_rh * (windlist1$Windmean^3)*
                    ((as.numeric(windlist1$RotorR)^2)*pi),na.rm=T)/1000;
    Effic <- (EneOutRed*100)/EneOutFul;
    windlist2$Energy_Output_Red <- EneOutRed;
    windlist2$Energy_Output_Voll <- EneOutFul;
    windlist2$Parkwirkungsgrad <- Effic;
    windlist2$Windrichtung <- as.numeric(windlist2$Windrichtung) * (-1)
    alllist[[index]] <- windlist2
  }
  invisible(alllist)
}
############### Calculate a Grid
GridFilter        <- function(shape, resol = 500, prop = 1,plotGrid="FALSE"){
  require(rgeos);
  if (prop < 0.01){prop = 0.01}
  grid <- raster::raster(extent(shape))
  res(grid) <- c(resol,resol)
  proj4string(grid)<-proj4string(shape)
  gridpolygon <- rasterToPolygons(grid)
  drylandproj<- spTransform(shape, CRS("+proj=laea"))
  gridpolproj<-spTransform(gridpolygon, CRS("+proj=laea"))
  gridpolproj$layer <- c(1:length(gridpolproj$layer))
  areagrid <- rgeos::gArea(gridpolproj, byid=T)
  dry.grid <- rgeos::intersect(drylandproj, gridpolproj)
  areadrygrid <- rgeos::gArea(dry.grid, byid=T)
  info <- cbind(dry.grid$layer, areagrid[dry.grid$layer], areadrygrid)
  dry.grid$layer<-info[,3]/info[,2]
  dry.grid <- spTransform(dry.grid, CRS(proj4string(shape)))
  if(!any(dry.grid$layer >= prop)) {
    print("Maybe the resolution is too high")
    stop("Try a smaller one.")
  }
  dry.grid.filtered <<- dry.grid[dry.grid$layer >= prop,];
  areaquares <- round(sum(sapply(dry.grid.filtered@polygons,
                              function(x) sapply(x@Polygons, function(y)
                                                    y@area)))/1000000,3)
  par(mar=c(5,5,5,4))
  if (plotGrid == "TRUE"){
    plot(shape, col="orange",
          main = paste("Resolution:", resol, "m and prop: ",prop,
                       "\n Total Area:", round(sum(areadrygrid)/1000000,3),
                       "km² \n Number Grids:", length(dry.grid.filtered),
                       "\n Sum Grid size:", areaquares, "km²"))
    plot(dry.grid.filtered, col="lightgreen",add=TRUE)
  }
  x <- lapply(dry.grid.filtered@polygons, function(x) sapply(x@Polygons,
       function(y) y@coords[,1]))
  y <- lapply(dry.grid.filtered@polygons, function(x) sapply(x@Polygons,
       function(y) y@coords[,2]))
  rect_Nu <- gCentroid(dry.grid.filtered,byid = T);  plot(rect_Nu,add=T)
  centpo <- coordinates(rect_Nu);     centpo <- as.data.frame(centpo)
  centpo$ID <- 1:nrow(centpo);     names(centpo) <- c("X","Y","ID")
  centpo <- dplyr::select(centpo, ID,X,Y)
  points(centpo$X,centpo$Y, col="blue", pch=20)
  text(centpo$X,centpo$Y,labels=centpo$ID, pos=2)
  invisible(centpo)
}
###############  Start an Initial Population
StartGA           <- function(Grid, n,nStart=100) {
  if (length(Grid$ID) <= n) {
    cat(paste("Amount Grid-cells: ", length(Grid$ID),"\n Amount of turbines: ", n))
    stop("The amount of grid-cells is smaller or equal to the number of turbines
          requested. Resolution / number of turbines or Rotorradius.")
  }
  if (length(Grid$ID) < (2*n)) {
    cat(paste("Amount Grid-cells: ", length(Grid$ID),"\n Amount of turbines: ", n))
    stop("The amount of grid-cells should at least be double the size of turbines
          requested. Resolution / number of turbines or Rotorradius.")
    cat("Press [enter] to continue")
    line <- readline()
  }
  subsetSel = list(); ids=list();
  for (i in 1:nStart){
    Grid$bin <- 0
    ids[i][[1]] <- sort(sample(x = Grid$ID, size = n, replace = F))
    Grid[Grid$ID %in% ids[i][[1]], ]$bin = 1
    subsetSel[i][[1]] <- Grid[Grid$bin == 1,]
  }
  return(subsetSel)
}
############### Calculate a fitness value for a population
fitness           <- function(selection, referenceHeight,
                     RotorHeight,SurfaceRoughness,Polygon,resol1,rot,dirspeed,
                     srtm_crop, topograp,cclRaster,...){
  dirspeed$wd <- round(dirspeed$wd,0)
  dirspeed$wd <-  round(dirspeed$wd/100,1)*100;
  if (any(names(dirspeed) == "probab") == FALSE) {
    dirspeed$probab <- 100/nrow(dirspeed)
  }
  dirspeed$probab <- round(dirspeed$probab,0)
  if (sum(dirspeed$probab) != 100) {
    dirspeed$probab <- dirspeed$probab*(100/sum(dirspeed$probab))
  }
  if   (any(duplicated(dirspeed$wd)==TRUE)) {
    for (i in 1:nrow(dirspeed[duplicated(dirspeed$wd)==F,])){
      temp <- dirspeed[dirspeed$wd ==  dirspeed[duplicated(
        dirspeed$wd)==F,][i,'wd'],];
      temp$ws <-with(temp, sum(ws * (probab/sum(probab))));
      temp$probab <- with(temp, sum(probab * (probab/sum(probab))));
      dirspeed[dirspeed$wd ==  dirspeed[duplicated(
        dirspeed$wd)==F,][i,'wd'],]$ws <- round(temp$ws,2)[1]
      dirspeed[dirspeed$wd ==  dirspeed[duplicated(
        dirspeed$wd)==F,][i,'wd'],]$probab <- round(temp$probab,2)[1]
    }
  }
  dirspeed <- dirspeed[!duplicated(dirspeed$wd)==TRUE,];
  dirspeed <- dirspeed[with(dirspeed, order(wd)), ]
  if (sum(dirspeed$probab) != 100) {
    dirspeed$probab <- dirspeed$probab*(100/sum(dirspeed$probab))
  }
  probabDir <- dirspeed$probab;
  pp <- sum(probabDir)/100;   probabDir <- probabDir/pp; sum(probabDir)
  windraster <-rasterize(Polygon, raster(extent(Polygon), ncol=180,
                          nrow=180),field=1)
  e = vector("list",length(selection));euniqu=vector("list",length(selection)) ;
  for (i in 1:length(selection)){
    e[[i]] <- calculateEn(selection[[i]], referenceHeight,
                     RotorHeight,SurfaceRoughness,
                            windraster = windraster, wnkl = 20, distanz=100000,
                             polygon1 = Polygon, resol=resol1, RotorR = rot,
                             dirSpeed = dirspeed, srtm_crop,topograp,cclRaster)
    ee  <- lapply(e[[i]], function(x){split(x, duplicated(x$Punkt_id))$'FALSE'})
    ee  <- lapply(ee, function(x){dplyr::select(x,-Ax,-Ay,-Laenge_B,-Laenge_A,-
                             Windmean,-WakeR,-A_ov, -Punkt_id)})
    enOut <- lapply(ee, function(x){ x[1,c(3,8,10)]});
    enOut <- do.call("rbind", enOut)
    enOut$probabDir <- probabDir
    enOut$Eneralldire <- enOut$Energy_Output_Red * (enOut$probabDir/100);
    enOut$EnergyOverall <- sum(enOut$Eneralldire);
    enOut$Efficalldire <- sum(enOut$Parkwirkungsgrad * (enOut$probabDir/100))
    AbschGesamt <- lapply(ee, function(x){ data.frame(x$TotAbschProz)});
    AbschGesamt <- do.call("cbind",AbschGesamt)
    AbschGesamt$RowSum <- rowSums(AbschGesamt);
    AbschGesamt <- AbschGesamt$RowSum
    xundyOrig <- selection[[i]][,2:3]; xundyOrig
    xundyOrig$EfficAllDir <- enOut$Efficalldire[1];
    xundyOrig$EnergyOverall <- enOut$EnergyOverall[1];
    xundyOrig$AbschGesamt <- AbschGesamt
    xundyOrig$Run <- i
    dt <-  ee[[1]]; dt <- dplyr::select(dt,RotorR, Rect_ID);dt;
    dt <- cbind(xundyOrig,dt)
    euniqu[[i]] <- dt
  }
  maxparkeff <- do.call("rbind", (lapply(euniqu, function(x) { x <-
                             dplyr::select(x[1,],EnergyOverall)})))
  maxparkeff$Parkfitness <- maxparkeff$EnergyOverall
  maxparkeff <- dplyr::select(maxparkeff, Parkfitness)
  for (i in 1:length(euniqu)) {
    euniqu[i][[1]]$Parkfitness <- maxparkeff[i,]
  };
  return(euniqu)
}
############### Selection method
selection1        <- function(fit, Grid,teil,elitism,nelit,selstate){
  new <- do.call("rbind", fit)
  new1 <- dplyr::select(split(new, duplicated(new$Run))$'FALSE', Run,
                             Parkfitness,EnergyOverall)
  new1 <- dplyr::arrange(new1,desc(Parkfitness))
  if (elitism == "TRUE"){
    print(paste(
          "Elitarism activated. Best ", nelit, " fitness values are increased"))
    new1[1:nelit,]$Parkfitness <- new1[1:nelit,]$Parkfitness*10
  }
  selv <- nrow(new1)-3;   new1 <- new1[-seq(length(new1[[1]]),selv,-1),];nrow(new1)
  selstate <- toupper(selstate)
  if (selstate == "FIX") {
    if (teil==1){teil=1}else{teil=2}
    print(paste("Selection Percentage:", round(100/teil,2), "%"))
    nPar <- ceiling(nrow(new1)/teil);nPar
    print(paste("FIXED Selection: How many parental individuals are selected:",
                nPar, "from",  nrow(new1),"with", ((1/teil)*100), "%"))
  }
  if (selstate == "VAR") {
    nPar <- ceiling(nrow(new1)/teil);nPar
    print(paste("VARIABLE Selection: How many parental individuals are selected:",
                nPar, "from",  nrow(new1),"with", ((1/teil)*100), "%"))
  }
  if (nPar > 100){
    nPar = 100
  }
  childsRunID <- sort(sample(new1[[1]], nPar, prob= new1$Parkfitness, replace=F));
  childsRunIDCopy <- childsRunID
  chile = length(childsRunID); child = vector("list",chile);
  for (z in 1:chile){
    child[[z]] <- dplyr::select(fit[[childsRunID[z]]], Run, Rect_ID,Parkfitness)
  }
  childbin <- vector("list",chile);
  for (i in 1: chile){
    childbin[[i]] <- Grid
    childbin[[i]]$Run <- child[[i]]$Run[[1]]
    childbin[[i]]$bin <- 0
    childbin[[i]]$Fitness <- child[[i]]$Parkfitness[[1]]
    for (e in 1:length(child[[i]][[1]])) {
      rectid <- child[[i]][e,2][[1]]
      childbin[[i]][childbin[[i]]$ID==rectid,]$bin = 1
    }
  }
  parents <- vector("list",(length(childsRunID)/2));
  for (i in 1:(length(childsRunID)/2)) {
    new <- dplyr::arrange(new1[new1$Run %in% childsRunID,]); new
    parents[[i]] <- sample(x = sort(childsRunID), 2, replace = F); parents[[i]]
    childsRunID <-  childsRunID[!(childsRunID %in% parents[[i]])]; childsRunID
  }
  parall <- unlist(parents)
  childbindf <- do.call("rbind",childbin); paralli <-
                         vector("list",length(parall));
  for (i in 1:length(parall)){
    paralli[[i]] <- dplyr::select(childbindf[which(childbindf$Run %in%
                                 parall[i]),], ID,Run,bin,Fitness)
  }
  parentsall <- data.frame(paralli)
  parents_Fitness <- parentsall[1,c(1,seq(4, length(parentsall),4))]
  parentsall <- parentsall[,c(1,seq(3, length(parentsall),4))];parentsall
  parents_new <- list(parentsall,parents_Fitness)
  return(parents_new)
}
############### Checking Crossover Inputs
readinteger       <- function(){
  print("Select appropriate Method. Either 'EQU' for equal crossover
        parts or 'RAN' for random parts.")
  crPaInter <- readline(prompt = "Type 'R' for random and 'E' for equal parts.")
  if(crPaInter=="R"){crossPart = "RAN"}
  if(crPaInter=="E"){crossPart = "EQU"}
  if  (crossPart!= "EQU" & crossPart !="RAN") {
    return(readinteger())
  }
  return(crossPart)
}
############### Gene Split Function used by the Crossover-Method
splitAt           <- function(x, pos) unname(split(x,
                           cumsum(seq_along(x) %in% pos)))
############### Crossover Method
crossover1        <-function(se6,u, uplimit,crossPart,...) {
  print(paste("Crossover Point Rate: ",u+1))
  se6fit <- se6[[2]][1,-1];
  se6 <- se6[[1]]
  se6 = se6[,-1];
  parid <- sample(1:length(se6));
  z = seq(1, length(parid),2);
  all <- vector("list", length(z));
  crossPart = toupper(crossPart)
  sene2fit <- vector(mode = "list",length = length(z))
  for (e in 1:length(z)) {
    r = z[[e]];
    sene <- se6[,parid[r]];
    sene1 <- se6[,parid[r+1]];
    senefit <- se6fit[,parid[r]];senefit;   sene1fit <- se6fit[,parid[r+1]];
    sene2fit[[e]] <- senefit+sene1fit/2;
    if (crossPart == "EQU"){
      crosEquPartN <- trunc(u+1)
      t1 <- ceiling(length(sene)/crosEquPartN);
      a <- split(sene,as.numeric(gl(length(sene),t1,length(sene))));
      b <- split(sene1,as.numeric(gl(length(sene1),t1,length(sene1))));
    }
    if (crossPart == "RAN"){
      u1 <- sort(sample(2:(length(sene)-1), u, replace=F));
      a <- splitAt(sene,u1);
      b <- splitAt(sene1,u1);
    }
    x1 <- rbind(a,b);
    perm <- gtools::permutations(n=2,r=ncol(x1),v=1:nrow(x1),repeats.allowed=T);
    permut <- list()
    for (pp in 1:nrow(perm)){
      gclist <-list()
      for (gnp in 1:length(perm[pp,])){
        parent01 <- perm[pp,gnp];
        if (parent01 ==1){
          gc <- a[[gnp]];
        } else {
          gc <- b[[gnp]]
        }
        gclist[[gnp]] <- gc
      }
      permut[[pp]] <- unlist(gclist);
    }
    permut <- do.call("cbind",permut);
    all[[e]] <- permut
  }
  nuCh <- ncol(all[[1]]);
  sene2fit_n <- do.call("cbind",sene2fit);
  sene2fit_n <- (sene2fit_n/mean(sene2fit_n))
  fitChi <- rep(x = sene2fit_n,each=nuCh)
  nI <- do.call("cbind", all);
  if (length(fitChi) != ncol(nI)){
    print(paste("Crossover -  Anzahl nicht gleich. Fehler."))
    break()
  }
  print(paste("How many parental pairs are at hand: ",length(z)))
  print(paste("How many permutations are possible: ", length(z)*(2^(trunc(u)+1))))
  partaksur = ncol(nI)
  if (partaksur >= uplimit){
    partaksur = uplimit
    print(paste("Population max limit reached: ", uplimit ))
  }
  partak <- sort(sample(1:length(nI[1,]),partaksur,prob = fitChi));
  print(paste("How many permutations are selected: ", length(partak) ))
  nI <- nI[,partak]
  return(nI)}
############### Mutate the genes of every Individual
mutation          <- function(a,p) {
  for (i in 1:length(a)) {
    rnd <- runif(n = 1,min = 0,max = 1)
    if (rnd < p) {
      if (a[i] == "0") {a[i] = 1} else {a[i] =1}
    }
  }
  return(a)
}
############### Adjust thenumber of turbines per windfarm
trimton           <- function(mut, nturb, allparks, nGrids, trimForce){
  uniRect <- sort(unique(allparks$Rect_ID))
  nGrids1 <- 1:nGrids
  lepa <- length(mut[1,])
  mut1 = list();
  for (i in 1:lepa) {
    mut1[[i]] = mut[,i]
    e = mut[,i]==1
    ele = length(e[e==T]);
    zviel = ele - nturb;
    welche <- which(e==TRUE);
    uniRectRest <- sort(uniRect[uniRect %in% welche]);
    uniRectRest1 <- sort(nGrids1[!nGrids1 %in% welche]);
    trimForce <- toupper(trimForce)
    indivprop <- dplyr::select(allparks, Rect_ID, Parkfitness, AbschGesamt);
    indivprop <- indivprop %>% group_by(Rect_ID) %>% summarise_each(funs(mean));
    k = 3
    propwelche <- data.frame(cbind(RectID=welche,
                       Prop=rep(mean(indivprop$AbschGesamt),length(welche))));
    propexi <- indivprop[indivprop$Rect_ID %in% welche,];
    propexi <- as.data.frame(propexi)
    npt <- (1+((max(propexi$AbschGesam)-
             propexi$AbschGesam)/(1+max(propexi$AbschGesam))))
    npt0 <- (1+((max(propexi$Parkfitness)-
             propexi$Parkfitness)/(1+max(propexi$Parkfitness))))
    NewProb <- 1/(npt/(npt0^k))
    propwelche[welche %in%  indivprop$Rect_ID,]$Prop <- NewProb;
    propwelcheN <-  data.frame(cbind(RectID=nGrids1,
       Prop=rep(min(indivprop$AbschGesamt),length(nGrids1))));
    propexiN <- indivprop[indivprop$Rect_ID %in% nGrids1,];
    propexiN <- as.data.frame(propexiN)
    npt1 <- (1+((max(propexiN$AbschGesam)-
             propexiN$AbschGesam)/(1+max(propexiN$AbschGesam))))
    npt2 <- (1+((max(propexiN$Parkfitness)-
             propexiN$Parkfitness)/(1+max(propexiN$Parkfitness))))^k
    NewProb1 <- (npt1/npt2)
    propwelcheN[propwelcheN$RectID %in%  indivprop$Rect_ID,]$Prop <- NewProb1;
    if (!all(propwelcheN$RectID %in%  indivprop$Rect_ID==TRUE)){
      qu <- min(NewProb1)
      propwelcheN[!propwelcheN$RectID %in%  indivprop$Rect_ID,]$Prop <- qu
    }
    propwelcheN <- propwelcheN[!propwelcheN$RectID %in% welche,];
    prob1 = propwelche$Prop;
    prob2 = propwelcheN$Prop;
    if (zviel != 0) {
      if (zviel > 0) {
        if (trimForce == "TRUE"){
          smpra <- sort(sample(welche, zviel,replace=F,prob = prob1));
          prob1[which(welche==smpra[1])]
        } else {
          smpra <- sort(sample(welche, zviel,replace=F));
        }
        mut1[[i]][smpra] = 0
      } else {
        if (trimForce == "TRUE"){
          smpra <- sort(sample(propwelcheN$RectID, (-zviel),
              replace=F, prob = prob2));
        } else {
          smpra <- sort(sample(propwelcheN$RectID, (-zviel),replace=F));
        }
        mut1[[i]][smpra] = 1;
      }
    }
  }
  mut1 <- do.call("cbind", mut1)
  return(mut1)
}
############### Get the Grid-IDs from the binary code to start a new generation
getRects          <- function(trimtonOut, Grid){
  childli = list();
  len1= dim(trimtonOut)[2]
  for (i in 1:len1) {
    childli[[i]] <- trimtonOut[,i]
  }
  rectidli = list();
  for (u in 1:len1){
    rectidli[[u]] <- which(childli[[u]]==1, arr.ind = T)
  }
  childnew = list()
  for (z in 1:len1) {
    childnew[[z]] <- Grid[rectidli[[z]],];
  }
  return(childnew)
}
############### This is the main function to start an optimization
genAlgo           <- function(Polygon1, Rotor, n, fcrR=3,referenceHeight=50,
                              RotorHeight=100, SurfaceRoughness=0.14,
                              Proportionality=1, iteration=100, mutr=0.001,
                             vdirspe,   topograp="FALSE",elitism="TRUE",   nelit=6,
                              selstate="FIX",crossPart1="EQU",trimForce="TRUE"){
  library(rgdal); library(raster); library(dplyr);
  library (data.table);library(gtools); library(varhandle)
  oldpar <- par(no.readonly = T)
  plot.new();   par(ask=F);
  resol2 <- fcrR*Rotor
  CrossUpLimit = 300
  inputData <- list(Input_Data=rbind("Rotorradius"=Rotor,
                                 "Number of turbines"=n,"Grid Shape Factor"= fcrR,
                                 "Iterations"=iteration,
                                "Mutation  Rate"=mutr,  "Percentage  of  Polygon"=
                                 Proportionality, "Topographie"=topograp,
                                "Elitarism"=elitism, "Selection Method"=selstate,
                                "Trim Force Method Used"=trimForce,
                                 "Crossover Method Used"=crossPart1,
                                "Reference Height"= referenceHeight,
                                 "Rotor Height"=RotorHeight,
                                 "Resolution" = resol2));
  inputWind <- list(Windspeed_Data=vdirspe)
  ProjLAEA = "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
              +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
  if (as.character(crs(Polygon1)) != ProjLAEA) {
    Polygon1 <- spTransform(Polygon1, CRSobj = ProjLAEA)
  }
  if  (crossPart1!= "EQU" & crossPart1 !="RAN") {
    crossPart1 <- readinteger()
  }
  Grid <- GridFilter(shape = Polygon1,resol = resol2,prop = Proportionality);
```

```
AmountGrids <- nrow(Grid)
nStart = (AmountGrids*n)/iteration;
if (nStart < 70) {nStart = 70};
if (nStart > 200) {nStart = 200}
nStart<- ceiling(nStart);
startsel <- StartGA(Grid,n,nStart);
maxParkwirkungsg = 0; allparkcoeff <- vector("list",iteration);
bestPaEn <- vector("list",iteration);  bestPaEf <- vector("list",iteration);
fuzzycontr <- vector("list",iteration);
fitnessValues <- vector("list",iteration);
nindiv <- vector("list",iteration); clouddata <- vector("list",iteration);
selcross <- vector("list",iteration); beorwor <- vector("list",iteration);
mut_rate <- vector("list",iteration);   allCoords <- vector("list",iteration);
if (topograp == "FALSE"){
  print("Topography is not taken into account.")
} else if (topograp == "TRUE"){
  print("Topography is taken into account.")
  par(mfrow=c(3,1))
  Polygon1 <-  spTransform(Polygon1, CRSobj =
                  crs("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"));
  extpol <- round(Polygon1@bbox,0)[,2]
  srtm <- getData('SRTM', lon=extpol[1], lat=extpol[2]);
  srtm_crop <- crop(srtm, Polygon1);
  srtm_crop <- mask(srtm_crop, Polygon1)
  Polygon1 <-  spTransform(Polygon1, CRSobj = crs(ProjLAEA));
  srtm_crop <- projectRaster(srtm_crop, crs = crs(ProjLAEA));
  plot(srtm_crop, main="Elevation from SRTM");
  assign(x = "srtm_cropUni",value = srtm_crop, envir = .GlobalEnv)
  plot(Polygon1,add=T); plot(dry.grid.filtered,add=T)

  ####################################################################
  ## Path to CORINE LAND COVER-Raster/legend. Must be inserted to
  ## be able to run the terrain effect model.
  ## The legend must be adapted priorly; A new column "Rauhigkeit_z"
  ## must be created which specifies the land rougness of each land cover.
  ## NOTE: Land classes with "NODATA" or "UNCLASSIFIED .." need numeric
  ## "Rauhigkeit_z"-values, otherwise R will interpret them as factors.

  ccl <- raster(" C:/................./g100_06.tif  ")
  rauhigkeitz <- read.csv("C:/................/clc_legend.csv",
                          header = T,sep = ";");

  cclPoly <- crop(ccl,Polygon1)
  cclPoly1 <- mask(cclPoly,Polygon1)
  cclRaster <- reclassify(cclPoly1, matrix(c(rauhigkeitz$GRID_CODE,
                         rauhigkeitz$Rauhigkeit_z),ncol = 2))
  assign(x = "cclRasterUni",value = cclRaster, envir = .GlobalEnv)
  plot(cclRaster, main="Surface Roughness from Corine Land Cover")
}
rbPal <- colorRampPalette(c('red','green'))
i=1
while (i <= iteration) {
  if (i==1) {
    fit <- fitness(selection = startsel,referenceHeight,
            RotorHeight,SurfaceRoughness, Polygon = Polygon1,
                   resol1 = resol2,rot=Rotor, dirspeed = vdirspe,
                   srtm_crop,topograp,cclRaster)
  } else {
    getRectV <- getRects(mut1, Grid)
    fit <- fitness(selection = getRectV,referenceHeight,
            RotorHeight,SurfaceRoughness, Polygon = Polygon1,
                   resol1 = resol2,rot = Rotor, dirspeed = vdirspe,
                   srtm_crop,topograp,cclRaster)
  }
  allparks <- do.call("rbind",fit);
  allparksUni <- split(allparks, duplicated(allparks$Run))$'FALSE';
  maxparkfitness <-  round(max(allparksUni$Parkfitness),4);
  meanparkfitness <- round(mean(allparksUni$Parkfitness),3);
  minparkfitness <- round(min(allparksUni$Parkfitness),3);
  MaxEnergyRedu <-  round(max(allparksUni$EnergyOverall),2);
  MeanEnergyRedu <- round(mean(allparksUni$EnergyOverall),2);
  MinEnergyRedu <- round(min(allparksUni$EnergyOverall),2);
  allCoords[[i]] <- allparks
  maxParkwirkungsg <- round(max(allparksUni$EfficAllDir),2);
  meanParkwirkungsg <- round(mean(allparksUni$EfficAllDir),2);
  minParkwirkungsg <- round(min(allparksUni$EfficAllDir),2);
  allparkcoeff[[i]] <- cbind(maxparkfitness,meanparkfitness,minparkfitness,
                         MaxEnergyRedu,MeanEnergyRedu,MinEnergyRedu,
                         maxParkwirkungsg,meanParkwirkungsg,minParkwirkungsg)
  clouddata[[i]] <- dplyr::select(allparksUni,
                                EfficAllDir,EnergyOverall,Parkfitness);
  cat(c("\n\n", i, ": Round with coefficients ", allparkcoeff[[i]], "\n"));
  xd <- allparks[allparks$EnergyOverall==
                  max(allparks$EnergyOverall),]$EnergyOverall[1];
  ind <- allparks$EnergyOverall == xd;
  bestPaEn[[i]] <- allparks[ind,][1:n,]
  xd1 <- allparks[allparks$EfficAllDir==
                  max(allparks$EfficAllDir),]$EfficAllDir[1];
  ind1 <- allparks$EfficAllDir == xd1;
  bestPaEf[[i]] <- allparks[ind1,][1:n,]
  afvs <- allparks[allparks$EnergyOverall==max(allparks$EnergyOverall),];
  cat(paste("How many individuals exist: ",  length(fit) ), "\n");
  cat(paste("How many parks are in local Optimum: ",
     (length(afvs[,1])/n) ), "\n")
  nindivfit <- length(fit)
  lebre <- length(unique(bestPaEn[[i]]$AbschGesamt))
  if (lebre <= 2){
    Col <- "green";      Col1 <- "green"
  } else {
    Col <- rbPal(lebre)[as.numeric(cut(-bestPaEn[[i]]$AbschGesamt,
           breaks = lebre))];
    Col1 <- rbPal(lebre)[as.numeric(cut(-bestPaEf[[i]]$AbschGesamt,
           breaks = lebre))]
  }
  x = round(bestPaEn[[i]]$EnergyOverall[[1]],2);
  y = round(bestPaEn[[i]]$EfficAllDir[[1]],2);
  e = bestPaEn[[i]]$EfficAllDir;
  x1 = round(bestPaEf[[i]]$EnergyOverall[[1]],2);
  y1 = round(bestPaEf[[i]]$EfficAllDir[[1]],2);
  e1 = bestPaEf[[i]]$EfficAllDir
  allparksNewplot <- dplyr::select(allparks,AbschGesamt,Rect_ID,Parkfitness);
  allparksNewplot <- allparksNewplot %>%
                         group_by(Rect_ID) %>%
                         summarise_each(funs(mean));
  if(any(allparksNewplot$Rect_ID %in% Grid$ID == F)){
    print(paste("Index of Grid not correct. Bigger than maximum Grid? "))
    break()
  }
  par(mfrow=c(1,2))
  plot(Polygon1, main=paste(i, "Round \n Best Energy Output: ", x,
                            "\n Wirkungsgrad: ", y ),
       sub =paste("\n Number of turbines: ", length(e)));
  plot(dry.grid.filtered, add=T)
```

```
points(bestPaEn[[i]]$X,bestPaEn[[i]]$Y,col=Col,pch=20,cex=1.5);
plot(Polygon1, main=paste(i, "Round \n Best Efficiency Output: ", x1,
                          "\n Wirkungsgrad: ", y1 ),
     sub =paste("\n Number of turbines: ", length(e1)));
plot(dry.grid.filtered, add=T)
points(bestPaEf[[i]]$X,bestPaEf[[i]]$Y,col=Col1,pch=20,cex=1.5)
if (i > 20) {
  besPE <- do.call("rbind",lapply(bestPaEn[1:i], function(x)
                                  max(x$EnergyOverall)))
  maxBisher <- max(besPE); WhichMaxBs <- which(besPE==max(besPE))
  if (length(WhichMaxBs) >= 2) {
    BestForNo <- bestPaEn[sample(WhichMaxBs,2)]
    BestForNo[[1]]$Run <- length(fit)+1
    BestForNo[[2]]$Run <- length(fit)+2
  } else {
    BestForNo <- bestPaEn[WhichMaxBs]
    BestForNo <- append(BestForNo, BestForNo)
    BestForNo[[1]]$Run <- length(fit)+1
    BestForNo[[2]]$Run <- length(fit)+2
  }
  last7 <- besPE[i:(i-5)]
  if (!any(last7==maxBisher)){
    cat(paste(
         "Park with highest Fitness level to date is replaced in the list.",
         "\n\n"))
    fit <- append(fit, BestForNo)
  }
}
if (i==1) {
  t0 <- split(allparks, duplicated(allparks$Run))$'FALSE'
  t0 <- t0$Parkfitness;       fitnessValues[[i]] <- t0
  rangeFitnessVt0 <- range(t0);
  maxt0 <- max(t0);
  meant0 <- mean(t0);
  allcoef0 <- c(rangeFitnessVt0, meant0);
  fuzzycontr[[i]] <- rbind(allcoef0);
  colnames(fuzzycontr[[i]]) <- c("Min","Max","Mean")
  teil=2
  if (selstate=="VAR"){
    teil=1.35
  }
  u = 1.1
  beorwor[[i]] <- cbind(0,0)
}
if (i>=2 && i <= iteration) {
  t0 <- split(allparks, duplicated(allparks$Run))$'FALSE';
  t0 <- t0$Parkfitness;
  fitnessValues[[i]] <- t0;
  rangeFitnessVt0 <- range(t0);
  maxt0 <- max(t0);
  meant0 <- mean(t0);
  mint0 <- min(t0);
  t1 <- fitnessValues[[i-1]];
  rangeFitnessVt1 <- range(t1);
  maxt1 <- max(t1);
  meant1 <- mean(t1);
  mint1 <- min(t1)
  maxDif <- maxt0 - maxt1;
  meanDif <- meant0 - meant1;
  minDif = mint0 - mint1
  WeightDif <- c(0.80,0.2,0.0)
  maxunt <- (maxDif*WeightDif[1])+(meanDif*WeightDif[2])+(minDif*WeightDif[3])
  allcoef1 <- c(rangeFitnessVt0, meant0);
  allcoef2 <- c(rangeFitnessVt1, meant1);
  fuzzycontr[[i]] <- rbind(allcoef1,allcoef2);
  colnames(fuzzycontr[[i]]) <- c("Min","Max","Mean")
  if(maxunt<0) {
    pri="deteriorated";teil=teil-0.02; u=u-0.06} else if (maxunt==0) {
      pri="not changed"; teil=teil; u=u} else {
        pri="improved"; teil=teil+0.017; u=u+0.03}
  if (teil > 5){teil=5; u=u+0.09;
               print("Min 20% Selected");
               print(paste("CPR is increased! CPR:",u,
               "SelPerc: ",teil))}
  if (trunc(u) < 0){u = 0.5;teil=teil-0.4;
               print(paste("Min  1  CrossPoints.  Selection  decreased.  CPR:",u,
               "SelPerc: ",teil))}
  if (u >= 4){u=4;teil=4;
               print(paste("Max 5 CrossPoints. Select fittest 25%.: ",teil))}
  if (teil <= 4/3){teil = 4/3;
               print(paste("Max 75% selected. SelPerc: ",teil))}
  if (length(fit) <= 20) {teil=1;u=u+0.07;
               print(paste(
               "Less  than  20  individuals.  Select  all  and  increase  Crossover-
               point rate: ", u,"SelPerc", teil))}
  if (teil > 5){teil=5;}
  u = round(u,2); teil=round(teil,3);
  print(paste("Fitness of this generation (",i,
               "), compared with the previous, has",pri,"by",
                round(maxunt,4)))
  meanunt <- meant0-meant1;
  beorwor[[i]] <- cbind(maxunt, meanunt)
}
if (selstate=="FIX"){
  if (teil==1){teil=1} else {teil=2}
}
if (crossPart1=="EQU"){
  u=round(u,2)
}
selcross[[i]] <- cbind(cross=trunc(u+1),teil)
selec6best <- selection1(fit, Grid,teil, elitism, nelit, selstate);
selec6best_bin <- selec6best[[1]]
print(paste("Selection  -  Amount of Individuals: ",
            length(selec6best_bin[1,-1])))
nindivsel<- length(selec6best_bin[1,-1]);
crossOut <- crossover1(selec6best, u, uplimit = CrossUpLimit,
                 crossPart=crossPart1) ;
print(paste("Crossover -  Amount of Individuals: ",length(crossOut[1,])));
nindivcros<- length(crossOut[1,]);
loOp <- (length(afvs[,1])/n)
if (loOp > 2) {
  mutrn <- round(runif(1, 0.03, 0.1),2);
  t1 <- (loOp*1.25)/42
  mutrn <- mutrn * (1+(t1)); mutrn
  mutrn <- round(mutrn +((i)/(20*iteration)),5);
  mut <- mutation(a = crossOut, p = mutrn);
  mut_rat <- mutrn
  cat(paste("1. Mutation Rate is", mutrn, "\n\n"))
} else {
  mut <- mutation(a = crossOut, p = mutr);
  mut_rat <- mutr
}
mut_rate[[i]] <- mut_rat
print(paste("Mutation  -  Amount of Individuals: ",length(mut[1,])));
```

```
    nindivmut <- length(mut[1,]);
    mut1 <- trimton(mut = mut, nturb = n, allparks = allparks,
                    nGrids = AmountGrids,trimForce=trimForce)
    print(paste("TrimToN   -  Amount of Individuals: ",length(mut1[1,])))
    nindiv[[i]] <- cbind(nindivfit,nindivsel,nindivcros,nindivmut)
    if (maxParkwirkungsg == 100) {
      i = iteration + 1
    } else {
      i = i+1
    }
  }
  mut_rate <- mut_rate[lapply(mut_rate,length)!=0];
  beorwor <- beorwor[lapply(beorwor,length)!=0] ;
  selcross <- selcross[lapply(selcross,length)!=0] ;
  clouddata <- clouddata[lapply(clouddata,length)!=0];
  allparkcoeff <- allparkcoeff[lapply(allparkcoeff,length)!=0]
  bestPaEn <- bestPaEn[lapply(bestPaEn,length)!=0] ;
  bestPaEf <- bestPaEf[lapply(bestPaEf,length)!=0] ;
  fuzzycontr <- fuzzycontr[lapply(fuzzycontr,length)!=0];
  fitnessValues <- fitnessValues[lapply(fitnessValues,length)!=0];
  nindiv <- nindiv[lapply(nindiv,length)!=0]
  allCoords <- allCoords[lapply(allCoords,length)!=0] ;
  alldata <- cbind(allparkcoeff,bestPaEn,bestPaEf,fuzzycontr,
                   fitnessValues,nindiv,clouddata,selcross,beorwor,
                   inputData,inputWind,mut_rate,allCoords)
  par(oldpar);
  return(alldata)
}
############################################################
############### Plotting functions for the result #############

############### Plot the evolution of the GA
plotEvolution <- function(resultMa,ask=T, spar=0.5){
  par(mfrow=c(1,1))
  result1 <- as.data.frame(do.call("rbind", resultMa[,1]))
  plot(result1$minParkwirkungsg, xaxt='n', main="Park Efficiency per Generation",
       xlab="Generation",ylab="Park Efficiency in %", cex=1.2,col="red", pch=20,
       ylim= c(min(result1$minParkwirkungsg),max(result1$maxParkwirkungsg)))
  axis(1,at = 1:nrow(result1),tick=T)
  grid(col = "black")
  points(result1$meanParkwirkungsg,ylab="MeanxParkwirkungsg",    cex=1.2,col="blue",
        pch=20)
  points(result1$maxParkwirkungsg,ylab="maxParkwirkungsg",      cex=1.2,col="green",
        pch=20)
  x <- 1:length(result1$MaxEnergyRedu)
  lmin <- smooth.spline(x,result1$minParkwirkungsg, spar=spar);
  lines(lmin, col='red', lwd=1.2)
  lmea <- smooth.spline(x,result1$meanParkwirkungsg, spar=spar);
  lines(lmea, col='blue', lwd=1.2)
  lmax <- smooth.spline(x,result1$maxParkwirkungsg, spar=spar);
  lines(lmax, col='green', lwd=1.2)
  op <- par(ask=ask)
  on.exit(par(op))
  plot(result1$MeanEnergyRedu,xaxt='n',
      main="Energy Yield per Generation",xlab="Generation",
      ylab="Energy    in    kW",    cex=1.2,col    ="blue",    pch=20,    ylim=
      c(min(result1$MinEnergyRedu),max(result1$MaxEnergyRedu)))
  axis(1,at = 1:nrow(result1),tick=T)
  grid(col = "black")
  points(result1$MaxEnergyRedu,ylab="maxParkwirkungsg", cex=1.2,col="green",
        pch=20)
  emean <- smooth.spline(x,result1$MeanEnergyRedu, spar=spar); lines(emean,
                         col='blue', lwd=1.2)
  emax <- smooth.spline(x,result1$MaxEnergyRedu, spar=spar); lines(emax,
                  col='green', lwd=1.2)
}
############### Plot the best X efficiency/energy results
plotResult    <-    function(resultMa,Polygon1,best=5,plotEn=1,topographie="FALSE"){
require(data.table);require(raster)
  op <- par(ask=FALSE);   on.exit(par(op));   par(mfrow=c(1,1))
  ProjLAEA = "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
  +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
  if (as.character(crs(Polygon1)) != ProjLAEA) {
    Polygon1 <- spTransform(Polygon1, CRSobj = ProjLAEA)
  }
  rbPal1 <- colorRampPalette(c('green','red'))
  resultSafe <- resultMa
  if (plotEn == 1) {
    a <- sapply(resultMa[,2], "[", "EnergyOverall")
    b <- data.frame(sapply(a, function(x) x[1]))
    order1 <- order(b, decreasing = F)
    resultMa <- resultMa[,2][order1]
    ledup <- length(resultMa)
    rectid <- (lapply(resultMa, function(x) x$Rect_ID));
    rectidt <- !duplicated(rectid)
    resultMa <- resultMa[rectidt]
    ndif <- length(resultMa)
    cat(paste("N different optimal configurations:", ndif,
              "\nAmount duplicates:", (ledup-ndif)))
    if (ndif < best) {
      cat(paste("\nNot enough unique Optimas.
          Show first best Half of different configurations."))
      best = trunc(ndif/2)
    }
    resultMa <- resultMa[(length(resultMa)-best+1):(length(resultMa))]
    for (i in (1:length(resultMa))){
      EnergyBest <- data.frame(resultMa[[i]])
      br = length(levels(factor(EnergyBest$AbschGesamt)))
      if (br > 1) {
        Col <- rbPal1(br)[as.numeric(cut(as.numeric(EnergyBest$AbschGesamt),
                                          breaks = br))]
      } else {
        Col = "green"
      }
      EnergyBest$EnergyOverall <- round(EnergyBest$EnergyOverall, 2)
      EnergyBest$EfficAllDir <- round(EnergyBest$EfficAllDir, 2)
      plot(Polygon1, col="lightblue",
         main=paste("Best                  Energy:",                   (best+1)-i,
                    "\n","Energy    Output",    EnergyBest$EnergyOverall[[1]],"kW",
"\n",               "Efficiency:",EnergyBest$EfficAllDir[[1]]));
      plot(dry.grid.filtered,add=T)
      mtext("Total wake effect in %", side = 2)
      points(EnergyBest$X,EnergyBest$Y,cex=2,pch=20,col=Col)
      text(EnergyBest$X,  EnergyBest$Y,  round(EnergyBest$AbschGesamt,0),  cex=0.8,
           pos=1, col="black")
      distpo <- dist(x = cbind(EnergyBest$X,EnergyBest$Y),method = "euclidian")
      mtext(paste("minimal Distance", round(min(distpo),2)), side = 1,line=0)
      mtext(paste("mean Distance", round(mean(distpo),2)), side = 1,line=1)
    }
  }
  if(topographie=="TRUE" && plotEn == 1){
    resol= as.integer(resultSafe[1,]$inputData['Resolution',])
    RotorHeight <- as.integer(resultSafe[1,]$inputData['RotorHeight',])
```

```
polygon1=Polygon1
sel1=EnergyBest[,1:2]
windpo <- 1
Polygon1  <-   spTransform(Polygon1,  CRSobj  =  crs("+proj=longlat  +datum=WGS84
                        +ellps=WGS84 +towgs84=0,0,0"));
extpol <- round(Polygon1@bbox,0)[,2]
srtm <- getData('SRTM', lon=extpol[1], lat=extpol[2]);
srtm_crop <- crop(srtm, Polygon1);
srtm_crop <- mask(srtm_crop, Polygon1)
Polygon1 <-  spTransform(Polygon1, CRSobj = crs(ProjLAEA));
srtm_crop <- projectRaster(srtm_crop, crs = crs(ProjLAEA));


ccl <- raster("C:/................./g100_06.tif ")
rauhigkeitz <- read.csv("C:/................./clc_legend.csv",
                        header = T,sep = ";");


cclPoly <- crop(ccl,Polygon1); cclPoly1 <- mask(cclPoly,Polygon1)
cclRaster <- reclassify(cclPoly1,
         matrix(c(rauhigkeitz$GRID_CODE,rauhigkeitz$Rauhigkeit_z),ncol=2))
orogr1 <- calc(srtm_crop, function(x) {x/(cellStats(srtm_crop,mean,na.rm=T))})
orogrnum  <-  raster::extract(x=  orogr1,  y  =  as.matrix((sel1)),  buffer=resol*2,
                            small=T,fun= mean,na.rm=T);orogrnum
windpo <- windpo * orogrnum
heightWind <- raster::extract(x= srtm_crop, y = as.matrix((sel1)), small=T,fun=
                            max,na.rm=T);heightWind
par(mfrow=c(1,1))
plot(srtm_crop, main="SRTM Elevation Data");points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(heightWind,0),cex=0.8);plot(polygon1,add=T)
plot(orogr1, main="Wind Speed Multipliers");points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(windpo,3),cex=0.8);plot(polygon1,add=T)
HeighttoBaro <- matrix(heightWind); colnames(HeighttoBaro) <- "HeighttoBaro"
air_dt <- BaroHoehe(matrix(HeighttoBaro),HeighttoBaro)
par(mfrow=c(1,1))
plot(srtm_crop,
     main="Normal Air Density",
     col=topo.colors(10));points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = rep(1.225,nrow(sel1)),cex=0.8);plot(polygon1,add=T)
plot(srtm_crop, main="Corrected Air Density",
     col=topo.colors(10));points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(air_dt$rh,2),cex=0.8);plot(polygon1,add=T)
SurfaceRoughness0 <- raster::extract(x= cclRaster, y = as.matrix((sel1)),
                                     buffer=resol*2, small=T,fun= mean,
                                      na.rm=T);
SurfaceRoughness1 <- raster::extract(x=terrain(srtm_crop,"roughness"),
                                      y = as.matrix((sel1)),
                                     buffer=resol*2, small=T,fun= mean,
                                      na.rm=T);
SurfaceRoughness <-SurfaceRoughness0*
                   (1+(SurfaceRoughness1/max(res(srtm_crop))));
elrouind <- terrain(srtm_crop,"roughness")
elrouindn <- resample(elrouind,cclRaster,method="ngb")
modSurf <- overlay(x = cclRaster,y = elrouindn, fun=
                   function(x,y){return(x*(1+(y/max(res(srtm_crop)))))})
par(mfrow=c(1,1)); cexa=0.9
plot(cclRaster,
      main="Corine Land Cover Roughness");points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(SurfaceRoughness0,2),
       cex=cexa);plot(polygon1,add=T)
plot(x=terrain(srtm_crop,"roughness",neighbors = 4),
      main="Elevation Roughness Indicator");
points(sel1$X,sel1$Y,pch=20);
```

```
  textxy(sel1$X,sel1$Y,labs = round((SurfaceRoughness1) ,2), cex=cexa);
  plot(polygon1,add=T)
  plot(modSurf, main="Modified Surface Roughness");points(sel1$X,sel1$Y,pch=20);
  textxy(sel1$X,sel1$Y,labs = round((SurfaceRoughness),2),
           cex=cexa);plot(polygon1,add=T)
  k_raster <- calc(modSurf, function(x) {x= 0.5/(log(RotorHeight/x))})
  k = 0.5/(log(RotorHeight/SurfaceRoughness))
  par(mfrow=c(1,1)); cexa=0.9
  plot(k_raster, main="Adapted Wake Decay Constant - K");
  points(sel1$X,sel1$Y,pch=20); textxy(sel1$X,sel1$Y,labs = round((k),3),
                                     cex=cexa);
  plot(polygon1,add=T)
}
if (plotEn == 2){
  a <- sapply(resultMa[,3], "[", "EfficAllDir")
  b <- data.frame(sapply(a, function(x) x[1]))
  order2 <- order(b, decreasing = F)
  resultMa <- resultMa[,3][order2]
  ledup <- length(resultMa)
  rectid <- lapply(resultMa, function(x) x$Rect_ID)
  rectidt <- !duplicated(rectid)
  resultMa <- resultMa[rectidt]
  ndif <- length(resultMa)
  cat(paste("N different optimal configurations:", ndif, "\nAmount duplicates:",
     (ledup-ndif)))
  if (ndif < best) {
    cat(paste("\nNot enough unique Optimas. Show first best Half of different
           configurations."))
    best = trunc(ndif/2)
  }
  resultMa <- resultMa[(length(resultMa)-best+1):(length(resultMa))]
  for (i in (1:length(resultMa))){
    EfficiencyBest <- data.frame(resultMa[[i]])
    br = length(levels(factor(EfficiencyBest$AbschGesamt)))
    if (br > 1) {
      Col1 <- rbPal1(br)[as.numeric(cut(EfficiencyBest$AbschGesamt,breaks = br))]
    } else {
      Col1 = "green"
    }
    EfficiencyBest$EnergyOverall <- round(EfficiencyBest$EnergyOverall, 2)
    EfficiencyBest$EfficAllDir <- round(EfficiencyBest$EfficAllDir, 2)
    plot(Polygon1, col="lightblue", main=paste("Best Efficiency:", (best+1)-i,
           "\n","Energy Output", EfficiencyBest$EnergyOverall[[1]], "kW", "\n",
           "Efficiency", EfficiencyBest$EfficAllDir[[1]]));
    plot(dry.grid.filtered,add=T)
    mtext("Total wake effect in %", side = 2)
    points(EfficiencyBest$X,EfficiencyBest$Y,col=Col1,cex=2,pch=20)
    text(EfficiencyBest$X, EfficiencyBest$Y, round(EfficiencyBest$AbschGesamt,0),
           cex=0.8, pos=1)
    distpo <- dist(x = cbind(EfficiencyBest$X,EfficiencyBest$Y),
                   method = "euclidian")
    mtext(paste("minimal Distance", round(min(distpo),2)), side = 1,line=0)
    mtext(paste("mean Distance", round(mean(distpo),2)), side = 1,line=1)
  }
}
if(topographie=="TRUE" && plotEn == 2){
  resol= as.integer(resultSafe[1,]$inputData['Resolution',])
  polygon1=Polygon1
  sel1=EfficiencyBest[,1:2]
  windpo <- 1
  if (1==1){
    Polygon1 <-   spTransform(Polygon1, CRSobj = crs("+proj=longlat +datum=WGS84
                                          +ellps=WGS84 +towgs84=0,0,0"));
```

```
extpol <- round(Polygon1@bbox,0)[,2]
srtm <- getData('SRTM', lon=extpol[1], lat=extpol[2]);
srtm_crop <- crop(srtm, Polygon1);
srtm_crop <- mask(srtm_crop, Polygon1)
Polygon1 <-  spTransform(Polygon1, CRSobj = crs(ProjLAEA));
srtm_crop <- projectRaster(srtm_crop, crs = crs(ProjLAEA));


ccl <- raster("C:/................../g100_06.tif")
rauhigkeitz <- read.csv("C:/................../clc_legend.csv",
                        header = T, sep = ";");


cclPoly <- crop(ccl,Polygon1); cclPoly1 <- mask(cclPoly,Polygon1)
cclRaster <- reclassify(cclPoly1,
       matrix(c(rauhigkeitz$GRID_CODE,rauhigkeitz$Rauhigkeit_z),ncol = 2))
orogr1 <- calc(srtm_crop, function(x)
                          {x/(cellStats(srtm_crop,mean,na.rm=T))})
orogrnum <- raster::extract(x= orogr1, y = as.matrix((sel1)), buffer=resol*2,
                          small=T,fun= mean,na.rm=T);orogrnum
windpo <- windpo * orogrnum
heightWind <- raster::extract(x= srtm_crop, y = as.matrix((sel1)),
                    small=T,fun= max,na.rm=T);heightWind
par(mfrow=c(1,1))
plot(srtm_crop, main="SRTM Elevation Data");points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(heightWind,0),cex=0.8);plot(polygon1,add=T)
plot(orogr1, main="Wind Speed Multipliers");points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(windpo,3),cex=0.8);plot(polygon1,add=T)
HeighttoBaro <- matrix(heightWind); colnames(HeighttoBaro) <- "HeighttoBaro"
air_dt <- BaroHoehe(matrix(HeighttoBaro),HeighttoBaro)
par(mfrow=c(1,1))
plot(srtm_crop, main="Normal Air Density",
       col=topo.colors(10));points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs =
       rep(1.225,nrow(sel1)),cex=0.8);plot(polygon1,add=T)
plot(srtm_crop, main="Corrected Air Density",
       col=topo.colors(10));points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(air_dt$rh,2),cex=0.8);plot(polygon1,add=T)
SurfaceRoughness0 <- raster::extract(x= cclRaster, y = as.matrix((sel1)),
                                     buffer=resol*2, small=T,fun=
                                     mean,na.rm=T);
SurfaceRoughness1 <- raster::extract(x=terrain(srtm_crop,"roughness"),
                                      y = as.matrix((sel1)),
                                     buffer=resol*2, small=T,fun=
                                      mean,na.rm=T);
SurfaceRoughness <-SurfaceRoughness0*
                    (1+(SurfaceRoughness1/max(res(srtm_crop))));
elrouind <- terrain(srtm_crop,"roughness")
elrouindn <- resample(elrouind,cclRaster,method="ngb")
modSurf <- overlay(x = cclRaster,y = elrouindn, fun=function(x,y)
                    {return(x*(1+(y/max(res(srtm_crop)))))})
par(mfrow=c(1,1)); cexa=0.9
plot(cclRaster, main="Corine Land Cover Roughness");
points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round(SurfaceRoughness0,2),cex=cexa);
plot(polygon1,add=T)
plot(x=terrain(srtm_crop,"roughness",neighbors = 4),
     main="Elevation Roughness Indicator");
points(sel1$X,sel1$Y,pch=20); textxy(sel1$X,sel1$Y,labs =
       round((SurfaceRoughness1),2),cex=cexa);plot(polygon1,add=T)
plot(modSurf, main="Modified Surface Roughness");
points(sel1$X,sel1$Y,pch=20);
textxy(sel1$X,sel1$Y,labs = round((SurfaceRoughness),2),
              cex=cexa);plot(polygon1,add=T)
       k_raster <- calc(modSurf, function(x) {x= 0.5/(log(RotorHeight/x))})
       k = 0.5/(log(RotorHeight/SurfaceRoughness))
       par(mfrow=c(1,1)); cexa=0.9
       plot(k_raster, main="Adapted Wake Decay Constant - K");
       points(sel1$X,sel1$Y,pch=20);
        textxy(sel1$X,sel1$Y,labs = round((k),3),
              cex=cexa);
       plot(polygon1,add=T)
     }
   }
}
############### Plot result with Google maps background
GooglePLot    <-    function(resultMa,Polygon1,best=1,plotEn=1){    require(rgeos);
require(RgoogleMaps)
  op <- par(ask=F)
  on.exit(par(op))
  par(mfrow=c(1,1))
  if (plotEn == 1) { en = "EnergyOverall" }
  if (plotEn == 2) { en = "Parkfitness"   }
  resultMa <- resultMa[,2][order(as.character(sapply(resultMa[,2],
                      "[", en)),decreasing = T)]
  resultMa <- resultMa[best]
  Solution <- do.call("rbind", resultMa)
  ProjPoly = proj4string(Polygon1);
  ProjLAEA = "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
  +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
  ProjLonLat <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0"
  if (proj4string(Polygon1)!=(ProjLonLat)){
    print("Polygon ist nicht in LatLon. Projiziere.GooglePlot")
    Polygon1 <- spTransform(Polygon1, CRSobj = crs(ProjLonLat))
  }
  PointSol <- data.frame(dplyr::select(Solution,X,Y));
  names(PointSol) <- c("lon","lat")
  PointSol1 <- SpatialPoints(coordinates(PointSol), proj4string = crs(ProjLAEA))
  PointSol1 <- spTransform(PointSol1, CRSobj = crs(ProjLonLat))
  PointSol1 <- as.data.frame(PointSol1)
  map <- GetMap(center = c(extent(gCentroid(Polygon1))[4],
                         extent(gCentroid(Polygon1))[1]), zoom = 11,
                          size= c(640,640))
  PlotOnStaticMap(MyMap = map, lat = PointSol1$lat, lon = PointSol1$lon,
                  zoom = 11, size= c(640,640),
                  cex = 1.1, pch = 19, col = "red", FUN = points, add = F)
  PlotPolysOnStaticMap(MyMap = map, polys = SpatialPolygons(Polygon1@polygons,

proj4string=Polygon1@proj4string),
                       border = NULL, lwd = 0.25, add=T)
  invisible(PointSol1)
}
############### Plot result in Google Chrome with real base map
GoogleChromePLot      <-      function(resultMa,Polygon1,best=1,plotEn=1)       {
require(googleVis)
  a <- GooglePLot(resultMa,Polygon1,best,plotEn)
  a$latlon <- paste(a$lat, a$lon, sep=":")
  map.gb <- gvisMap(a, locationvar="latlon", options = list(showTip=T, showLine=T,
                    enableScrollWheel=TRUE,mapType="hybrid",
                    useMapTypeControl=T, width=1400, height=800, icons=paste0("{",
                    "'default': {'normal': 'http://icons.iconarchive.com/",
                    "icons/icons-land/vista-map-markers/48/",
                    "Map-Marker-Ball-Azure-icon.png',\n",
                    "'selected': 'http://icons.iconarchive.com/",
                    "icons/icons-land/vista-map-markers/48/",
                    "Map-Marker-Ball-Right-Azure-icon.png'", "}}"))
  )
  plot(map.gb)
}
############### Plot all fitness values of the GA
plotcloud <- function(result,pl="FALSE"){
  opar <- par(no.readonly = T)
  clouddata <- result[,7]
  EffCloud <- lapply(clouddata, function(x) x = x[[1]]);EffCloud
  EneCloud <- lapply(clouddata, function(x) x = x[[2]]);EneCloud
  FitCloud <- lapply(clouddata, function(x) x = x[[3]]);FitCloud
  EffCldInd <- list();EneCldInd <- list();FitCldInd <- list();
  for (i in 1:length(clouddata)){
    l <- length(clouddata[[i]]$EfficAllDir); l
    EffCldInd[[i]] <- t(as.matrix(rbind(rep(i,l),EffCloud[[i]])));EffCldInd
    EneCldInd[[i]] <- t(as.matrix(rbind(rep(i,l),EneCloud[[i]])));EffCldInd
    FitCldInd[[i]] <- t(as.matrix(rbind(rep(i,l),FitCloud[[i]])));EffCldInd
  }
  EffCldInd <- do.call("rbind",EffCldInd)
  EffCldIndmax <- data.frame(EffCldInd)
  EffCldIndmax <- group_by(EffCldIndmax,X1) %>%
    summarise(max=max(X2),mean=mean(X2),min=min(X2),sd=sd(X2))
  EneCldInd <- do.call("rbind",EneCldInd)
  EneCldIndmax <- data.frame(EneCldInd)
  EneCldIndmax <- group_by(EneCldIndmax,X1) %>%
    summarise(max=max(X2),mean=mean(X2),min=min(X2),sd=sd(X2))
  FitCldInd <- do.call("rbind",FitCldInd)
  FitCldIndmax <- data.frame(FitCldInd)
  FitCldIndmax <- group_by(FitCldIndmax,X1) %>%
    summarise(max=max(X2),mean=mean(X2),min=min(X2),sd=sd(X2))
  if (pl=="TRUE"){
    par(mfrow=c(2,3))
    plot(FitCldInd, main="Fitness",xlab="Generation",ylab="Fitnessvalue",
       pch=20,col="red",cex=1.3);
    lf <- smooth.spline(x=FitCldInd[,1],y=FitCldInd[,2], spar=0.1);
    lines(lf, col='red', lwd=1.2)
    points(x=FitCldIndmax$X1,y=FitCldIndmax$max,type="l",col="red")
    points(x=FitCldIndmax$X1,y=FitCldIndmax$min,type="l",col="red")
    plot(EffCldInd, main="Efficiency",xlab="Generation",
         ylab="Efficiency in %",pch=20,col="orange",cex=1.3);
    le <- smooth.spline(x=EffCldInd[,1],y=EffCldInd[,2], spar=0.1);
    lines(le, col='orange', lwd=1.2)
    points(x=EffCldIndmax$X1,y=EffCldIndmax$max,type="l",col="orange")
    points(x=EffCldIndmax$X1,y=EffCldIndmax$min,type="l",col="orange")
    plot(EneCldInd, main="Energy",xlab="Generation",ylab="Energy in kW",
pch=20,col="blue",cex=1.3);
    len <- smooth.spline(x=EneCldInd[,1],y=EneCldInd[,2], spar=0.1);
    lines(len, col='blue', lwd=1.2)
    points(x=EneCldIndmax$X1,y=EneCldIndmax$max,type="l",col="blue")
    points(x=EneCldIndmax$X1,y=EneCldIndmax$min,type="l",col="blue")
    plot(x=FitCldIndmax$X1,y=FitCldIndmax$sd,  main="Standard  Deviation  Fitness",
          xlab="Generation", ylab="Standard Deviation of Population",
          col="red",type="b")
    plot(x=EffCldIndmax$X1,y=EffCldIndmax$sd, main="Standard Deviation Efficiency",
         xlab="Generation", ylab="Standard Deviation of Population",
         col="orange",type="b")
    plot(x=EneCldIndmax$X1,y=EneCldIndmax$sd,   main="Standard   Deviation   Energy",
         xlab="Generation", ylab="Standard Deviation of Population",
         col="blue",type="b")
  }
  clouddatafull <- cbind(Fitn=FitCldIndmax,Eff=EffCldIndmax,Ene=EneCldIndmax)
  par(opar)
  invisible(clouddatafull)
}
############### Plot the development of the GA
plotparkfitness <- function(result,spar=0.5){
  rslt <- as.data.frame(do.call("rbind", result[,'allparkcoeff']))
  mutres <- as.data.frame(do.call("rbind", result[,'mut_rate']))
  nindiv1 <- as.data.frame(do.call("cbind", result[,'nindiv']))
  nindiv1 <- nindiv1[-seq(4,length(nindiv1),4)]
  opar <- par(no.readonly = T)
  selcross <- unlist(result[,'selcross'])
  selteil <- selcross[seq(2,length(selcross),2)]
  crossteil <- selcross[seq(1,length(selcross),2)]
  layout(matrix(c(1,1,1,1,2,3,4,5),2,4, byrow = TRUE));
  rbPal <- colorRampPalette(c('red','green'));
  Col <- rbPal(4)[as.numeric(cut(as.numeric(rslt$maxparkfitness),breaks = 4))]
  plot(rslt$minparkfitness, xaxt='n', main="Parkfitness per Generation",
       ylab="Parkfitness in %", cex=1.2,col="red", pch=20,
       ylim= c(min(rslt$minparkfitness),max(rslt$maxparkfitness)));
  axis(1,at = 1:nrow(rslt),tick=T)
  points(rslt$meanparkfitness,ylab="MeanParkF", cex=1.2,col="blue", pch=20);
  points(rslt$maxparkfitness,ylab="maxParkF", cex=1.2,col="green", pch=20)
  x <- 1:length(rslt$maxparkfitness)
  lmin <- smooth.spline(x,rslt$minparkfitness, spar=spar);
  lines(lmin, col='red', lwd=1.2);
  lmea <- smooth.spline(x,rslt$meanparkfitness, spar=spar);
  lines(lmea, col='blue', lwd=1.2);
  lmax <- smooth.spline(x,rslt$maxparkfitness, spar=spar);
  lines(lmax, col='green', lwd=1.2)
  grid(col = "gray")
  par(mar=c(5,5,3,2))
  farbe <- rep(seq(1,3,1),length(nindiv1)/3);farbe;
  ndindiplot <- as.integer(nindiv1)
  plot(ndindiplot,type="b",col=farbe,cex=2,pch=20, main="N-Individuen",
      axes = FALSE, ylab="N",ylim=c(0,max(ndindiplot)+100))
  axis(side = 2,tick = TRUE);
  axis(side = 1,tick = TRUE,at =seq(1,length(ndindiplot),3),
                                 labels =(1:(length(ndindiplot)/3)))
  legend("topleft",title="Amount of Individuals in: ",lty = c(1,1,1),
         cex=0.5,inset = c(0.01,0.01),
         box.lty=0,box.lwd=0,c("Fitness","Selection","Crossover"),
         col=farbe[1:3],xjust = 0)
  plot(1*100/selteil,ylim=c(20,110),type="b",cex=2,col="green",
       pch=20,main="Selection percentage",ylab="Percentage",xlab="Generation")
  grid(col = "gray")
  selrpl <- 1*100/selteil;timeticksel <- which(selrpl>75);
  selrplval <- selrpl[selrpl>75]
  textxy(timeticksel,selrplval,labs = timeticksel,cex = 0.7)
  plot(crossteil,col=crossteil,main="n Crossoverparts",xlab="Generation",
       ylab="Crossover Points",ylim=c(1,8),cex=1,pch=15); grid(col = "gray")
  timetickcro <- which(crossteil>median(crossteil));
  crorplval <- crossteil[crossteil>median(crossteil)]
  textxy(timetickcro,crorplval,labs = timetickcro,cex = 0.5)
  plot(as.numeric(t(mutres)),type="b",main="Mutation Rate",xlab="Generation",
       ylab="Crossover Points",cex=1,pch=15)
  mutrpl <- as.numeric(t(mutres));   timetick <- which(mutrpl>median(mutrpl));
  mutrplval <- mutrpl[mutrpl>median(mutrpl)]
  textxy(timetick,mutrplval,labs = timetick,cex = 0.7)
  grid(col = "gray")
  op1 <- par(ask=T)
  on.exit(par(op1))
  par(mfrow=c(1,1))
  plot(ndindiplot,type="b",col=farbe,cex=2,pch=20, main="N-Individuen",
       axes = FALSE, ylab="N",ylim=c(0,max(ndindiplot)+100))
  axis(side = 2,tick = TRUE);
  axis(side = 1,tick = TRUE,at =seq(1,length(ndindiplot),3),
       labels =(1:(length(ndindiplot)/3)))
  legend("topleft",title="Amount of Individuals in: ",pch = c(20,20,20),cex=1,
         inset = c(0.01,0.01), box.lty=0,box.lwd=0,
         c("Fitness","Selection","Crossover"),  text.col=farbe[1:3],  col=farbe[1:3],
         xjust = 0)
  op2 <- par(ask=T)
  on.exit(par(op2))
  par(mfrow=c(2,1))
  plot(1*100/selteil,ylim=c(20,110),type="b",cex=2,col="green",pch=20,
       main="Selection percentage",ylab="Percentage",xlab="Generation")
  grid(col = "gray")
  selrpl <- 1*100/selteil;timeticksel <- which(selrpl>75);
  selrplval <- selrpl[selrpl>75]
  textxy(timeticksel,selrplval,labs = timeticksel,cex = 0.5)
  plot(crossteil,col=crossteil,main="n Crossoverparts",xlab="Generation",
       ylab="Crossover Points",ylim=c(1,8),cex=1,pch=15); grid(col = "gray")
  timetickcro <- which(crossteil>median(crossteil));
  crorplval <- crossteil[crossteil>median(crossteil)]
  textxy(timetickcro,crorplval,labs = timetickcro,cex = 0.5)
  op3 <- par(ask=T)
  on.exit(par(op3))
  par(mfrow=c(1,1))
  rbPal <- colorRampPalette(c('red','green'));
  Col <- rbPal(4)[as.numeric(cut(as.numeric(rslt$maxparkfitness),breaks = 4))]
  plot(rslt$minParkwirkungsg, xaxt='n', main="Mutation Influence", ylab=" in %",
       cex=1.2,col="red", pch=20, ylim= c(min(rslt$minParkwirkungsg),
                                           max(rslt$maxParkwirkungsg)));
  axis(1,at = 1:nrow(rslt),tick=T)
  points(rslt$meanParkwirkungsg,ylab="MeanParkEff", cex=1.2,col="blue", pch=20);
  points(rslt$maxParkwirkungsg,ylab="maxParkEff", cex=1.2,col="green", pch=20)
  x <- 1:length(rslt$maxparkfitness);
  lmin <- smooth.spline(x,rslt$minParkwirkungsg, spar=spar);
  lines(lmin, col='red', lwd=1.2);
  lmea <- smooth.spline(x,rslt$meanParkwirkungsg, spar=spar);
  lines(lmea, col='blue', lwd=1.2)
  lmax <- smooth.spline(x,rslt$maxParkwirkungsg, spar=spar);
  lines(lmax, col='green', lwd=1.2);
  grid(col = "gray")
  if (length(timetick)!=0){
    abline(v = timetick,col="black");
    mtext(mutrplval,side = 3,at = timetick,cex = 0.8)
  }
  op4 <- par(ask=T)
  on.exit(par(op4))
  par(mfrow=c(1,1))
  rbPal <- colorRampPalette(c('red','green'));
  Col <- rbPal(4)[as.numeric(cut(as.numeric(rslt$maxparkfitness),breaks = 4))]
  plot(rslt$minParkwirkungsg, xaxt='n', main="Selection Influence", ylab=" in %",
       cex=1.2,col="red", pch=20, ylim= c(min(rslt$minParkwirkungsg),
                                           max(rslt$maxParkwirkungsg)));
  axis(1,at = 1:nrow(rslt),tick=T)
  points(rslt$meanParkwirkungsg,ylab="MeanParkEff", cex=1.2,col="blue", pch=20);
  points(rslt$maxParkwirkungsg,ylab="maxParkEff", cex=1.2,col="green", pch=20)
  x <- 1:length(rslt$maxparkfitness);
  lmin <- smooth.spline(x,rslt$minParkwirkungsg, spar=spar);
  lines(lmin, col='red', lwd=1.2);
  lmea <- smooth.spline(x,rslt$meanParkwirkungsg, spar=spar);
  lines(lmea, col='blue', lwd=1.2)
  lmax <- smooth.spline(x,rslt$maxParkwirkungsg, spar=spar);
```

```
lines(lmax, col='green', lwd=1.2);   grid(col = "gray")
if (length(timeticksel)!=0){
  abline(v = timeticksel,col="green");
  mtext(selrplval,side = 3,at = timeticksel,col="green",cex = 0.8)
}
op5 <- par(ask=T)
on.exit(par(op5))
par(mfrow=c(1,1))
rbPal <- colorRampPalette(c('red','green'));
Col <- rbPal(4)[as.numeric(cut(as.numeric(rslt$maxparkfitness),breaks = 4))]
plot(rslt$minParkwirkungsg, xaxt='n', main="Crossover Influence", ylab=" in %",
     cex=1.2,col="red", pch=20, ylim= c(min(rslt$minParkwirkungsg),
                                        max(rslt$maxParkwirkungsg)));
axis(1,at = 1:nrow(rslt),tick=T)
points(rslt$meanParkwirkungsg,ylab="MeanParkEff", cex=1.2,col="blue", pch=20);
points(rslt$maxParkwirkungsg,ylab="maxParkEff", cex=1.2,col="green", pch=20)
x <- 1:length(rslt$maxparkfitness);
lmin <- smooth.spline(x,rslt$minParkwirkungsg, spar=spar);
lines(lmin, col='red', lwd=1.2);
lmea <- smooth.spline(x,rslt$meanParkwirkungsg, spar=spar);
lines(lmea, col='blue', lwd=1.2)
lmax <- smooth.spline(x,rslt$maxParkwirkungsg, spar=spar);
lines(lmax, col='green', lwd=1.2);   grid(col = "gray")
if (length(timetickcro)!=0){
  abline(v = timetickcro,col="red");
  mtext(crorplval,side = 3,at = timetickcro,col="red",cex = 0.8)
}
sddata <- plotcloud(result);
fitsd <- dplyr::select(sddata,contains("fit"));
effsd <- dplyr::select(sddata,contains("eff"));
enesd <- dplyr::select(sddata,contains("ene"));
op6 <- par(ask=T)
on.exit(par(op6))
par(mfrow=c(4,1))
plot(rslt$minparkfitness,    xaxt='n',    main="Parkfitness    per    Generation",
     ylab="Parkfitness in %", cex=1.2,col="red", pch=20,
     ylim= c(min(rslt$minparkfitness),max(rslt$maxparkfitness)))
axis(1,at = 1:nrow(rslt),tick=T)
grid(col = "black")
points(rslt$meanparkfitness,ylab="MeanParkF", cex=1.2,col="blue", pch=20)
points(rslt$maxparkfitness,ylab="maxParkF", cex=1.2,col="green", pch=20)
x <- 1:length(rslt$maxparkfitness)
lmin <- smooth.spline(x,rslt$minparkfitness, spar=spar);
lines(lmin, col='red', lwd=1.2)
lmea <- smooth.spline(x,rslt$meanparkfitness, spar=spar);
lines(lmea, col='blue', lwd=1.2)
lmax <- smooth.spline(x,rslt$maxparkfitness, spar=spar);
lines(lmax, col='green', lwd=1.2)
plot(1*100/selteil,ylim=c(20,110),type="b",lwd=2,col="green",pch=20,
     main="Selection percentage",ylab="Percentage",xlab="Generation");
grid(lty = 2)
plot(crossteil,col=crossteil,pch=20,cex=2,main="n Crossoverparts",
     xlab="Generation",ylab="Crossover Points",ylim=c(1,6));grid(lty = 2)
plot(enesd$Ene.sd, type="b",col="blue",pch=20,lwd=2,main="Standard Deviation");
grid(lty = 2)
par(new = TRUE)
plot(effsd$Eff.sd, type="b",col="orange",lwd=2,axes = FALSE, bty = "n",
     xlab = "", ylab = "",pch=20)
par(new = TRUE)
plot(fitsd$Fitn.sd, type="b",col="red",lwd=2,axes = FALSE, bty = "n",
     xlab = "", ylab = "",pch=20)
timeticksd <- which(mutrpl>median(mutrpl));
```

```
  sdrplval <- fitsd$Fitn.sd[timeticksd]
  if (length(timeticksd) != 0){
    textxy(timeticksd,sdrplval,labs = timeticksd,cex = 0.5)
    abline(v = timeticksd)
    mtext(mutrplval,side = 3,at = timetick,cex = 0.8)
  }
  op7 <- par(ask=T)
  on.exit(par(op7))
  par(mfrow=c(1,1))
  plot(fitsd$Fitn.sd, type="b",col="red",lwd=2,axes = TRUE, bty = "n", xlab = "",
       ylab = "",pch=20, main="Mutation Rate influence on Standard Deviation")
  if (length(timeticksd) != 0){
    textxy(timeticksd,sdrplval,labs = timeticksd,cex = 0.7)
    abline(v = timeticksd)
    mtext(mutrplval,side = 3,at = timetick,cex = 0.8)
  }
  par(opar)
}
############### Plot the compared fitness values
plotfitnessevolution <- function(result,spar=0.1){
  opar <- par(no.readonly = T)
  x <- result[,4];x
  x <- x[-c(1)];x
  x1 <- do.call("rbind",x);x1
  par(mar= c(4,5,4,2))
  result <- as.data.frame(do.call("rbind", result[,1]))
  layout(mat = matrix(c(1,2,3,4,4,4), nrow = 2, ncol = 3, byrow = TRUE))
  minge <- x1[seq(1,length(x1[,1]),2),1];minge
  minge2 <- x1[seq(2,length(x1[,2]),2),1];minge2
  ming3 <- minge-minge2; ming3
  ming3 <- c(0,ming3)
  ming3 <- as.data.frame(ming3)
  ming3$farbe <- 0
  ming3$farbe[ming3$ming3 < 0]  <- "red" ;
  ming3$farbe[ming3$ming3 > 0] <- "green";
  ming3$farbe[ming3$ming3 == 0] <- "orange"
  plot(ming3$ming3,type="b",col=ming3$farbe,pch=20,
      cex=2,xlab="Generation",ylab="Beter or Worse");
  title(main="Minimal Fitness Values",
        sub = "compared to previous generation",col.main="red"); abline(0,0)
  grid(col = "black")
  meange <- x1[seq(1,length(x1[,1]),2),3];meange
  meange2 <- x1[seq(2,length(x1[,2]),2),3];meange2
  meag3 <- meange-meange2; meag3
  meag3 <- c(0,meag3)
  meag3 <- as.data.frame(meag3)
  meag3$farbe <- 0
  meag3$farbe[meag3$meag3 < 0]  <- "red" ;
  meag3$farbe[meag3$meag3 > 0] <- "green";
  meag3$farbe[meag3$meag3 == 0] <- "orange"
  plot(meag3$meag3,type="b",col=meag3$farbe,pch=20,cex=2,
      xlab="Generation",ylab="Beter or Worse");
  title(main="Mean Fitness Values",
      sub = "compared to previous generation",col.main="orange"); abline(0,0)
  grid(col = "black")
  maxge <- x1[seq(1,length(x1[,1]),2),2];maxge
  maxge2 <- x1[seq(2,length(x1[,2]),2),2];maxge2
  mg3 <- maxge-maxge2;mg3;
  mg3 <- c(0,mg3)
  mg3 <- as.data.frame(mg3)
  mg3$farbe <- 0
  mg3$farbe[mg3$mg3 < 0]  <- "red" ;
  mg3$farbe[mg3$mg3 > 0] <- "green";
  mg3$farbe[mg3$mg3 == 0] <- "orange"
  plot(mg3$mg3,type="b",col=mg3$farbe,pch=20, cex=2,xlab="Generation",
       ylab="Beter or Worse");
  title(main="Maximal Fitness Values",
  sub = "compared to previous generation",col.main="darkgreen"); abline(0,0)
  grid(col = "black")
  rbPal <- colorRampPalette(c('red','green'))
  Col <- rbPal(4)[as.numeric(cut(as.numeric(result$maxparkfitness),breaks = 4))]
  plot(result$minparkfitness, xaxt='n', main="Parkfitness per Generation",
       ylab="Parkfitness in %", xlab="Generation",cex=2,col="red", pch=20,
       ylim= c(min(result$minparkfitness),max(result$maxparkfitness)))
  axis(1,at = 1:nrow(result),tick=T)
  grid(col = "black")
  points(result$meanparkfitness,ylab="MeanParkF", cex=2,col="blue", pch=20)
  points(result$maxparkfitness,ylab="maxParkF", cex=2,col="green", pch=20)
  x <- 1:length(result$maxparkfitness)
  lmin <- smooth.spline(x,result$minparkfitness, spar=spar);
  lines(lmin, col='red', lwd=1.6)
  lmea <- smooth.spline(x,result$meanparkfitness, spar=spar);
  lines(lmea, col='blue', lwd=1.6)
  lmax <- smooth.spline(x,result$maxparkfitness, spar=spar);
  lines(lmax, col='green', lwd=1.6)
  par(opar)
}
############### Plot a Heatmap of the selected Grid Cells
heatMap  <- function(result,si=2, Polygon1){
  bpe <- do.call("rbind",result[,'allCoords']);
  bpe <- bpe[c(1,2)]
  sizing = as.integer(result[,'inputData'][[1]][,1]['Resolution'])/si
  dupco <- geoR::dup.coords(bpe,simplify = TRUE);
  bpe$Ids <- as.integer(rownames(bpe));
  dupco <- lapply(dupco, function(x) as.integer(x));
  dupcosum <- lapply(dupco, function(x) length(x));
  bpenew <- vector("list",length(dupco))
  for (i in 1:length(dupco)){
    bpenew[[i]] <- bpe[bpe$Ids==dupco[[i]][1],];
    bpenew[[i]]$Sum <- dupcosum[[i]][1]
  }
  bpenew <- do.call("rbind",bpenew);
  bpenew <- bpenew[-3]
  polo <- sp::SpatialPoints(sp::coordinates(cbind(bpenew$X,bpenew$Y)))
  exMar <- 50
  x.range <-   range(bpenew$X); y.range <-   range(bpenew$Y)
  grd <- expand.grid(x=seq(from=x.range[1]-exMar, to=x.range[2]+exMar, by=sizing),
                     y=seq(from=y.range[1]-exMar, to=y.range[2]+exMar, by=sizing))
  sp::coordinates(grd) <- ~ x+y;   sp::gridded(grd) <- TRUE
  idwout <- as.data.frame(gstat::idw(formula = bpenew$Sum~1,
                          locations = polo,newdata=grd))
  plot1<-ggplot2::ggplot(data=idwout,mapping=ggplot2::aes(x=x,y=y))+
    c(ggplot2::geom_tile(data=idwout,ggplot2::aes(fill=var1.pred)))+
    ggplot2::geom_point(data=bpenew,mapping=ggplot2::aes(x=X,y=Y),
                        show.legend = TRUE,size=sqrt(sqrt(bpenew$Sum)),alpha=0.6)
  plot1+ggplot2::scale_fill_gradient(low="red",high="green")+ggplot2::coord_equal()
}

############################################################
############### Function-Calls to start an optimization and Plot the results
#############
# p <- plot.windrose(spd = data.in$ws,dir = data.in$wd, dirres=10, spdmax=20)
# Res_Poly <- genAlgo(AreaConsidered, n=12, SurfaceRoughness=0.3,Rotor=30,fcrR=3,
#                     RotorHeight=60,referenceHeight=60,iteration=5,
#                     Proportionality=1,mutr=0.08,vdirspe = data.in,
#                     topograp="FALSE", elitism="TRUE",nelit=7, selstate="FIX",
#                     crossPart1 = "EQU", trimForce="TRUE")
# result = Res_Poly
# Polygon = AreaConsidered
# plotResult(resultMa = result, Polygon1 = Polygon, best = 1,
#            plotEn =1,topographie = "FALSE");
# heatMap(result,si=5,Polygon1 = Polygon)
# plotEvolution(result,T,0.3)
# plotparkfitness(result,0.1)
# plotfitnessevolution(result)
# plotcloud(result,"TRUE")
# GoogleChromePLot(result,Polygon,1,1)
```