

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

**Flächendeckende Ermittlung potenzieller
Windturbinenstandorte**
**Area-wide determination of potential wind turbine
locations**

Masterarbeit
Informatik

April 24, 2025

Vorgelegt von Presented by	Marvin Krebber Matrikelnummer: 407219 marvin.krebber@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. rer. nat. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Zweitprüfer Second examiner	Prof. Dr. rer. nat. Thomas Noll Lehr- und Forschungsgebiet: Software Modellierung und Verifikation RWTH Aachen University
Betreuer Supervisor	Dr. rer. nat. Pascal Richter Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. rer. nat. Pascal Richter, for his support throughout this thesis and for the opportunity to write my Master thesis about this topic. I would like to thank my brother Julian Ruiz for his feedback and his help proofreading the thesis. Lastly, I would like to thank Prof. Dr. rer. nat. Erika Ábrahám and Prof. Dr. rer. nat. Thomas Noll for agreeing to examine my work.

Contents

1	Introduction	1
1.1	List of Abbreviations	2
1.2	Method	2
2	Testing Methodology	3
2.1	State of the Art	4
2.2	Contribution	6
2.3	Outline	7
3	Wind Potential	7
3.1	Global Wind Atlas Implementation	10
3.1.1	GeoTiff data format	10
3.1.2	Performance	11
3.1.3	Verification	11
3.2	DWD Implementation	12
3.2.1	Performance	12
3.2.2	Verification	13
3.3	Conclusion	13
4	Potential Area	14
4.1	Geographic Data	14
4.1.1	Official real estate cadasters	16
4.2	Potential Area Simulation	19
4.2.1	First approach	19
4.2.2	Second approach	22
4.2.3	Comparison of precision	24
4.3	Correctness Errors of the First Approach	25
4.4	Verification	27
4.4.1	Potential areas by distance regulations	27
4.4.2	Potential areas by wind exclusion	28
4.5	Optimizing Performance of the First Approach	29
4.5.1	Optimising get_minimum_distances	29
4.5.2	Optimising find_points_in_geometries	29
4.5.3	connect_points	30
4.6	General	30
4.7	Performance	31
4.7.1	Final performance of the first approach	31
4.7.2	Performance of the second approach	31
4.7.3	Conclusion	32
5	Optimal Positioning	32
5.1	Optimization Goals	33

5.2	Optimization Algorithms	33
5.2.1	Wind data	34
5.2.2	Chosen algorithm	34
5.3	Formal Problem	35
5.4	Grouping Areas Together	36
5.5	Greedy Ellipse Implementation	37
5.6	Maximizing Ellipse Sizes	39
5.6.1	Binary search for global maximum ellipse size	39
5.6.2	Binary search to maximize individual ellipse size	40
5.7	Encountered Problems	41
5.8	Other Performance Improvements	43
5.9	Annealing Pattern Implementation	43
5.9.1	Generating the grid	44
5.9.2	Optimize found grid alignment	45
5.10	Comparison of the Results	45
5.10.1	Chosen metrics	46
5.11	Greedy Ellipse vs Pattern Annealing	47
5.11.1	Runtime	51
5.12	Comparison to Greedy Ellipse Algorithm	53
5.12.1	Comparison of annual energy production	53
5.12.2	Comparison of wake loss	54
5.12.3	Comparison of leveled cost of energy	55
5.13	Positioning Comparison	56
5.13.1	Better placements than attractiveness	56
5.13.2	Worse placements than attractiveness	56
5.13.3	Direct positioning comparison	57
5.14	Comparison Conclusion	60
6	Future Work	60
7	Conclusion	61
	References	62
A	Figures	64
	List of Tables	70
	List of Figures	70
	List of Listings	71

1 Introduction

Land-based wind turbines have emerged as one of the most cost-effective sources of energy. They offer a renewable, environmentally friendly option, as the climate crisis worsens and energy security is becoming worse. The Ukraine war has shown how dependent Germany is from fossil fuel imports and how important it is to have a secure cheap energy source for the economy. Reducing greenhouse gas emissions and maintaining security of supply is crucial for the economic future and the climate.

Currently, wind energy accounts for 26.2% of German power generation and thus represents the largest share of renewable energy [14]. Until 2030, 80% of electricity production must be derived from renewable sources, which currently stands at 62% [14]. This is the current political goal [14].

This paper is about showing a way to quickly and accurately determine the potential areas where wind turbines can be built and determine the according optimal positioning of the wind turbines. If state or federal laws change, the potential areas and the optimal positioning can be swiftly recalculated by only changing the input configuration.

The expansion of wind energy has plummeted after 2017, as the German government introduced more complicated approval procedures, which are now being addressed [14]. This shows how complicated regulations can hinder the expansion of renewable energy. Determining areas where wind turbines can be built according to regulations in a customizable fast way is crucial, since they vary state by state. There are many regulations regarding distance regulations, which specify how far away from different real estate types wind turbines can be built. For example, wind turbines are only allowed to be built at least 1000 m away from residential areas.

The implementation follows the criteria, methods, and data sources of the official German wind potential studies to make the results more impactful and comparable. The correctness and accuracy are verified by comparing the results of the test project Herzogenrath to other sources like potential area studies and the commercial tool windPRO. The performance improvements are shown by a comparison between the profiling of the pre-existing versus the improved/new implementation. The Python implementation uses a JSON data structure for the input configuration and output. Python was chosen for its vast libraries and ease of use. Even though the Python native functions are slow, they can be replaced with fast libraries like NumPy¹.

There are three main steps until the optimal positioning of wind turbines. Every step is verified, optimized, extended, and sped up, and new approaches are presented. The three major steps are the collection of the real estate and wind data, the determination of potential areas and the optimization of turbine placements inside a potential area. For each step there was already a pre-existing implementation, which was developed by David Wenderdel, Nicolai Radke [13], Jakob Dobmeier and others at the hybrid systems chair.

The errors in fetching the geometric data have been corrected and the official real estate cadaster for each German state has been implemented.

¹NumPy, URL <https://numpy.org/>

A new approach to the pre-existing implementation of the potential area calculation is presented, which is faster and as accurate as the underlying geometric data allows. The wind data is used to exclude areas that do not meet the minimum wind criteria, which was not in the previous implementation and the Global Wind Atlas is added as a wind data source.

In contrast to the potential area calculation, where a solution can be found quickly and accurately, the optimal positioning of the wind turbines in the potential areas is a very complex problem. Not only must the turbine placement be varied, but also the turbine amount and the turbine type/height. The goal of this paper is to determine a good approximation of the optimal positioning and resulting energy output as fast as possible.

A greedy approach and a simulated annealing approach are taken and compared, to achieve a runtime of seconds instead of minutes, while maintaining a good wind farm efficiency. For instance, such speed allows calculating the additional potential energy output of a city quickly if new turbines were to be built. This would be useful for city planning if, for example, the local laws change and the distance to residential areas is reduced.

1.1 List of Abbreviations

AEP	Annual Energy Production
ALKIS/ATKIS/LIKA	Official real estate cadastre
CRS	Coordinate Reference System
DWD	German weather service
GIS	Geographic Information System
GWA	Global Wind Atlas
INSPIRE	Infrastructure for Spatial Information in Europe
km	Kilometer
LCOE	Levelized Cost of Energy
m/s	Meter per second
MIP	Mixed Integer Programming
NaN	Not a Number
OSM	Open Street Map
WFS	Web Feature Service
WMS	Web Map Service

1.2 Method

There are three consecutive steps to find the potential areas and the corresponding optimal positions for wind turbines, which can be seen in Figure 1.

The first step is to fetch the geographic data (GIS objects) of the project area and the historic wind data (wind data always needs to be fetched).

The second step is to calculate the potential areas in which turbines could be built according to the law. Every state has its own regulations, which specify a distance of how far away wind turbines can be built from different real estate types. For example, a law might require a wind turbine to be at least one kilometer apart from residential areas. The specified distance regulations and optionally the historic wind data are used to exclude areas that do not meet the required distance regulations and the wind criteria.

In the final step, the optimal positioning of the wind turbines in the potential areas is determined, which maximizes the set goal, such as the annual energy output. The wind data and pre-existing turbines are taken into account to calculate the optimal turbine positions and the resulting energy output. The pre-existing turbine placements do not have to be taken into account if they were considered in the previous step or if they should be rebuilt, for example.

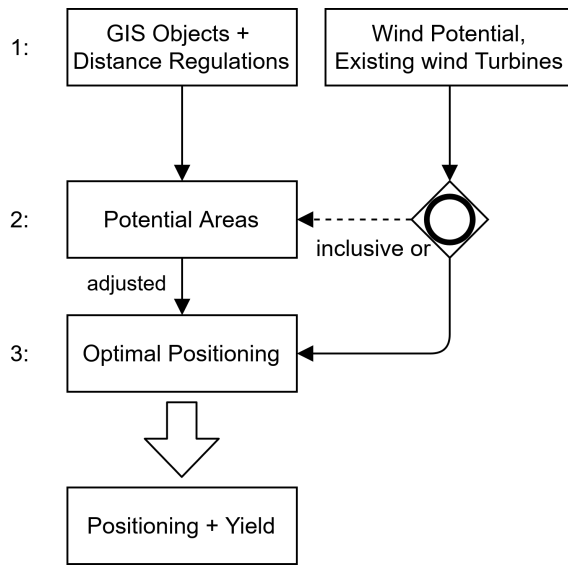


Figure 1: BPMN Representation of the process steps resulting in potential areas and corresponding optimal positioning and its power yield.

2 Testing Methodology

All performance results and verifications were conducted on the ‘project’ Herzogenrath, which is a small city in North Rhine-Westphalia, Germany. The city also conducted a wind potential study, which was used to verify the results of the potential area calculation. The project area is the rectangle with the coordinates 50.80721, 6.0065; 50.91653, 6.16443.

All performance tests were conducted on an AMD Ryzen 5 5600X 6-Core @ 3.70GHz with 32GB of RAM. The performance metrics were measured and the function runtimes were inspected with the help of the Python profiler cProfile² and are visualized with SnakeViz³.

²cProfile, URL <https://docs.python.org/3/library/profile.html>

³SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

Since the performance was in the seconds to minutes range and remained consistent across multiple tests, taking the average of multiple runs was not necessary. An exception is the reading of the GIS data from storage because the file is so big, which is why it is excluded from the runtimes in optimal positioning. For potential area both runtimes with and without reading the GIS data from storage are shown, because the saved files needed to be read changed. The variance in the runtime of reading the GIS data from storage was at maximum 1 second for a 50MB file.

GDAL⁴, a python library, which is used for the geospatial data processing, does not run on Windows without docker, which is why the runtime for calculations that require it were tested in docker (OS virtualization).

2.1 State of the Art

In this section the state-of-the-art methods and existing approaches are presented.

There are many studies on the potential areas for wind turbines in Germany. The most important ones are the federal wind potential study [7] and the state's potential area studies like North Rhine Westphalia [6], Lower Saxony [8], Saarland [15].

The potential area calculation and the wind exclusion directly follow the criteria of the German federal study [8]. All areas with an average wind speed less than 7m/s at 150m height are excluded [6]. Additionally, all areas that are within a certain distance from different real estate types are excluded, because of the regulations of the individual states.

To calculate the potential areas, Lower Saxony [8] uses a PostgreSQL-Database with a PostGIS spatial database extension, which is accessed by Python. They buffer the GIS data with the Database and then translate the GIS objects into a boolean grid with a resolution of $25 \times 25m$. They overlay these grids resulting in the exclusion areas.

The first approach of the potential area calculation was based on the approach of Lower Saxony [8]. The new approach improves upon the first approach by removing the boolean grid and by directly merging the GIS objects.

The real estate data used in the studies is the official real estate cadaster of the states.

The most common wind data sources are adopted from the individual states wind potential studies, which are the German Weather Service (DWD) and the Global Wind Atlas (GWA).

The optimal positioning of wind turbines is a highly complex problem with many variables [11]. The optimal positioning of wind turbines depends on the used wake model. The used wake model is the Jensen Park model, which is a simplified model of the wind flow around wind turbines. This is still state of the art for annual energy production (AEP) calculations for large wind farm calculations, which should be performant [11].

Energy and economy objectives are the main objective functions for the optimal positioning of wind turbines [11]. Pre-selecting different most suitable wind turbine mod-

⁴GDAL, URL <https://gdal.org>

els and corresponding heights is a common approach to reduce the search space [11]. Nonuniform land based wind farms reach an about 9.84% higher AEP and an 2.1% lower Levelized Cost of Energy (LCOE), but the variation of turbine types increases the runtime significantly [11].

The state-of-the-art algorithms for optimal positioning are meta-heuristic algorithms such as a Genetic Algorithm and a Particle Swarm Algorithm, which are the most common algorithms [11].

- **Genetic algorithms (GA)** is inspired by the process of natural selection in nature. The algorithm starts with a population of potential solutions called chromosomes. Each chromosome is evaluated based on a fitness function, which is the objective function. The selection process chooses the wind turbines from the population with the highest fitness values. The selected wind turbines are then combined and mutated to create another generation of the population, which is called crossover. The mutation is a random change of the chromosomes to introduce diversity into the population. It stops when a certain number of generations is reached or when the fitness function converges.
- **Particle Swarm algorithms (PSA)** are similar to the genetic algorithm, but they are inspired by the social behavior of birds and fish. The population consists of particles which represent potential solutions. Each particle has a position and a velocity in the search space and the direction to the best found solution. Every particle will move towards the areas with the best possible solution, because the swarm converges towards the best solution.
- **Simulated Annealing (SA)** is a probabilistic optimization algorithm that is inspired by the annealing process in metallurgy. The algorithm starts with a high temperature and gradually cools down. The algorithm starts with a random solution and iteratively explores the search space by making small changes to the current solution. A solution is accepted if it improves the objective function or if it is accepted with a certain probability based on the current temperature. The temperature is gradually decreased, which reduces the probability of accepting worse solutions. It stops when the temperature reaches a certain threshold or when a certain number of iterations is reached.
- **Greedy Algorithm** is a simple algorithm that makes the locally optimal choice at each step. It iteratively builds a solution by selecting the best option available at each step with a criterion. The process is repeated until a complete solution is found or no more options are available. The greedy algorithm is not guaranteed to find the global optimum, but it is very fast and can work well for some problems.

The federal wind potential study [7] uses a greedy approach placing wind turbines greedily in the areas with the highest wind speeds. They place circles with the radius 4 times the turbine diameter as a simplification of turbines.

Another more accurate, but very slow approach is contained in the pre-existing implementation. It calculates an attractiveness matrix for every grid point in the potential area, which is then used to calculate the optimal positioning with an algorithm that chooses the points greedily according to the attractiveness score [13].

2.2 Contribution

The goal of this work is to verify, speed up, extend and find new approaches for every step of the pre-existing implementation until coming to the result of the optimal positioning of wind turbines. Before this work, there was a pre-existing implementation for most calculations, which was developed by David Wenderdel, Nicolai Radke [13], Jakob Dobmeier and others at the hybrid systems chair.

The pre-existing implementation for the DWD wind data source is verified. Then the global wind atlas data source is newly implemented, which can be used as an alternative. It was added because it was used in multiple official wind potential studies. The DWD implementation is verified by comparing it to its North Rhine-Westphalia Poster [9]. The GWA implementation is verified by comparing it to its online atlas [1].

The fetching of the GIS objects from the Open Street Map (OSM) and a prototype of the GIS object fetching for some of the official real estate cadasters was in the pre-existing implementation. The GIS object data sources are verified for correctness, which had some errors in fetching and properly parsing the GIS objects. The official real estate cadaster of every state, which is available for free use, is implemented.

The initial approach of the potential area calculation was pre-existing, but it was very slow and had some errors. First, the initial potential area calculation is corrected with the help of the wind potential study results of the city Herzogenrath. The calculation has been optimized and accelerated, resulting in a final performance that is 11.5 times faster than the previous implementation, with a runtime of 11.8 seconds, even including all the validation corrections. Of the total runtime, 4.5 seconds is just reading the GIS object data from storage.

A new approach to calculating the potential areas was implemented, which is more accurate and faster than the optimized first approach. The exclusion of areas that do not meet the minimum wind criteria is newly implemented for both approaches.

For optimal positioning, the Levelized Cost of Energy (LCOE) and Annual Energy Production (AEP) calculations were in the pre-existing implementation, but there were errors found in the AEP calculation, which were fixed. For greedy ellipse optimal positioning there was an implementation, which served as a base for the new implementation. The pre-existing implementation of the greedy ellipse approach was not optimized for maximum turbine count (specified beforehand) and several steps and optimizations were missing.

For optimal positioning, a fast greedy ellipse approach led to a good approximation of the optimal positioning and the resulting energy output. The performance and the positioning are compared to the greedy attractiveness matrix approach from the pre-existing implementation and the commercial software windPRO in the potential areas of the Herzogenrath example. The performance of the newly implemented greedy

ellipse approach takes seconds compared to the minutes of the greedy matrix approach. The greedy ellipse approach reached a maximum energy deviation of 2% and in some cases reaching an even better result than the attractiveness matrix approach. The Levelized Cost of Energy only deviated by at maximum 0.2 cents per kWh, which is very good, since the LCOE is very sensitive to the energy output.

2.3 Outline

In section 3, the criteria for wind potential suitability are discussed, and the German wind potential studies are analyzed for which wind data source they use and what accuracy they possess. The performance and correctness of the German weather service and of the new Global Wind Atlas implementation are stated, and the inner workings are explained.

In section 4, the geographic data sources are discussed since they directly influence the results of the potential area results. Then the APIs for the official real estate cadasters of the individual states are explained, since they are the most accurate and relevant. Then the potential area simulation is explained in detail. This section discusses two approaches that come to the same solution, although one is more accurate and faster. The correctness of the simulation is verified, and the improvement steps are stated to speed up the initial implementation. The exclusion of areas that do not meet the minimum wind criteria is explained. In the end, the performance of the two approaches is compared.

In section 5, the optimal positioning simulation is explained. First, the possible optimization goals and the optimization algorithms are discussed. Then, the implementation of the greedy ellipse approach and the according heuristics are explained. Additionally, the implementation of the pattern annealing approach is explained and compared to the greedy ellipse approach. Finally, the runtime and the performance are compared to the pre-existing greedy matrix attractiveness approach and the commercial software windPRO.

In section 6, the future changes, optimizations, and potential results are discussed.

3 Wind Potential

The wind data is used to exclude areas with low wind potential and to calculate the optimal positioning of the wind turbines. For the analysis of the wind potential, three metrics can be considered: wind speed, wind power and power generated by a specific turbine model (turbine power). The methodology in this paper follows the methods of the federal wind potential area study to ensure comparability with the individual states studies, which follow the federal study's methodology.

The criteria for wind speed and power stated in the federal study [8] are adopted by using the mean wind speed or wind power at a height of 150 meters above ground [6] and excluding all areas which have wind speeds below $7m/s$ or which have a wind power density below $250W/m^2$ [8] [6].

For these wind thresholds, the data basis has to be taken into account. Different wind data sources can differ greatly from one another in terms of average wind speeds, because they are measured in different time frames, more often or less often, and in different heights. Therefore, the absolute figures are only meaningful in conjunction with the underlying data source. As can be seen in Table 1, the wind data sources differ from state to state. Four states use proprietary data and some do not consider wind potential at all, since the entire state is above the minimum wind speed threshold. The rest use data from the German Weather Service (DWD) or the Global Wind Atlas (GWA).

While the DWD data implementation was already pre-existing, the Global Wind Atlas is newly implemented as a data source to make the result more comparable to the individual states studies.

States	Data source	API Status
Brandenburg	dwd.de	Implemented
Berlin	no restriction	—
Baden-Württemberg	Government released data	available
Bavaria	Government released data	available
Bremen	N/A	N/A
Hesse	Government released data	N/A
Hamburg	N/A	N/A
Mecklenburg-Western Pomerania	dwd.de	Implemented
Lower Saxony	globalwindatlas.info	newly Implemented
North Rhine-Westphalia	Government released data	available
Rhineland-Palatinate	Government released data	available
Schleswig-Holstein	no restriction	—
Saarland	globalwindatlas.info	newly Implemented
Saxony	Government released data	N/A
Saxony-Anhalt	globalwindatlas.info	newly Implemented
Thuringia	N/A	N/A

Table 1: Wind data sources of the wind potential area studies of the individual states

The DWD data has a precision of a 200×200 m grid [10], while the GWA precision is 175.8×278 m grid [2]. The wind area potential study in Saarland states that the global wind atlas overestimates the wind speed by about 1m/s and are not directly comparable to wind reports [15]. This is also what my own observations show in the Herzogenrath project. The Herzogenrath mean wind speeds in 100m height are in the range 4.8–6.0 m/s and 6.4–8.4 m/s for the DWD and GWA data respectively. The wind speeds for LANUV are in the range 5.25–6.75 m/s, as seen in Figure 2. Therefore, the DWD data is about 0.4 m/s lower than the LANUV data, which should be more accurate.

This difference is because of the measuring precision in meters and in time (how often was measured as well as which time range), since these are mean wind speeds, the more data points the more accurate the result. For the GWA the wind was measured every 6 hours [2], but for DWD the wind was measured every hour [10]. The DWD data is from 1981 to 2000 [10], while the GWA data is from 1979 to 2013 [2] and the LANUV NRW data is from 1981 to 2010 [12]. The GWA data is the newest and the DWD data is the oldest, which is one reason for the difference in the mean wind speeds.

The wind was measured at the heights 50 m, 80 m, 90 m, 100 m for DWD and in 10 m, 50 m, 100 m, 150 m, 200 m for GWA [10] [2]. If heights between these heights are requested they are interpolated by the pre-existing implementation, based on the winds in the heights above and below the requested height. The GWA data includes wind heights above 100 m, unlike DWD, which makes the GWA data more suitable for wind turbines taller than 100 m, since the interpolated data is more accurate. The 150 m wind height is used for excluding areas by wind speed for the potential areas, as done in the federal study. Therefore, for the potential areas, the GWA data is more suitable, since a wind height at 150 m is contained in the data. However, the DWD data has more heights under 100 m, which makes the DWD data more accurate for wind turbines lower than 100 m.

To calculate the optimal positioning, the wind should be measured on site, since the mentioned data can differ enough from the actual wind speeds. The wind speeds based on these data sources should only indicate where to exclude areas and where the wind speeds are the highest, but not the exact wind speeds.

The relief in Figure 4, 5 and 2 shows that the ‘shape’ of the wind speeds stays the same, but the absolute values can differ up to 2m/s. Therefore, I recommend using the GWA data as not to exclude areas, but to use the DWD data as a fallback.

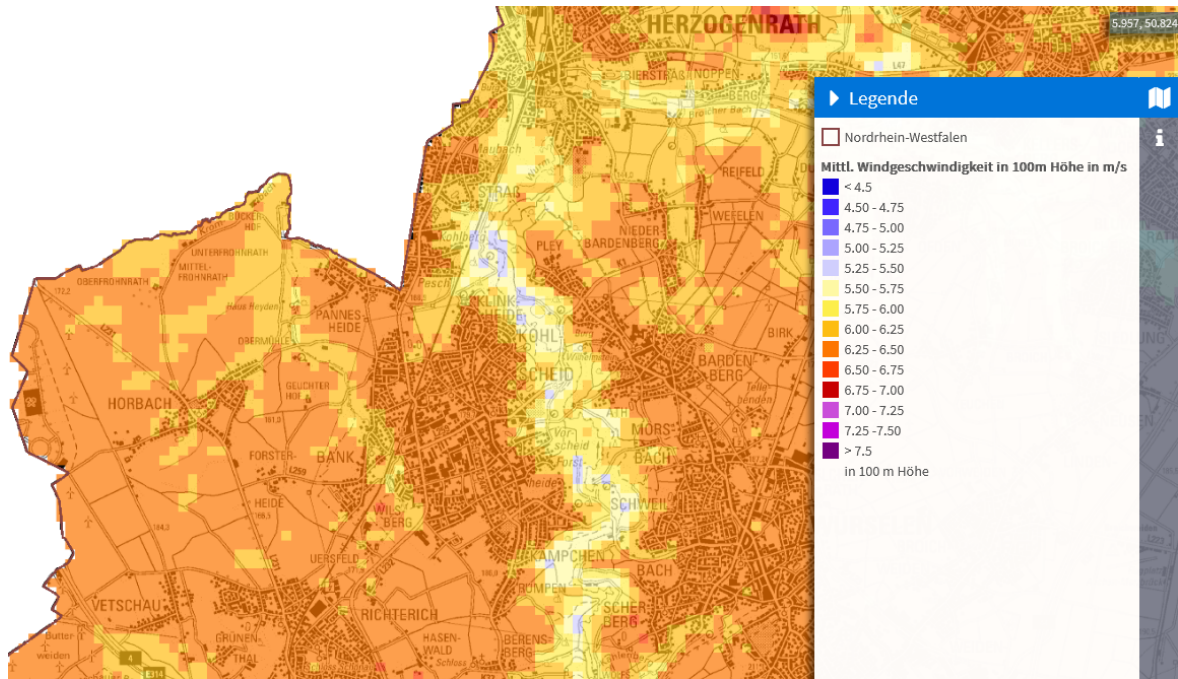


Figure 2: Energy atlas NRW¹ mean wind speed in 100m height

3.1 Global Wind Atlas Implementation

The GWA data is stored in a GeoTiff format, which is a raster image format for storing georeferenced data.

3.1.1 GeoTiff data format

The Geographic Tag Image File Format (GeoTiff) is essentially a normal Tiff image, but with additional georeferencing information, which is stored in the Tiff tags.

The optionally available additional metadata consists of the horizontal and vertical data, the spatial extent, the type of coordinate reference system (CRS), the spatial resolution, layer information, ellipsoid and geoid.

The horizontal data consists of the coordinates (latitude and longitude) of the image data, while the vertical data represents the elevation. The spatial extent is the real world area the image data covers, the spatial resolution is measured in pixel values per unit length and the layer information specifies what type of data each layer consists of. For example, there may be multiple layers for different wind speeds or wind power values. Ellipsoids and geoids are models of the shape of the earth, which are used for the image.

The data is stored in a raster format, which is a grid of cells, where each cell has a value, which represents the wind speed or wind power in this case. Missing values will often be represented by a special value, like 0 or NaN, which is used in the GWA data.

¹Energieatlas NRW, URL <https://www.energieatlas.nrw.de/site/planungskarten/wind>

The goal is to extract a small rectangular area from the image data, that corresponds to the coordinates of the project, as can be seen in the visualization Figure 3 in red. To accomplish that, first the data is stored in a 2D array, where the first dimension is the height and the second dimension is the width of the image. The height and width of the data can be converted to latitude and longitude values with the help of the georeferencing information. Then the upper left corner and the lower right corner of the required area are calculated by converting the project's latitude and longitude values to row and col values of the array, as can be seen in Figure 3 in red. Then the array is sliced to the required area and every pixel in the remaining array is converted to a latitude, longitude, wind speed or wind power value tuple, which is used for further processing.

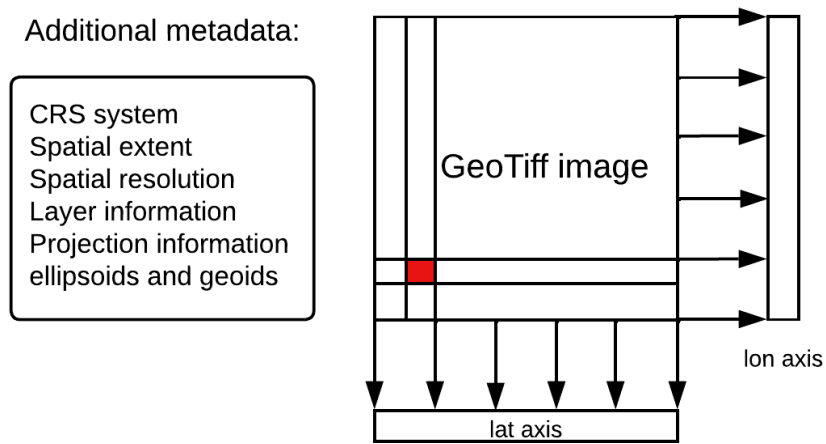


Figure 3: GeoTiff Image format

3.1.2 Performance

If the requested data is in the data set, the data only is processed into a data format, which can directly be used for further simulations. Otherwise, the data is interpolated to the requested wind height. As seen in Figure 32 and Figure 33, the GWA data processing is faster than the DWD data processing, since the GWA data is stored in a GeoTiff format, which is faster to read than the CSV format used for the DWD data. Extracting the data for the Herzogenrath project takes about 5 seconds, with height interpolation included, which is an acceptable time for the data processing. The GWA runtime was tested in Docker (operating system virtualization) because GDAL (Python library) does not run on my Windows machine without virtualization.

3.1.3 Verification

The web atlas of the global wind atlas is less precise than the actual data, but as you can see in Figure 4, the data is read correctly by the GWA implementation, since all the data points match.

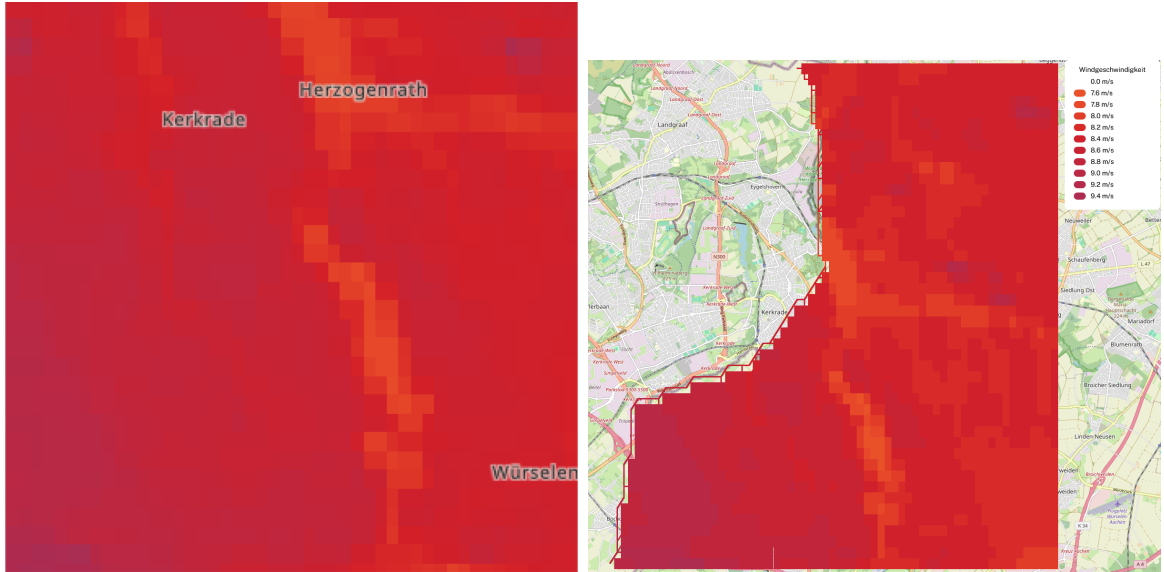


Figure 4: The maps show the mean wind speed in 150 m height. Left is the GWA online atlas [1] and right the same area, but the data was processed and visualized by the implemented code.

3.2 DWD Implementation

The DWD data is stored in CSV format, which are a grid of cells, where each cell stores the wind speed. It can be precomputed which latitude and which longitude maps to which row and column and the required rows and columns are converted into a 2D array. This was already implemented in the pre-existing implementation.

3.2.1 Performance

Since the turbine power or the wind power might not be required, an option to skip such output is newly implemented. It takes a long time to calculate the turbine power for big areas, because integrals are necessary for calculation, which are very slow.

All Python native math functions were converted to NumPy functions, which are faster, like `np.sqrt` and `np.log`.

The Figure 33 shows the performance of the DWD data processing, which takes about 8 seconds for the Herzogenrath project with height interpolation included. This is only 3 seconds slower than the GWA processing. The data is stored in the original CSV format, and reading them is relatively slow, but changing this behavior would require converting the data to a different format, like GeoTiff or GeoJson. The additional benefit of converting the data to a different format is not worth the additional complexity, since the data is already in a grid format and is already fast enough.

3.2.2 Verification

As seen in Figure 5, the data from the DWD is read correctly and correctly visualized by the implemented code, since all the data points match.

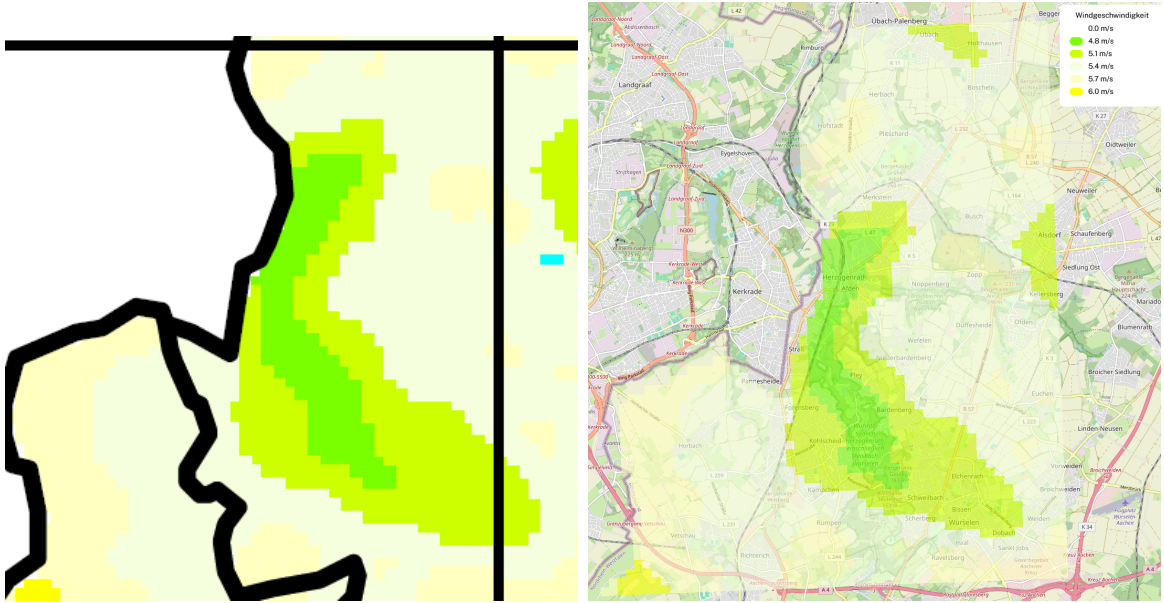


Figure 5: The maps show the mean wind speed in 80 m height. Left is a PDF of the DWD data [9] and right you can see the same area, but the data was processed and visualized by the implemented code.

3.3 Conclusion

Both implementations of the wind data processing are working and the results are correct. The Global Wind Atlas (GWA) data is faster to process, since it is stored in a GeoTiff format, which is faster to read than the CSV format used for the German weather service (DWD) data. But they are very similar in speed nonetheless with a 3-second difference.

The GWA wind measurements are not taken as often as the DWD measurements, but still often enough using a 6-hour interval. The wind measurements are more recent for the GWA data, while the DWD data only uses data from before the year 2000. The GWA data contains wind heights above 100 m, unlike DWD, which is important for wind turbines taller than 100 m. However, the DWD data includes more wind heights under 100 m, which makes the interpolation more accurate. The GWA data is less accurate as it overestimates the wind speed by about 1 m/s and has a bigger grid size than the DWD data.

Even though the GWA grid is bigger and it overestimates the wind speed a little, the GWA data is better suited to use for the potential area study since overestimation is better than underestimation in the potential area calculation, which excludes areas with low wind potential.

4 Potential Area

The potential area step calculates where wind turbines can be built according to the law and available land. The German law states that wind turbines have to be built a certain distance away from residential areas, streets, forests, and other geographic objects classes (GIS object classes). These laws vary state by state, therefore the distance regulations have to be specified for each state, as an input to the analysis. For each GIS object class a distance regulation can be specified, e.g. 1 km away from residential areas. Therefore, the potential area simulation calculates the potential areas where wind turbines can be built, based on the distance regulations and the GIS objects.

The wind potential can be considered optionally in the potential area calculation. If it is included it will exclude all areas, which are below the specified wind speed threshold, e.g. 7 m/s at 150 m height.

In the potential area calculation, pre-existing turbine placements are not considered as a default, but can be optionally included. Even if they are included in the potential area calculation, the pre-existing wind turbines will still be considered in optimal positioning, because the turbine wake effects need to be considered. The pre-existing wind turbines can be included in the potential area calculation for example, if a far enough distance from the pre-existing wind turbines is specified such that the turbine wake does not influence new wind turbines.

All GIS object classes, which were identified as relevant for the potential area calculation either from the potential area studies or otherwise, are listed in Table 2. Ideally only the GIS classes, which are specified in the regulations would be fetched, but for some data sources like the official real estate cadasters it is not possible to specify the GIS object classes before fetching. All fetched GIS objects are categorized into the GIS classes, as each data source has its own specifications.

4.1 Geographic Data

The result of the potential area simulation can only be as accurate as the underlying geographic data. After the data is fetched, it can be manually adjusted and added onto if necessary.

After the fetching of the GIS objects is done it is then categorized into one of the GIS classes. Two data sources are used to fetch the GIS objects of the project region, being the newly implemented real estate cadasters (LIKA/ALKIS) from the individual states in Germany and OpenStreetMap (OSM) as a fallback. The sources can also be mixed depending on the GIS object class. OpenStreetMap (open source version of Google) was already implemented and used as a source for the GIS objects.

Every federal state in Germany has to make the real estate data public to comply with the European INSPIRE law⁵, but evidently not all of it has to be free. The

⁵INSPIRE, URL https://de.wikipedia.org/wiki/Infrastructure_for_Spatial_Information_in_the_European_Community

official geo data is crucial, since this is the data used by the potential area studies of the individual states and therefore more accurate to the analysis.

All APIs have some sort of API limits. For OpenStreetMap's Overpass API, the limit is 10,000 requests per day or about 1 GB per day. This threshold should not be reached for a normal potential area smaller than two cities. But it will be reached for an area as big as a federal state. In this case, the data should be downloaded and hosted locally.

The official real estate cadasters have size limits of about 10,000 up to one million features (GIS objects) per query. For most requests to the official cadasters, multiple pages have to be fetched with a count parameter, which is the page number. This limit should also not be reached in the normal use-case.

Overarching category	GIS object classes	
residential	- residential - mixed residential	
other types of areas	- commercial - industrial - health resort - farmyard - retail - tourism - quarry - military	
roads	- highway - federal road - state road - county road - other road	
railway	- electrified railway - all railway	
airport	- commercial airport - all airport	
nature	- nature reserve - biosphere reserve - natural monument - national park - landmark - breeding area - special protected area birds - special area of conservation - city park	
forest	- forest mixed - forest broadleaved - forest needleleaved - natural monument	
water	- water standing - water flowing - water protection area	
power generation	- wind turbine - generator other	
other	- overhead power line - weather radar dwl - city boundary - unclassified	

Table 2: All used GIS object classes for the potential area calculation

4.1.1 Official real estate cadasters

All states except Bavaria have freely available APIs, listed in Table 3. Most APIs rely on Web Feature Service (WFS), which is a standard protocol for serving geospatial features across the web. The WFS APIs return data as the Geography Markup Language (GML), which is an XML-based format for representing geographic features. This format can easily be converted with the Python osgeo library into the GeoJSON format. The REST APIs can return data in GeoJSON format directly.

They make the following data available:

- Streets and Rivers (Flurstück)
- Buildings (Gebäude, Bauwerk): e.g. houses and factories.
- Actual use areas (Nutzung): e.g. residential areas, industrial areas, and forests.
- Administrative units (Verwaltungseinheit): e.g. districts, cities and state lines.

Fetching data from the Buildings and actual use APIs are newly implemented for all real estate data cadaster APIs, since these are the relevant classes for the potential area calculation and the rest can be fetched from OSM if necessary. Additionally, fetching data for all nature-related restrictions in Germany, such as bird protection areas, is newly implemented, which is also freely available.

The fetching itself is similar for all WFS or REST APIs. For all APIs except Schleswig-Holstein the coordinates of the bounding box can be used to request the data. For Schleswig-Holstein the data is requested by Community key, which is a unique ID for each community, like a city. Each state categorizes their real estate data differently and has different metadata, which has to be matched to the GIS object classes. Even if the categories are the same the Umlauts may be spelled differently, which is why Umlauts are substituted before categorizing, e.g. ä to ae. Each state also may use a different coordinate projection type, which is converted to the standard EPSG:4326. Each WFS API has a different page size limit, which means that the data has to be fetched in multiple requests depending on the limit.

In Bavaria, only the actual use areas are free. The paid APIs use WFS, but the free actual use API uses the Web Map Service (WMS), which is a standard protocol for serving geospatial data as images. Essentially, they return a literal picture of the data as a GeoTiff, as explained in the subsection 3.1.1, where the layers are the overarching categories residential, vegetation, streets, and water. The pixel color and pictograms on the area represent the actual use of the land. Only the pixel color categories are implemented, since extracting the pictograms is complicated. After extracting the pixel areas, they are clustered together and the concave hull is calculated to create an area polygon.

State	Data type	API type	Endpoint	API Access	Free
Brandenburg	simplified	WFS	isk.geobasis-bb.de	Open	Yes
Berlin	simplified	WFS	stadt-berlin.de/... gebäude stadt-berlin.de/... flächen	Open	Yes
Baden-Württemberg	ALKIS nora	WFS	owsproxy.lgl-bw.de	Open	Yes
Bavaria	simplified	WFS	geoservices.bayern.de	Restricted	No
Bavaria	ALKIS geotiff	WMS	geodaten.bayern.de	Open	Yes
Bremen	ALKIS nas	WFS	niedersachsen.de	Open	Yes
Bremen	ALKIS geotiff	WMS	geodienste.bremen.de	Open	Yes
Hesse	simplified	WFS	gds.hessen.de	Open	Yes
Hamburg	LIKA	REST	api.hamburg.de	Open	Yes
Mecklenburg-Western Pomerania	simplified	WFS	geodaten-mv.de	Open	Yes
Mecklenburg-Western Pomerania	ALKIS nas	WFS	geodaten-mv.de	Open	Yes
Mecklenburg-Western Pomerania	ATKIS nas	WFS	geodaten-mv.de	Open	Yes
Lower Saxony	simplified	WFS	niedersachsen.de	Open	Yes
North Rhine-Westphalia	LIKA	REST	ogc-api.nrw.de	Open	Yes
Rhineland-Palatinate	simplified	REST	geoportal.rlp.de	Open	Yes
Schleswig-Holstein	simplified	WFS	service.gdi-sh.de	Stored Queries	Yes
Saarland	LoD2	WFS	geoportal.saarland.de	Open	Yes
Saarland	LIKA	REST	geoportal.saarland.de	Restricted	No
Saarland	simplified	WFS	geoportal.saarland.de	Open	Yes
Saxony	simplified	WFS	geodienste.sachsen.de	Open	Yes
Saxony-Anhalt	simplified	WFS	sachsen-anhalt.de	Open	Yes
Thuringia	simplified	WFS	geoproxy.geoportal-th.de	Open	Yes
All nature restriction areas	simplified	WFS	geodienste.bfn.de	Open	Yes

Table 3: All real estate cadaster APIs for the individual states in Germany. The Data type ‘simplified’ stands for ‘ALKIS vereinfacht’, which is the simplified access to ALKIS.

4.2 Potential Area Simulation

4.2.1 First approach

In the following, it is explained how the pre-existing implementation of the first potential area calculation works after fixing errors in the calculation and various performance improvements.

In the first approach, the rectangular project area is divided into a square grid with a customizable grid size. This can be seen as the black points in Figure 7, such that there is a point every 20 meters, for example.

First, all the required GIS objects have to be fetched from OpenStreetMap and other sources. Then, for each GIS class, the boolean area exclusion mask is calculated, which is then combined into a global potential area mask by multiplication, which is illustrated in Figure 6. A mask is a 2-dimensional boolean array, which maps to the corresponding grid points.

All viable potential area grid points are grouped together by the clustering algorithm DBSCAN, to find out which points belong together. The concave hull of each cluster/area is found with the alpha shape algorithm and are then returned as the potential area polygons.

The option to consider wind was newly added to the pre-existing implementation. If the wind is to be considered, first the wind exclusion area is calculated.

The wind exclusion area is calculated by selecting all wind data points which are below the specified wind threshold, e.g. 7 m/s at 150 m height. Then a square is created around every data point with the point as the center. Each square is the size of the grid, e.g. $200 \times 200m$. Then all squares get merged into polygons, which are the exclusion area for the wind data.

Then the created polygons are converted into GeoJSON polygons, which are then treated as the “wind” GIS object class with a minimum distance of 0 m. This leads to the exclusion of all potential area points, which have winds below the wind threshold.

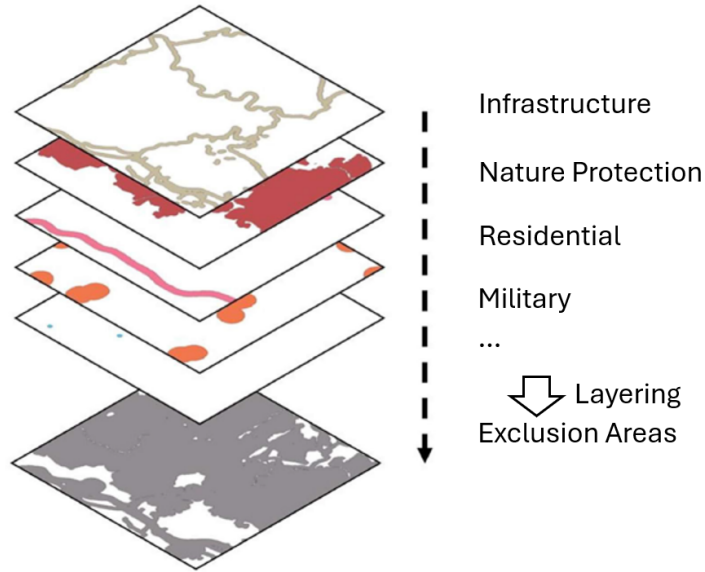


Figure 6: Overlaying different GIS class exclusion masks resulting in a potential areas layer. Figure from WINNIEPOT study [8].

Calculating the area exclusion mask:

In Figure 7 an example of an exclusion step is shown. First, the GIS object is converted into a custom coordinate system, where the unit is meter and the origin (0,0) is the lower left corner of the project area. This speeds up the distance calculations, since latitude and longitude are not directly convertible to meters.

Then, all grid points contained in the GIS object are calculated. These, in addition to the polygon hull points (corners) themselves, are the GIS object points, which are represented as the red points in Figure 7.

After all GIS object points are collected for every GIS object of the class, the distance from each GIS object point to every point in the square grid is calculated until a threshold is reached. This threshold is set to the minimum distance required by the specified distance regulation, e.g. 500 m away from residential areas.

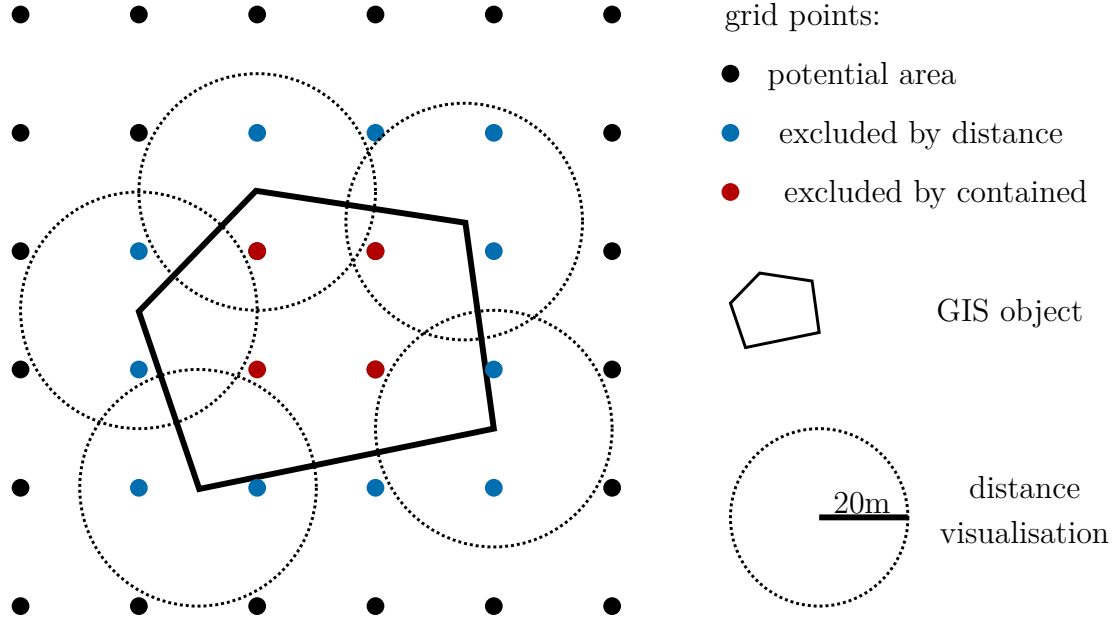


Figure 7: An example of the method to exclude points with a GIS object. The distance visualization is not drawn for the red points, because of the visual clutter.

The boolean exclusion grid for the corresponding GIS object class is achieved by comparing the distance from each grid point to the GIS object points with the minimum distance required by the specified distance regulation. The blue points in Figure 7 are excluded because they are too close to the GIS object and the red points because they are contained. The distance of the GIS object class regulation is represented visually as circles around the GIS object points.

These excluded points are removed as potential areas by multiplying the global boolean potential area grid mask with the exclusion mask of the current GIS object class. After this is done with every specified regulation and their corresponding GIS object class, the mask only contains viable points. In the example, all viable points would be all black points, which are not excluded.

4.2.2 Second approach

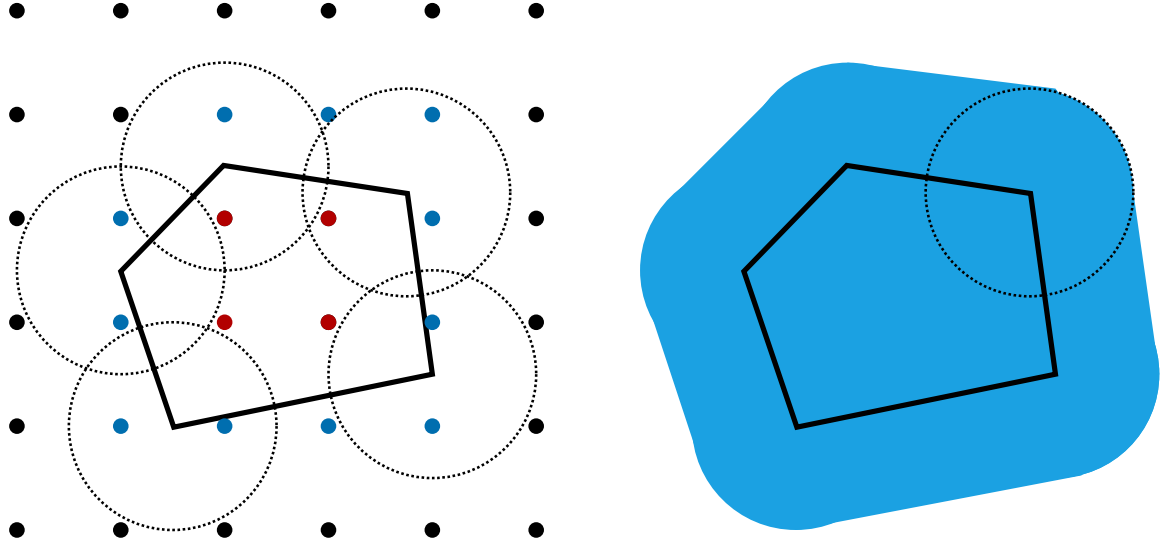


Figure 8: Potential area method comparison. The first method using the grid is on the left. The buffer operation of the newly implemented second method is on the right.

The second newly implemented approach, which proved to be as precise as possible and faster in every realistic case, is to buffer and merge all GIS objects together directly, instead of using a static grid.

The direct union approach is analogue to the first approach, with the difference of using the geometry library shapely, instead of doing it manually. Merging the objects directly does not result in any precision loss. The first approach is in principle the same as the shapely buffer operation, but in Python and with less accuracy, compared to shapely which uses the faster C++. A comparison between the two approaches is made in Figure 8.

The pseudocode for the new second approach is depicted in Listing 1. It is not the actual code, since it does not contain the parsing and the fixing of the GIS objects from the specific data sources, but it is otherwise similar to the actual implementation. All the major operations are done in the shapely library.

In the new approach, first, the GIS data is converted to the custom coordinate system with the unit in meters, just like the first approach does, which allows applying the minimum distances directly without conversion. Then all GIS objects are parsed into shapely shapes, and a buffer is applied with the buffer size of the corresponding distance regulation. Next, every expanded GIS object is merged together into one big MultiPolygon.

Since this MultiPolygon states where turbines cannot be built according to the distance regulations, the polygon needs to be inverted to get the resulting potential areas. The MultiPolygon is inverted by taking the difference of the rectangular project

```

from shapely import union_all, box, shape
def get_potential_areas(transformed_gis_objects, regulation, wind_data=None):
    merged_gis_objects = merged_wind_squares = []
    for gis_object in transformed_gis_objects:
        min_distance = regulation.get(gis_object.object_class, 0)
        # minimum buffer for 1D shapes
        if gis_object.geometry["type"] in ["Point", "LineString"]:
            min_distance = max(min_distance, 1)
        polygon = shape(gis_object.geometry)
        merged_gis_objects.append(polygon.buffer(min_distance))
    merged_gis_objects = union_all(merged_gis_objects)
    bounding_box = box(minx, miny, maxx, maxy)
    if wind_data is not None:
        # create a square for each position
        for p in positions: # positions < 7 m/s
            sqm = 100 # 100 m example number for square size in meters
            square = box(p[0] - sqm, p[1] - sqm, p[0] + sqm, p[1] + sqm)
            merged_wind_squares.append(square)
        wind_exclusion_area = union_all(merged_wind_squares)
        bounding_box = bounding_box - wind_exclusion_area
    difference = bounding_box - merged_gis_objects
    return difference

```

Listing 1: Pseudocode for the second approach of the potential area calculation.

area and the MultiPolygon. The resulting potential area MultiPolygon can be directly viewed or converted back to latitude and longitude.

If the wind should be considered in the potential area calculation, the calculated wind exclusion area is subtracted from the bounding box of the project area, just like the merged GIS objects before.

Just as for the first approach the wind exclusion area is calculated by selecting all wind data points which are below the specified wind threshold and creating a square around every data point with the point as the center. The difference here is that the MultiPolygon of the merged squares, which is the exclusion area, is directly subtracted from the bounding box of the project area.

If a GIS object is a line or a point and there is no distance regulation given or a 0 m distance, then a buffer of 1 m is applied, since these one-dimensional shapes have no area and otherwise would get ignored at the inverting step of the MultiPolygon. The 1 m buffer operation converts them into two-dimensional polygons with an area, which is visualized in Figure 9. This guarantees an area of about $3.14m^2$ for points, for example.

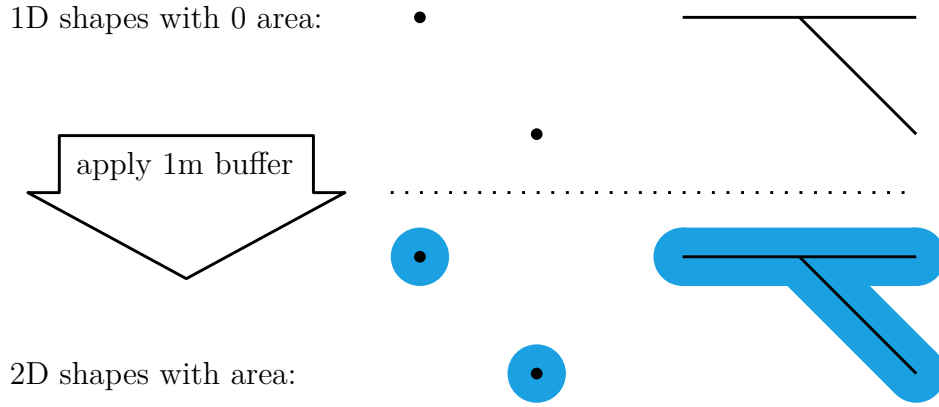


Figure 9: Example for buffering 1D shapes into 2D shapes.

4.2.3 Comparison of precision

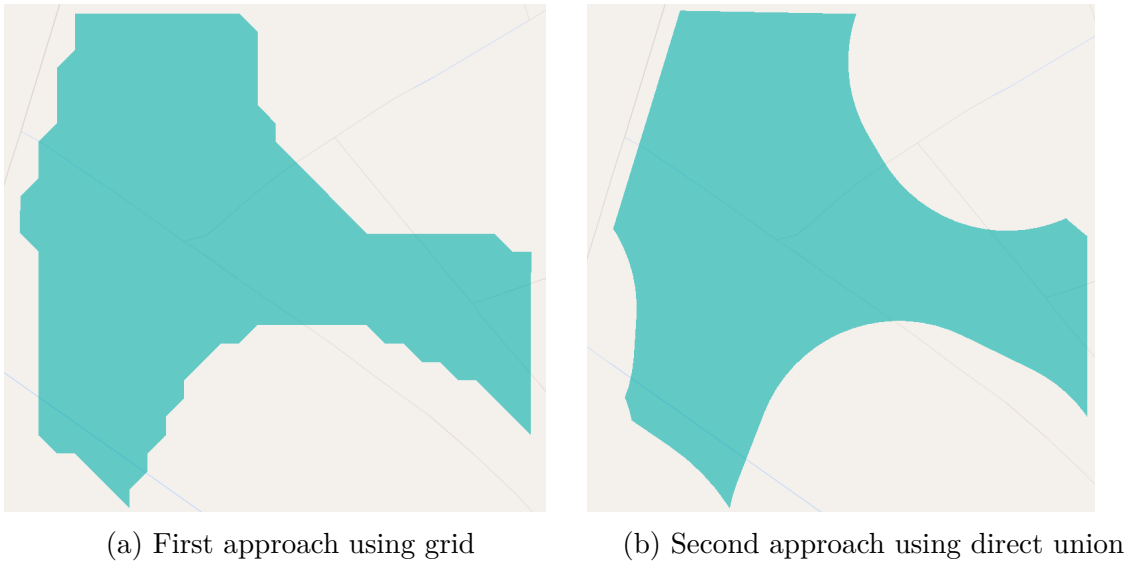


Figure 10: Same area for both approaches to compare the precision.

The second approach is as accurate as the GIS data allows.

For the first grid-based approach, the resulting potential area is bound to the grid points, which are already inherently less precise. The area that goes over the points, but not far enough to the next, is discarded and results in raggy edges, as compared in Figure 10. Therefore, the approach is as precise as the size of the grid.

The first approach only uses contained points and corner points of the GIS object, to calculate the distances. The first approach does not consider the edges of the GIS object, which makes the potential area less precise, since it might miss grid points along the edges.

Additionally, holes in the potential area will either be disregarded or they will split the area in two at the clustering step. This happens because the found potential grid points get clustered together and its concave hull is calculated, which cannot have holes inside them. So either the hole is disregarded entirely by the concave hull or the hole splits the potential area into two separate areas, because they get grouped into different clusters.

The potential area study in Lower Saxony also uses a grid-based approach with a $25 \times 25m$ grid [8]. However, the additional accuracy of the second approach is not relevant if only the potential area grid points and not the hull is used for further calculations. For example in the optimal positioning of turbines, the potential area grid points are used as the potential turbine locations. Therefore, calculating the grid points was added to the first approach as well, which can be done very fast since the potential areas are small in comparison to the whole project area.

4.3 Correctness Errors of the First Approach

There were multiple errors in the original implementation, which are explained in the following.

```
hull_index = None
# find out which polygon is the outer polygon
for i in range(0, len(shapely_polygons)):
    next_index = (i + 1) % len(shapely_polygons)
    if contains(shapely_polygons[i], shapely_polygons[next_index]):
        hull_index = i
        break
if hull_index is not None:
    # create shapely polygon with outer hull and inner holes
    outer_hull = shapely_polygons[hull_index]
    del shapely_polygons[hull_index]
    polygon = Polygon(outer_hull, holes=shapely_polygons)
else:
    polygon = MultiPolygon(shapely_polygons)
```

Listing 2: Finding hull of MultiPolygon

GIS objects, which are MultiPolygons were not correctly considered and therefore ignored. MultiPolygons are polygons with holes inside or a collection of polygons in a GeoJSON. The problem is that the definition of a MultiPolygon is not the same between GeoJSON and shapely. In shapely, a MultiPolygon is a collection of polygons, which also can have holes.

Therefore, it has to be checked if a MultiPolygon consists of multiple connected polygons or if it is a polygon with holes, when it is fetched. In the fixed implementation it is checked which polygon is the outer hull with the shapely contains method. If

there is no outer hull, it is assumed that it is a collection of polygons and then every contained polygon is treated as an individual polygon. The algorithm for this process can be found in Listing 2.

If a fetched polygon is invalid, the `make_valid` function of `shapely` is newly applied to try to fix the errors contained in the Polygon. For instance, a polygon would be considered invalid if the first and the last point of the polygon is not in the exact same location.

If the distance for a regulation was set smaller than the square size or to 0, the GIS objects class was entirely ignored. The reason was that the circles were too small to contain any grid points, which is visualized by the blue points in Figure 11. This results in not considering the GIS object class at all. For example, if every point in the distance 0 m should be excluded, the circles would be 0 in size, which would not contain any grid points.

The solution for this is, that for every distance smaller than the square size the minimum distance of the square size is used, which is the minimum distance between two points in the grid. This makes it, so only the grid points are excluded, which are too close or inside the GIS object.

If the GIS objects are too small or misaligned to contain any grid points or have an invalid geometry, they were also ignored. As shown in Figure 11 no grid points are contained in the GIS object, because the GIS object is too small. Therefore, in addition to the contained grid points, the polygon hull points (corners) are always considered, which are the red points in the figure, which would exclude the blue grid points. This also leads to a higher accuracy, since the GIS object hull points are more accurate than the grid points.

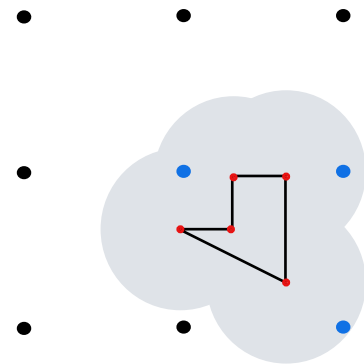


Figure 11: Small GIS object example.

The clustering algorithm DBSCAN is used to group the found potential area grid points together, of which the hull forms a potential area polygon. This has the side effect that a hole splits the polygon into two or the hole might get disregarded entirely, instead of creating holes in the polygon. The second approach does not have this issue.

The “`eps`” setting of the clusterer is the maximum distance between two samples in order to be considered neighbours and ultimately being grouped together. The clustering `eps` setting is reduced from 1.5 times the square size to 1.414 times the square size. Setting it to 1.414 times the square size is a better approximation of the square root of a square, which means now only the diagonal of the square size is considered as ‘neighbours’. The lower precision setting caused that more potential areas were grouped together, than intended. For example, a street running through a potential area should split the potential area into two separate areas. Previously, because the `eps` setting was too high, points beyond the streets were considered as

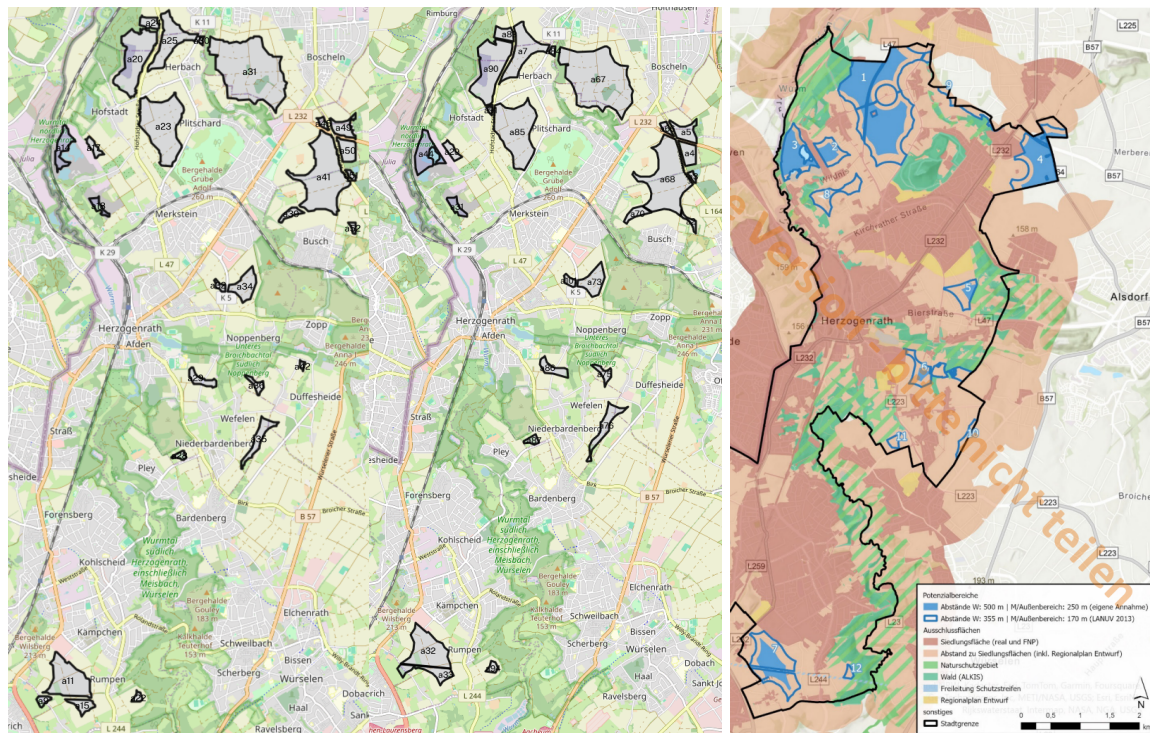
¹`make_valid`, URL https://shapely.readthedocs.io/en/latest/reference/shapely.make_valid.html

‘neighbors’ and therefore both areas are considered as one potential area, instead of two separate areas.

4.4 Verification

4.4.1 Potential areas by distance regulations

The Herzogenrath report for the potential areas is going to be compared to the first and second approach of the potential area implementation. The GIS objects and the map for the potential area simulation were fetched from OpenStreetMap and the potential area was calculated with the distance regulations from the Herzogenrath report. The distance regulation for mixed residential areas was not used, since OSM does not have this category. Therefore, the distances to mixed residential areas are the same as to regular residential areas.



(a) First approach with cus- (b) Second approach with (c) Herzogenrath Report
tom grid buffer and union results (marked in blue)

Figure 12: The potential area simulation with data from OpenStreetMap compared to the Herzogenrath potential area report [3]. Only areas with a square meter size higher than 10000 are shown.

The three comparable results of the Herzogenrath project area are shown in Figure 12.

The potential area results of the Herzogenrath report (c) and the potential area from both calculations are the same. It is clear the calculation is correct, since the

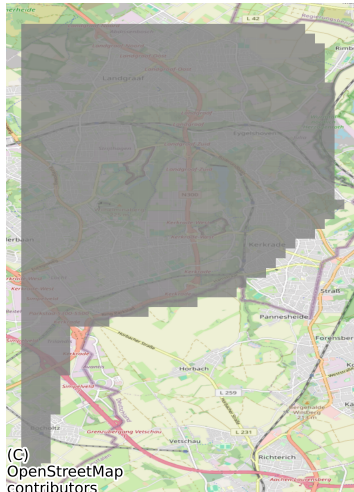
areas are nearly identical, they only deviate from the Herzogenrath report when the GIS objects are not the same. An example is the big top left area a23 (first approach) where the lower left corner is larger, because the mixed residential area is misclassified as a regular one, which has a higher distance regulation.

The second (union) approach, which can be seen on the right in Figure 12, achieves the same result as the first approach, but with more accurate polygons, since there is less accuracy loss. The top leftmost area is missing the hole for the lake, because it split the area in two and the areas below 10000 m² are not shown.

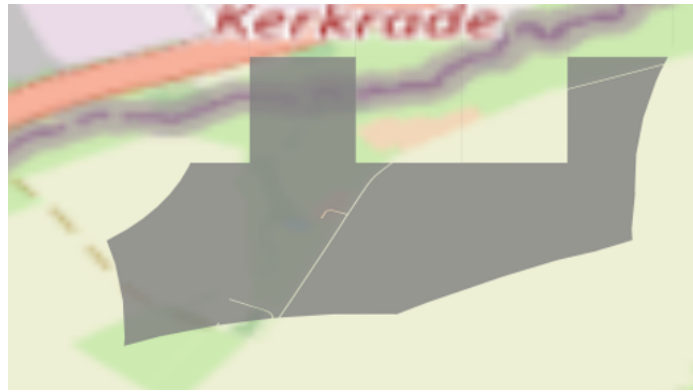
This is why I conclude that both potential area calculations are correct, but the first approach is less accurate than the second.

The result directly depends on the underlying GIS objects, which are fetched from OpenStreetMap and other sources such as the real estate cadaster, which makes them hard to compare to other results with different data. That the two approaches achieve the same result underline their correctness.

4.4.2 Potential areas by wind exclusion



(a) Wind exclusion area



(b) Example Potential area considering wind.

Figure 13: Wind exclusion area showing winds lower than 7 m/s from GWA.

In Figure 13 the wind exclusion area is shown on the left. The grey area is showing all winds lower than 7 m/s at 150 m height. On the right an example potential area is shown, where the wind exclusion area is subtracted from merged GIS object. Since the wind potential is not as precise the resulting potential area, the hull looks like steps and not like a smooth area, as illustrated on the top edge of the example. This is because of the squares made from the wind data points.

The wind exclusion area is calculated correctly and the resulting potential areas are also correct, although if the wind is considered they might result in areas smaller than necessary because of the resolution of the wind data.

4.5 Optimizing Performance of the First Approach

The performance of the first simulation was not satisfactory, since the simulation took about 2 minutes for the small test project Herzogenrath. The profiling from Figure 35 showed that calculating the minimum distances (`get_mindistances_by_subdivision`) took 119 seconds and reading the written file took 11.1 seconds, as visible in Figure 34. Originally, the distances from every grid point to the GIS object points were calculated and saved to a file, which was about 165MB for Herzogenrath. In comparison, the file size for the raw GIS objects is only about 50MB. Since all distances were precomputed and stored for the potential area simulation, the potential areas can be recalculated rapidly, especially when the distance regulations change. The most time is spent calculating the distances between each point of the grid to each GIS object class, and saving to the file `gis_data_for_pa`, which turned out to be slower than calculating it on the fly.

4.5.1 Optimising `get_minimum_distances`

This function calculates the distances from the grid points to the GIS object points. Instead of calculating the distance of every point of the grid to every GIS object, it is only necessary to calculate the distance until the minimum distance threshold away from the GIS object is reached. Because the distance regulations aim to exclude points that are too close to the GIS object, it is possible to stop calculating the distances if the minimum distance is reached. This was achieved by introducing an upper bound that is set to the minimum distance of the distance regulation. However, this means that the distances to each grid point cannot be precomputed. They therefore have to be computed every time in the potential area simulation.

Enabling multithreading on the KDtree⁶, which is used to calculate the distances of each neighboring point, significantly improved performance.

The function runtime is now less than half a second (0.452s), compared to the 31.3s before. The profiling is shown in Figure 39 and Figure 36. The potential area calculation is now about half a second slower than before, because the distances are not precomputed like before. Now the fetched GIS objects are entirely independent of the potential area simulation, which means that the GIS objects can be changed without having to recalculate the distances.

4.5.2 Optimising `find_points_in_geometries`

This is the function to find the grid points contained in the GIS objects. For our grid a custom coordinate system is used, which is the distance in meters to the origin point, instead of EPSG:4326 coordinates, because distances can be calculated more efficiently in this format and there may be different coordinate types for different sources. Instead of converting the custom grid to a EPSG:4326 based grid, which was

⁶KDtree, URL <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html>

done before, converting the GIS objects to the custom coordinate system is faster, because they consist of fewer points than the grid.

The find points in geometries function originally created a GeoDataFrame¹ of the GIS object and a GeoDataFrame¹ of the square grid and then joined them together. Just creating the GeoDataFrames took about 20 seconds, and can even run out of memory for bigger projects, as it is very memory-intensive. Instead of creating a GeoDataFrame¹ of the grid, now a STRTree² of the square grid is created, which is a spatial index for two-dimensional spatial data, which is ideal for our grid. The STRTree is queried with each GIS object, which returns all the grid points contained inside the GIS object.

Unlike the GeoDataFrame¹ that had to be created for each GIS object class, the STRTree² is only created once for the entire grid, which speeds up the calculation significantly. The node_capacity for the STRTree, which is the amount of branches of the tree, is set to the length of the square grid, which further speeds up the calculation.

After the performance improvements, the function takes 6.63s compared to 74.2s that it took before (see Figure 38, 35).

4.5.3 connect_points

This function finds additional points for line GIS objects (like streets), if the points are too far apart. This is necessary, because otherwise the concave hull algorithm would not work correctly and lines would not be considered correctly.

The number of interpolated points was 5 points plus the distance of the coordinates divided by the square size. This is reduced to one point instead of 5 since more turned out to be unnecessary. This means that now for two points, which are about the distance of the square size apart, two additional points are generated (it is rounded up). Before optimization, it was seven additional points.

The new runtime of the function is about 4 times faster. It now takes 2.83s, which took 10.3s before (see Figure 38, 37).

4.6 General

Instead of using the Python built-in math functions, every inbuilt function was replaced with the NumPy⁷ equivalent, which yielded more than 2 seconds of improvement, because it is implemented in the more performant lower level programming language C. For example, np.array and np.logical_and replaces '[' and '*'. Non in-place array operations were replaced with faster in-place array operations, e.g. 'extend' instead of '+='. Some redundant calculations were removed, like instead of calculating the custom square grid for each GIS object class, to only calculate it once for all classes.

¹GeoDataFrame, URL <https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoDataFrame.html>

²STRTree, URL <https://shapely.readthedocs.io/en/2.0.6/strtree.html>

⁷NumPy, URL <https://numpy.org/>

4.7 Performance

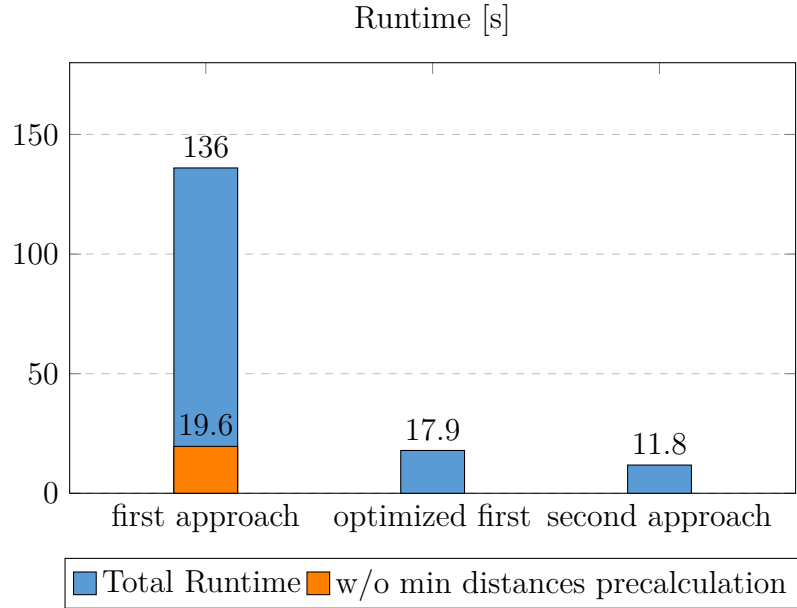


Figure 14: Runtime comparison between all results.

4.7.1 Final performance of the first approach

Calculating the potential areas, including calculating all minimum distances took 136 seconds as illustrated in Figure 14. After performing all performance improvements and correctness fixes, the total runtime takes 17.9s.

Even if the calculation of the minimum distances is excluded (only calculate the potential areas itself), the total runtime is still faster, since it was 19.6s. This is because reading and parsing the big minimum distances file ($\sim 150\text{MB}$ for Herzogenrath) takes longer than calculating it on the fly with the improved minimum distance calculation.

4.7.2 Performance of the second approach

Calculating the potential areas with the second approach takes 7.14s, as evident in Figure 40. The total time is 11.8s, which includes reading in the GIS objects from file, as shown in Figure 14.

The only functions left that use a lot of time are the library functions, so there is little improvement potential without coming up with an entirely different approach.

The reordering to first merge the GIS objects and then apply the buffer is significantly slower, since the GIS objects are many small polygons that consist of many individual points. Applying the buffer first saves a lot of calculation time when merging the GIS objects, since the buffering reduces the number of points significantly. This also means that the smaller the distances of the distance regulations are the slower this approach runs and the bigger the distances are the slower the first approach runs.

When all distances are set to 0 meters, the second approach is slower than the first approach. In this case the first approach takes 16.1s, compared to the second approach which takes 19.2s, as evident in Figure 42.

However, when all distances are set to 20 meters instead, then the second approach is already faster than the first approach. The first approach always uses 20 m distance as a minimum, because of its grid, so this is a fairer comparison. In that case the second approach takes 15.5s, compared to the 16.1s of the first approach, as evident in Figure 41.

The distance regulations are usually higher rather than lower. Most importantly for residential areas, which have the most amount of tiny GIS objects, the distance is generally about 1000 meters in Germany. Therefore, in all realistic cases, the second approach is faster than the first approach.

4.7.3 Conclusion

The first pre-existing grid-based approach of the potential area simulation was improved significantly in terms of performance and correctness. The performance was improved from about 2 minutes to about 18 seconds for the small test project Herzogenrath. The correctness of the first approach was improved by fixing multiple bugs, which were caused by the grid based approach.

The newly implemented direct union second approach only has a runtime of 7.14s, compared to the first approach, which takes 14.8s without reading the GIS objects from file. It is double as fast not considering reading the GIS data. The smaller the minimum distances of the distance regulations are, the slower the second approach performs, since more calculations are necessary to merge the GIS objects, but it is still faster than the first approach in every realistic case.

The second approach is as accurate as the GIS objects allow, since it uses the shapely library to merge the GIS objects directly. In contrast, the first approach is less precise using a static grid, but the added accuracy might not be required if the result is only further processed with the same grid.

Both approaches are verified to give correct results, depending on the underlying GIS objects. In conclusion, since the newly implemented second approach is better in performance and accuracy, it should be used in the future.

5 Optimal Positioning

The optimal positioning simulation optimizes the wind turbine placements according to its objective. It is calculated for the potential areas, which were calculated in the previous step. These areas can also be modified or excluded manually by the user, if they are not suitable for wind turbines. The existing turbine placements have to be considered in the optimal positioning, since they were previously ignored in the potential area calculation. The optimal positioning is tested for the potential areas of

the Herzogenrath project. The potential areas that are too close together have to be grouped together, because the turbine wakes can influence each other.

5.1 Optimization Goals

The objectives, which are relevant for the placement of wind turbines, are:

- AEP (Maximizes Annual Energy Production)
- LCOE (Minimizes Levelized Cost of Energy)
- NUMBER_OF_TURBINES (Maximizes turbine amount)
- SHADOW (Minimizes shadow flicker)
- NOISE (Minimizes generated noise)
- RESIDENTIAL_DISTANCE (Maximizes distance to residential areas)

Multiple objectives can be weighted in ratios against each other and optimized simultaneously, but the more objectives are taken into account, the longer the evaluation takes making the calculation slower.

The most important objectives are to maximize the annual energy production (AEP), maximize the number of turbines, and minimize the Levelized Cost of Energy (LCOE). The LCOE is the average electricity generation cost over its lifetime, so the minimum electricity cost to break even. It includes costs such as purchasing and installing the turbines, street connection, and the cabling costs of connecting each turbine to the power grid. It also includes interest rate and other costs such as maintenance and insurance. The biggest factor in the LCOE is the turbine cost and the cabling costs, which are directly related to the distance between the turbines.

Other objectives include minimizing the shadow flicker effect of turbines, the noise generated, and the distance to residential areas to reduce the impact on nearby residences, which can oppose the construction of the project.

Additionally, the chosen turbine model must also be considered and optimized, since different models have different power curves and heights, which can be utilized differently in the terrain and winds of the location.

5.2 Optimization Algorithms

There are different optimizing strategies and algorithms such as Greedy algorithms, simulated annealing, Mixed Integer Programming (MIP), machine learning approaches like a genetic approach, brute force and a combination of these methods. The algorithm is chosen based on the required speed and accuracy, since these two factors are inversely related.

The Greedy algorithm is the fastest but not the most optimal, since it prioritizes to make locally optimal choices, instead of exploring the solution space more thoroughly for a global optima.

Brute forcing all possible combinations is the most accurate, but also the slowest, since it has to check all possible combinations. Just like brute force, Mixed Integer Programming is also very accurate and reaches a nearly optimal solution, but it is faster than brute force.

The machine learning approaches like a **genetic** approach are excluded, since the results should be fast.

Simulated annealing is a good compromise between speed and accuracy, but it is not as fast as a greedy algorithm, since it explores the solution space more thoroughly and tries to find the global optimum by avoiding local optima.

If a more optimal solution is aimed for, the algorithm can be changed to a genetic approach or even to Mixed Integer Programming, depending on the required precision, but the simulation time will increase significantly with increased precision. The equivalent Mixed Integer Programming (MIP) implementation of the greedy approach reached a similar solution, but it took more than 30 times longer to compute.

5.2.1 Wind data

For the site planning, the wind data is essential and the more accurate the data, the better the results. A 10% error in the mean wind speed can lead to a 30% error in the resulting power production calculation [5]. A particularly rough terrain and large obstacles can lead to a significant deviation in the wind data [5]. Ideally, data from existing wind turbines near the site should be used to determine a reliable value and if that is not available a wind yield study has to be conducted, where surveyors may even set up measuring masts [5]. The mean wind speeds in section 3 have a rough resolution and are not accurate enough for a precise simulation. They are only sufficient for excluding areas with low mean wind speeds. With this lacking data, the simulation can only give a rough estimate of the potential power production. It is still possible to achieve an accurate optimal positioning, because the wind distribution and wind direction are not affected by the imprecision of the average wind speed.

5.2.2 Chosen algorithm

The aim is to get a good estimation of the potential power production in the shortest time possible. Two optimization algorithms are chosen and compared. A greedy and a simulated annealing algorithm are used in combination with heuristics to find a good solution in a fast time. The goal of the optimization is to maximize the total number of wind turbines, while minimizing the mutual wake effects within the wind farm. This means that the greedy ellipse algorithm can only be used to optimize Annual Energy Production (AEP) and not any other objective, because the heuristics are specifically designed for this objective, unlike the pre-existing greedy attractiveness algorithm [13], which can be used for most objectives. This is the most common objective for wind farm optimization and therefore the most important one.

The following heuristic is used to simplify the problem. As a rule of thumb a distance of 4–5 times the rotor diameter in the main wind direction and 3 times the

rotor diameter in the cross wind direction is a good distance to have between the turbines to minimize the turbine wake effects [5].

The federal wind potential study uses a distance of 4 rotor diameters of the wind turbines to each other [7]. This can be implemented by placing circles in the area. Placing circles is easier and faster to calculate than ellipses, but the circles are less optimal, since the distance between the turbines is smaller. Therefore, ellipses with a 3:5 ratio were chosen based on the rule of thumb.

Unlike the federal wind potential study the ellipses will not be placed greedily in the best wind position, but they are placed to reduce turbine wakes. The greedy algorithm works by placing ellipses in the potential area until it is filled, which is a good estimation of an optimal positioning that is weighing the AEP with the mean farm efficiency. The mutual interference between individual wind turbines is unavoidable, but it is acceptable to a certain extent. Therefore, the aim of optimizing the positioning is to maximize the mean efficiency to about 90% of the turbines [5].

Since the focus is on the performance of the algorithm, the algorithm will only optimize for one turbine type in the wind farm and compare the results of different turbine types in the end, even though the results of mixed layouts can reach a higher power output.

After all turbines are placed, the ellipses are expanded as much as possible to increase the distance between the turbines, which minimizes the turbine wake effects and maximizes the power production. Although that maximizes the AEP, this also increases the LCOE, since the cabling cost increases with the distance between the turbines.

The annual energy production and Levelized Cost of Energy (LCOE) will be calculated by the pre-existing calculation [13], which takes turbine wake effects, wind speeds and heights into account.

Instead of finding the maximum turbine amount automatically, it is also possible to specify the turbine amount to be placed, but if a higher turbine amount is specified than the ellipse size allows, the ellipse size needs to be configured smaller to allow for the placement of more turbines.

5.3 Formal Problem

This is the formal problem for the objective of maximizing the turbine amount, combined with the mentioned heuristics. The formal problem can be directly solved by a mixed integer programming solver.

The input to the problem is the turbine model and the alignment direction. The objective is to maximize the number of turbines, while keeping the turbines as far apart as possible.

The constraint states, that only one of two turbine locations can be chosen if location 2 is contained in the ellipse of location 1, which is calculated with the standard ellipse formula in equation 3. The locations are rotated according to the `alignment_direction`, which is the wind direction.

Objective function:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i \in \text{locations}} \text{Turbine}_i \\ \text{Turbine}_i \in & \{0, 1\} \end{aligned} \quad (1)$$

Constraints:

$$\text{Turbine}_{loc1} + \text{Turbine}_{loc2} \leq 1 \quad \forall loc1, loc2 \in \text{locations} \quad (2)$$

$$\text{if } \left(\frac{\text{rotated_dx}}{a_{\text{ellipse}}} \right)^2 + \left(\frac{\text{rotated_dy}}{b_{\text{ellipse}}} \right)^2 \leq 1 \quad (3)$$

where:

$$\text{rotated_dx} = dx \cdot \cos(\text{ad_rad}) + dy \cdot \sin(\text{ad_rad})$$

$$\text{rotated_dy} = -dx \cdot \sin(\text{ad_rad}) + dy \cdot \cos(\text{ad_rad})$$

$$dx = loc1[0] - loc2[0]; dy = loc2[1] - loc1[1]$$

Parameters:

a_{ellipse} : Semi-major axis of the ellipse, which is 5 times the rotor diameter by default.

b_{ellipse} : Semi-minor axis of the ellipse, which is 3 times the rotor diameter by default.

ad_rad : Alignment direction in radians

locations : All points in (lat, lon).

5.4 Grouping Areas Together

The potential area calculation leads to areas, which are separate, but might be too close together that the turbine wakes can influence each other if they are treated as separate areas. This can happen in areas divided by streets for example. This is why the areas need to be grouped together to reach a valid result.

This can be done either by manually specifying them in the optimization case or by grouping them together automatically. In the visual frontend, the areas can be edited and selected manually easily, or if none are selected, the areas are grouped together automatically.

In Listing 3 the areas are clustered together if the distance between the boundary box of the areas is smaller than the distance of the rotor diameter times the major-ellipse axis. This only applies to the ellipse algorithm, since other algorithms might need a higher threshold.

```

all_clusters = seenIds = []
distance = a_ellipse * rotor_diameter_m
while len(seenIds) < len(all_potential_areas):
    grid_points = area_cluster_ids = []
    left = right = top = bottom = None
    for area in all_potential_areas.items():
        if area[0] not in seenIds:
            grid_points = area[1].squares
            area_left = min(point[0] for point in grid_points)
            area_right = max(point[0] for point in grid_points)
            area_top = max(point[1] for point in grid_points)
            area_bottom = min(point[1] for point in grid_points)
            if len(area_cluster_ids) == 0:
                area_cluster_ids.append(area[0])
                seenIds.append(area[0])
                left, right = area_left, area_right
                top, bottom = area_top, area_bottom
            else:
                middle = ((area_left + area_right)/2, (area_top + area_bottom)/2)
                if (left - distance < middle[0] < right + distance
                    and top + distance > middle[1] > bottom - distance):
                    grid_points += area[1].squares
                    area_cluster_ids.append(area[0])
                    seenIds.append(area[0])
                    left, right = min(left, area_left), max(right, area_right)
                    top, bottom = max(top, area_top), min(bottom, area_bottom)
    all_clusters.append({"ids": area_cluster_ids, "squares": grid_points})

```

Listing 3: Clustering areas together, which are too close to each other

5.5 Greedy Ellipse Implementation

Since a potential area is not discrete, but continuous, the potential area is divided into a grid, where each grid point is a potential position for a wind turbine. Here the variable grid from the potential area calculation is reused ($20 \times 20m$ grid by default). The optimization objective is not only to maximize the turbine amount but also to combine it with the best alignment direction.

The optimal positioning algorithm is shown in Listing 4. Every ellipse is rotated into the same direction, which aligns the entire wind farm to that direction. There needs to be different alignments, because aligning the wind farm to the wind direction increases the spacing between the placements, which results in a reduction of wake effects and maximizes the power production. The maximum turbine amount is calculated for all alignments within ± 30 degrees from the main wind direction in 5 degree steps.

Everything beyond 30 degrees from the main wind direction turned out to be less optimal and therefore not necessary to be calculated.

In reality, the wind direction is not constant, but changes over time. Hence, the real turbines always face the current wind direction, but this is taken into account by the smaller width of the ellipse in the crosswind direction.

After the max turbine amounts are calculated for all directions, the alignment direction with the most wind turbines is chosen greedily. If the turbine amounts are the same, the alignment direction closest to the main wind direction is chosen. The alignment direction with the highest Annual Energy Production (AEP) is not chosen here since it is not necessarily the alignment direction with the highest final AEP, since the AEP needs to be calculated before the individual expansion of the ellipses, because the expansion takes the majority of the runtime.

After the best alignment direction is found, the best turbine placement needs to be calculated. This is done by increasing the ellipse size to the maximum possible size. This is achieved in two steps. First the global ellipse size is maximized and then the individual ellipse sizes are maximized. The individual ellipse sizes cannot be directly maximized, because the ellipses are placed one after another.

In the first step, the greedy algorithm in combination with a binary search is used to find the biggest global ellipse size for all turbines. Then another binary search is used to maximize the individual ellipse size for every turbine, considering the biggest global ellipse size as the new minimum.

This results in the layout with the turbines being placed as far apart as possible, while keeping the turbine amount the same. In the end, the result for each turbine model is compared and the layout with the highest AEP is returned.

A greedy algorithm is applied to place the turbines, which works as follows: The grid points are sorted, depending on the current alignment direction (main wind direction ± 30). For example, if the wind comes from the top right, the top right most grid point would come first, then the next top right most grid point and so on.

The first wind turbine is placed in the first available point. Then the points, which are contained in the ellipse around the first wind turbine, are removed from the grid, by the `reduce_grid` function. The next wind turbine is placed in the first available point and so on, until no more grid points are available. This results in achieving the maximum amount of turbines that can be placed in the potential area, with the given ellipse size.

If existing turbines are considered, all points contained in the ellipses of the existing turbines are removed from the grid points by the `reduce_grid` function. This has to be done depending on the alignment direction. Therefore, the existing turbines are basically just the first turbines to be placed each time.

```

mWD = self._main_wind_direction
sort_right = -1 if mWD < 90 or mWD > 270 else 1
sort_top = -1 if mWD < 180 else 1
self.available_grid_points_sorted = sorted(self._all_grid_points_m,
      key=lambda item: (sort_top * item[1][1], sort_right * item[1][0]))
alignment_directions = np.arange((mWD - 30) % 360, (mWD + 31) % 360, 5)
# find the best wind direction for the most turbines
farm_res = [
    self._get_max_number_of_turbines(ad) for ad in alignment_directions]
directions_with_results = list(zip(alignment_directions, farm_res)).sort(
    key=lambda x: (x[1][0], -abs(x[0] - self._main_wind_direction)))
self._optimal_wind_direction = directions_with_results[-1][0]
turbine_count = directions_with_results[-1][1][0]
res_farm = directions_with_results[-1][1][1]
# double the size is impossible as another turbine would fit
lower, upper = 1, 2
# maximize average ellipse size
if turbine_count > 1:
    res_farm, lower = self._optimize_single_alignment_direction(
        self._optimal_wind_direction, turbine_count, lower, upper)
aep_res = self._get_AEP_mwh(res_farm)
# maximize individual ellipse size
pivots_res = None
if turbine_count > 2:
    upper_farm, pivots_res = self._expand_individual_ellipse_size(
        ad, tt, lower, turbine_count, res_farm)
    aep_lower = self._get_AEP_mwh(upper_farm)
    if aep_lower > aep_res:
        res_farm = upper_farm

```

Listing 4: Optimal positioning algorithm shortened

5.6 Maximizing Ellipse Sizes

5.6.1 Binary search for global maximum ellipse size

The algorithm to find the global maximum ellipse size is shown in listing 5. First an upper and a lower bound for the ellipse size have to be set. The lower bound is obviously one times the ellipse size and the upper bound cannot be over double the size, since otherwise another turbine would fit in the area, which leads to a wrong result. Therefore, the upper bound is set to two.

Then a binary search is performed, which approaches the solution by halving the search space every iteration. It consists of a while loop that is run until a precision threshold is reached, which is 5 m by default. In the loop the pivot is set to (*lower* +

```

def _optimize_single_alignment_direction(
    self, alignment_direction_deg: float, turbine_type: tuple[str, float],
    turbine_count: int, lower: float, upper: float):
    lower_farm = None
    while upper - lower > self._precision:
        pivot = (lower + upper) / 2
        turbine_placements = self._place_all_turbines(
            alignment_direction_deg, pivot, turbine_type, turbine_count)
        if turbine_placements:
            lower = pivot
            lower_farm = turbine_placements
        else:
            upper = pivot
    return lower_farm, lower

```

Listing 5: Expanding the global ellipse size to its maximum

$upper)/2$, which is the test for the middle value in the binary search. Then all turbines are tried to be placed with the new ellipse size. If a correct placement was found, the lower bound is raised to the current pivot. Otherwise, the upper bound is set to the pivot.

5.6.2 Binary search to maximize individual ellipse size

The algorithm to expand the individual ellipse sizes is shown in listing 6. After the global maximum ellipse size is found, the same strategy is used to find the best individual ellipse size. In order to change the individual ellipse sizes a corresponding list of pivot values is used instead of a single pivot value.

The individual increase in ellipse size can lead to a layout with a lower AEP than previously, because the ellipses might get pushed more behind each other rather than being offset. This is why the new layout is only accepted if the AEP is higher than before. Otherwise, the previous result is kept.

Since the binary search has to be performed for each turbine, this takes a lot of time calculating. To speed up the calculation the following heuristics are applied.

The lower bound is set to the newly found global ellipse size and the upper bound is set to two times the lower bound size. Often multiple ellipses cannot be expanded further, since the global ellipse size was already maximized. For example when ellipses are grouped densely together in the middle of an area. Therefore, a test is performed before the binary search is run to test if the current ellipse can be expanded further. The test is done by applying the precision threshold to the lower bound.

There is a difference using the `place_all_turbines` function when using individual ellipse sizes instead of a global ellipse size. The difference is that the function has to check before placing a turbine if another turbine is contained in the ellipse.

```

def _expand_individual_ellipse_size(
    self, ad, tt, lower: int, turbine_count:int):
    pivots_res = [lower] * turbine_count
    for i in range(turbine_count):
        # test if can expand ellipse
        pivots = pivots_res.copy()
        pivots[i] = lower + self._precision
        turbine_placements = self._place_all_turbines(
            ad, lower, tt, turbine_count, pivots)
        if not turbine_placements:
            continue
        pivots = pivots_res.copy()
        # double the size is impossible
        upper = lower * 2
        lower_temp = lower
        while (upper - lower_temp) > self._precision:
            pivots[i] = (lower_temp + upper) / 2
            turbine_placements = self._place_all_turbines(
                ad, lower, tt, turbine_count, pivots)
            if turbine_placements:
                lower_temp = pivots[i]
                lower_farm = turbine_placements
                pivots_res[i] = pivots[i]
            else:
                upper = pivots[i]
    return lower_farm, pivots_res

```

Listing 6: Expanding the individual ellipse sizes to their maximal size

5.7 Encountered Problems

I found a bug in the pre-existing implementation of the annual energy production (AEP) calculation, which led to a completely wrong result. When comparing the AEP of the optimal positioning results with the AEP of windPRO, the resulting AEP was about 6 times lower than the AEP of windPRO for all areas. That is why I tested a layout with only one turbine placement, which confirmed that there was something wrong with the calculation.

The AEP calculation and the fetched wind data itself were correct. The problem was that the wind data was returned with a wrong corresponding wind height. This error was very hard to diagnose since the returned wind height was the height of the location not the wind height. It was not obvious without looking into the API documentation, since both the wind height above ground and ground height above sea level can be similar. At my test location this error resulted in the wind speeds getting interpolated

down instead of up, which led to the deviation in AEP.

In the test area with only one placement the wrong height was 230 m and the actual wind height was 10 m. Therefore, the wind speeds got interpolated down to 100 m turbine height instead of getting interpolated up to 100 m, which resulted in an AEP about 6 times lower than the actual amount.

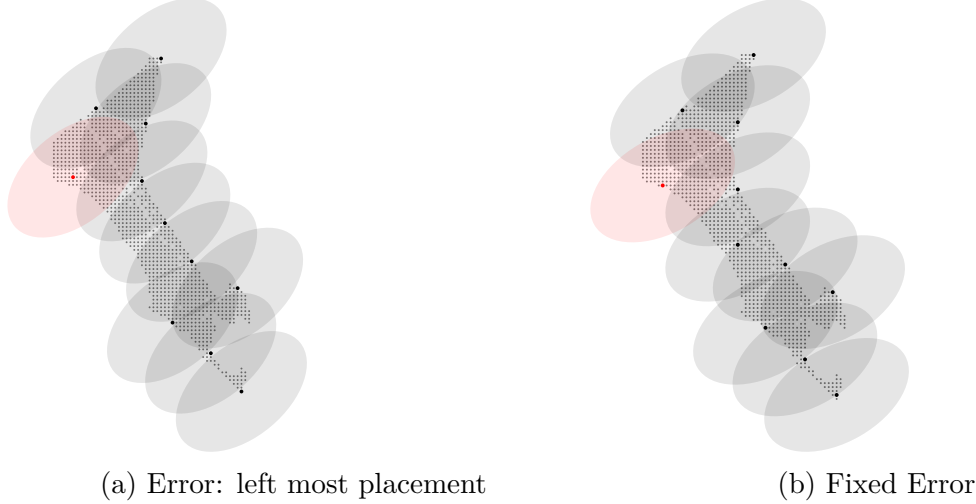


Figure 15: Error in the optimal positioning algorithm, where the leftmost ellipse is not expanded to its maximum.

The red point in Figure 15 indicates that there is an error in the optimal positioning algorithm. The point, which represents a turbine is expected to be on the edge of the area. The error is that the ellipse of the turbine is not expanded to its maximum size, causing the misplacement.

The ellipse is not expanded to its maximum, because the growing of its own ellipse size causes the placement to be pushed down lower than the next placement to the right. This in return causes the order of the placements to change, because the points are sorted to the top right. Since the order of the placements changed, but not the order of the pivots, the binary search will then abort, because the switched placement is not feasible anymore. The reordering causes the expanding of the ellipse to stop when the point is on the same height as the next placement.

The solution to this problem is to save the initial turbine placements and then check if the next turbine ellipse from the old placement contains the position candidate. The position candidate is skipped if it is contained. If not, the position candidate is placed. This way the order of the placements is kept and the error is avoided. This only works if the turbines are placed one after another and not when every ellipse is expanded at the same time.

An exponential search to find the upper bound for the individual ellipse sizes does not work here, because the turbine positions can be pushed upwards or downwards which causes the pivot sequence to change.

5.8 Other Performance Improvements

The most called function is the function to check if a point is contained inside an ellipse, therefore the biggest speed improvements can be achieved there.

Since rotating the grid points is expensive, we try to avoid it as much as possible. To check if a point is inside the ellipse, the grid needs to be rotated first. Instead of directly checking the ellipse, it can be checked first if the point is inside the square with each side being the biggest diameter of the ellipse. Instead of using the absolute distance for the check, the squared distances are used since it is faster to calculate.

A cache from functools⁸ was introduced to cache the function to rotate the grid points in the alignment direction, since the grid points have to be rotated often to check which points are contained in an ellipse.

Instead of rotating and sorting the grid points every time, the grid points are sorted once at the beginning and rotated for each alignment direction.

5.9 Annealing Pattern Implementation

The dense packing of ellipses, which all face the same direction, results in a hexagon pattern, because it is essentially circle packing and the hexagon shape for circle packing was proved optimal by Graham [4].

In the pattern based approach, first a hexagon grid is generated, where the grid points are the turbine placements and where the wind comes from the right. Simulated annealing is used to find the optimal alignment of the pattern to the potential area, which reaches the highest turbine amount. This approach, other than the greedy approach, does not have discrete points to place the turbines on a grid, but aligns the pattern to the continuous area.

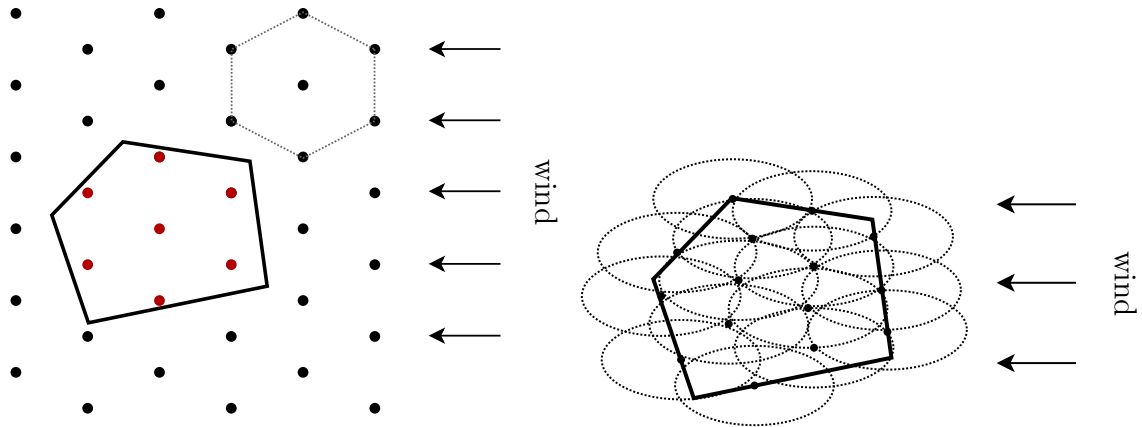


Figure 16: Comparison between hexagon pattern (left) and greedy ellipse (right).

As in the greedy approach, for each direction in ± 30 degrees, the placement is calculated and the highest turbine amount is chosen. If the turbine amounts are tied,

⁸functools, URL <https://docs.python.org/3/library/functools.html>

the closest direction to the main wind direction is chosen, just like the ellipse approach. The whole grid is turned in the alignment direction, but additionally the grid needs to be translated back into the center of the area. This is done by translating the center of the grid to the center of the bounding box of the area. Due to the rotation, the grid size has to be larger than the bounding box to properly fit the grid within the area.

In Figure 16 the differences between the hexagon pattern and the greedy ellipse approach are visualized. After the best alignment direction was found, just as in the greedy approach, the grid distance is maximized. The difference is that instead of increasing both the ellipse axes, only the minor axis is increased by increasing the grid cell heights for each column individually. The maximum height to increase the space between placements is also found with a binary search. Here, only one step of increasing the individual column heights is necessary instead of two.

Before the turbine amount and best alignment can be calculated with annealing, the existing turbines need to be considered. This is done by creating an ellipse for each existing turbine in the alignment direction and subtracting the ellipses from the area, by calculating the difference between the area and the existing turbine ellipses.

The best alignment of the grid such that the most points are contained in the area can be found with simulated annealing, but not with a direct calculation. For example a straight line can be intersected with the area and the points in between can be calculated, because the grid has the same distance between the points. However, since the area can be very irregular and even have holes in it, the points cannot be calculated directly.

The annealing process to find the best alignment of the grid for the potential area works as follows:

The for loop of the annealing needs to have a maximum iteration, which is set to 100. First in the perturb step, the grid is translated randomly in the area limiting the maximum translation to the cell size, since it is a grid. Then in the objective function, the contained points are calculated, similar to potential area, with a STRTree of the grid points and querying it by the area. Then in the accept step, the new alignment is accepted if the turbine amount is higher than the old alignment or if a random number is smaller than the temperature.

5.9.1 Generating the grid

The hexagon pattern with the distances of the minor and major axes of an ellipse does not result in the most dense packing (see Figure 17).

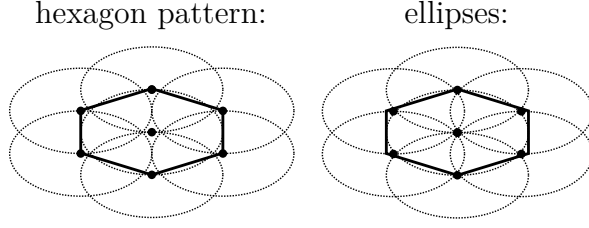


Figure 17: Comparison hexagon pattern to ellipse packing.

If the height of the grid is set to the minor ellipse axis length, the width of the grid can be calculated to achieve the most dense packing of ellipses. Every second row is offset by half the height of the minor ellipse axis, to achieve the hexagon pattern. The correct width is calculated by using the ellipse formula with y set to half the height of the minor ellipse axis:

$$1 = \frac{x^2}{a^2} + \frac{y^2}{b^2} = \frac{x^2}{a^2} + \frac{(b/2)^2}{b^2} \Leftrightarrow x = \frac{\sqrt{3}}{2}a \quad (4)$$

This results in a $b \times \frac{\sqrt{3}}{2}a$ grid with every second row offset by $\frac{b}{2}$ and where b is the minor ellipse axis and a is the major ellipse axis.

5.9.2 Optimize found grid alignment

Just as in the greedy approach, the maximum ellipse size is found with a binary search on the ellipse size, but here only the minor axis is increased. If both axes were to be increased, the wake would be less optimal. The minor axis equals the grid cell height.

Since the ellipse placements are so similar to the hexagon pattern, similar heuristics can be used to speed up the calculation. Just as before, the doubling of the height of the grid is not possible, since fewer turbines than the required minimum would be placed.

The positioning of the grid in the find maximum turbine amount stage is saved and used as the initial translation of the grid to speed up the annealing process. The positioning of every binary search step is also saved, so that the positioning has to be varied less.

The annealing process can be stopped early in the optimization stage when the annealing finds the target turbine amount or higher.

Similar to the ellipse algorithm, each row can be checked for a possible height increase. The grid height is set to the minor ellipse axis length plus the minimum precision, and it is checked whether the turbine amount remains the same.

5.10 Comparison of the Results

The performance of the different algorithms is tested with the 10 areas from the potential area calculation of the project Herzogenrath. The turbine model, which was chosen for the analysis, is the VESTAS V80-2.0MW with a hub height of 100 m.

First the newly implemented greedy ellipse and pattern results are compared to each other. Then the greedy ellipse results are compared to the greedy matrix attractiveness algorithm and the windPRO results. Unlike the implemented algorithms in this paper, the turbine amount has to be specified before optimizing for the attractiveness algorithm and windPRO. WindPRO is a commercial software for wind farm planning and optimization.

The pre-existing greedy attractiveness matrix algorithm [13] works by first calculating the attractiveness matrix, which gives every position an attractiveness score depending on the objective. For AEP that would be the lowest wake effect on the other turbines. The matrix is calculated for every turbine amount and the best position is chosen greedily until the desired turbine amount is reached. The greedy attractiveness algorithm results in better outcomes than a genetic or a random approach in most cases.

Unlike the attractiveness algorithm, in windPRO the same wind data could not be used for the calculation. This is why the AEP results might differ.

For the greedy ellipse and pattern algorithm the layout was optimized for the highest turbine amount. The greedy matrix attractiveness algorithm results were optimized for the highest Annual Energy Production (AEP). The WindPRO results were optimized for the lowest Levelized Cost of Energy (LCOE), which does not make a huge difference to AEP. This was necessary, because if the AEP is optimized, the LCOE is not calculated by windPRO. However, because the turbines will be placed closer to each other, the AEP will be slightly lower, but the LCOE will be significantly better (I calculated both).

In WindPRO, the smart placement algorithm was used instead of a random placement, which appears to be some kind of genetic algorithm.

WindPRO uses the same wake decay DTU-Standard for its wake loss and AEP calculations, but a fixed wake decay constant of 0.09. The LCOE implementation for greedy ellipse, attractiveness and pattern annealing uses a wake decay factor calculated dependent on the hub height and surface roughness, which is about 0.06 to 0.09 depending on the surface roughness of the site. For both windPRO and the implementation, the wake model is the N.O. Jensen Park 2 2018 model.

The cost parameters for the cost calculations are a lifetime of 20 years, an interest rate of 2.5%, and an onshore cost model. The specific costs for the LCOE calculation are different for the implementations and windPRO, because in windPRO the costs cannot be configured. The biggest costs for the wind park are the turbines, the foundation and the cabling costs. The turbine cost and foundation cost is set to the wind pro costs of 2 million euro and 280,000 euro. However, since the cabling costs are calculated differently, the resulting LCOE is not directly comparable.

5.10.1 Chosen metrics

The following metrics are chosen to compare the results of the optimal positioning algorithms:

Annual energy production (AEP) is the total amount of energy produced by a

wind farm in a year.

$$AEP \text{ (kWh)} = Power(kW) \times Hours \text{ in a Year} \times CapacityFactor \quad (5)$$

$$= P \times 8760 \times \frac{Actual \text{ energy output}}{Maximum \text{ energy output}} \quad (6)$$

Levelized cost of energy (LCOE) is the minimum cost in Euro per MWh at which the electricity must be sold to reach the breakeven point.

$$LCOE = \frac{(Investment \text{ Costs} \times Capital \text{ Recovery Factor}) + Operating \text{ Costs}}{AEP} \quad (7)$$

$$Compound \text{ Interest} = (1 + Yearly \text{ Interest Rate})^{Life \text{ Cycle Duration}} \quad (8)$$

$$Capital \text{ Recovery Factor} = Yearly \text{ Interest Rate} \cdot \frac{Compound \text{ Interest}}{Compound \text{ Interest} - 1} \quad (9)$$

Turbine amount is the number of turbines placed in the potential area.

Farm Efficiency is the percentage of the total potential power production that is actually achieved.

$$Farm \text{ Efficiency} = \frac{AEP}{AEP_{No \text{ Wake Loss}}} \quad (10)$$

Wake Loss is the loss of energy production due to the wind turbine wake.

$$Wake \text{ Loss} = 1 - Farm \text{ Efficiency} \quad (11)$$

The AEP is the most important metric to compare different placement algorithms. When the same turbine amount is placed, this should be compared directly. The LCOE is the most important metric if the amounts differ since ultimately the LCOE is the deciding factor for the profitability of a project. Since it is the final electricity price that needs to be paid to break even. The higher the AEP per turbine the lower the LCOE.

The wake loss is important to compare the different placement algorithms, since it is directly showing the efficiency loss caused by turbine placements, but a placement with a higher wake loss can produce more energy, because the turbines are placed in better wind conditions. Since the turbine amount directly impacts the LCOE and AEP, the turbine amount weighs the AEP and the LCOE against each other, where a higher turbine amount usually results in a higher LCOE and AEP. The generated power and LCOE should be in a good balance, where the target farm efficiency should be about 90% or higher [5]. Therefore, the goal is to reach a wake loss of about 10% with the highest possible AEP and the lowest LCOE.

5.11 Greedy Ellipse vs Pattern Annealing

Ten areas resulting from the potential area calculation of Herzogenrath are used for the comparison of which the most relevant can be seen in Figure 29. All areas are sorted by the found turbine amounts of the ellipse algorithm.

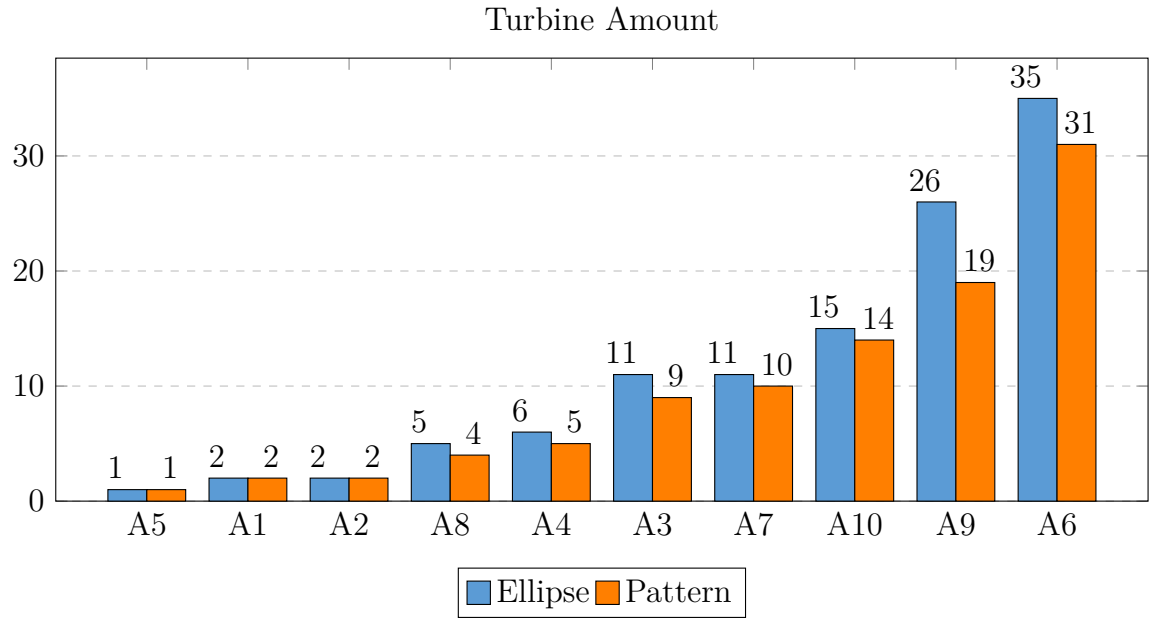


Figure 18: Comparison of found turbine amount of the ellipse and pattern based algorithm for each area.

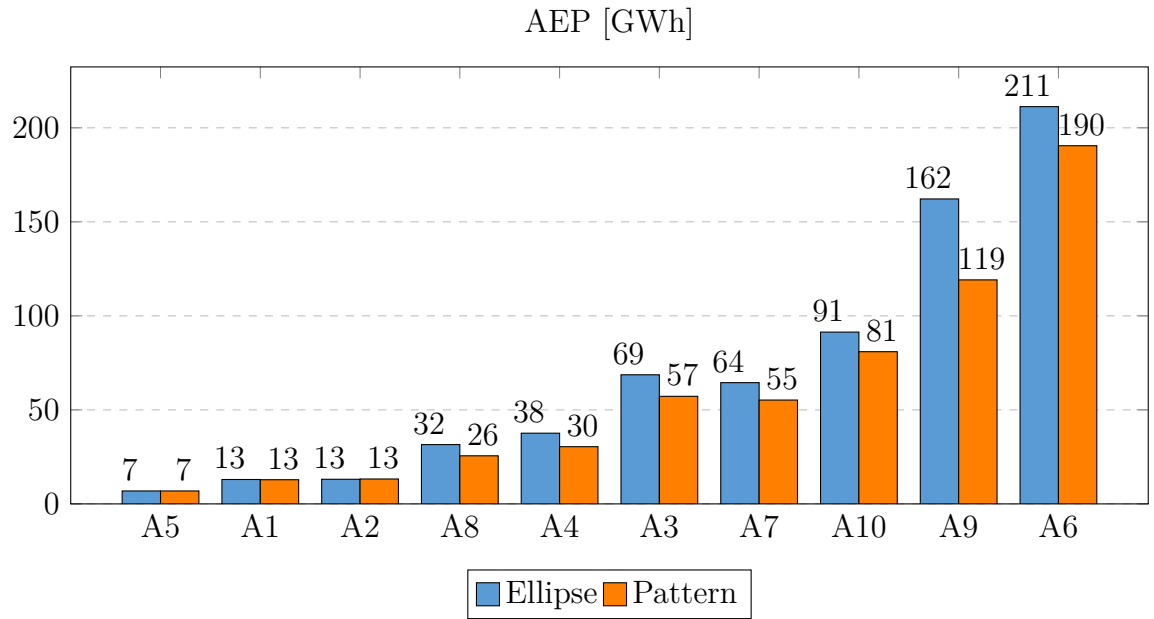


Figure 19: Comparison of Annual Energy Production of the ellipse and pattern based algorithm for each area.

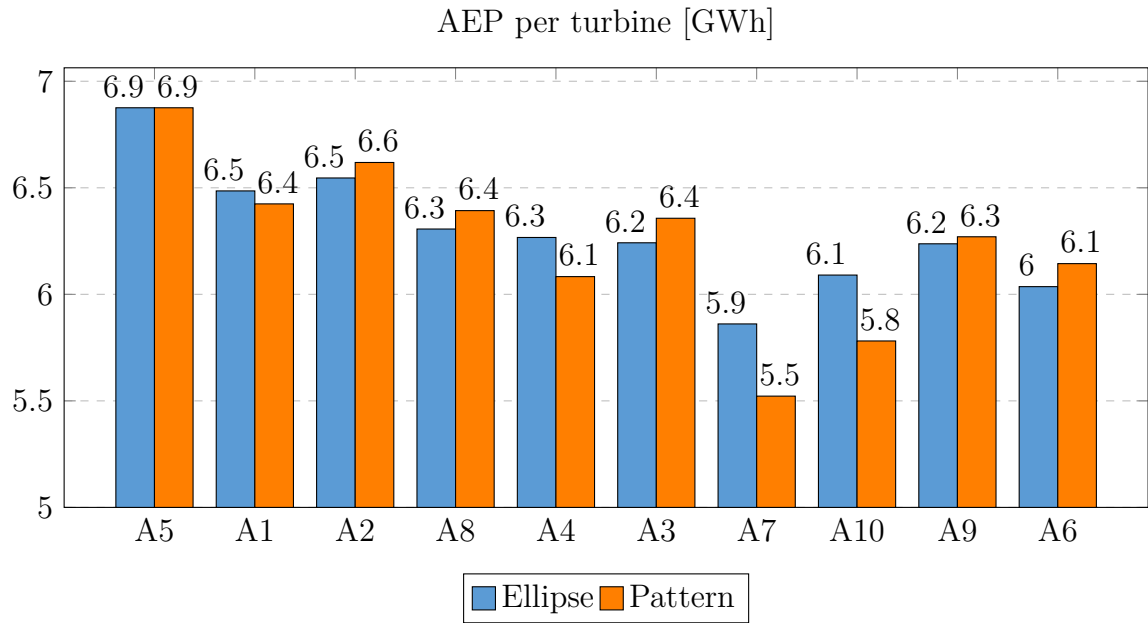


Figure 20: Comparison of Annual Energy Production per turbine of the ellipse and pattern based algorithm for each area.

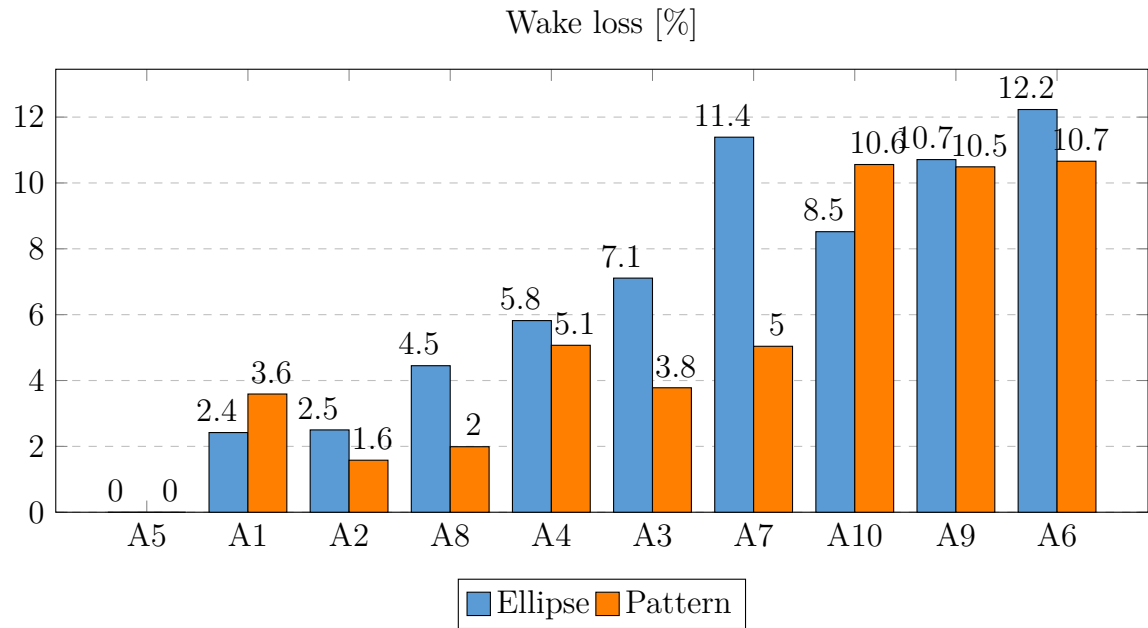


Figure 21: Comparison of wake loss of the ellipse and pattern based algorithm for each area.

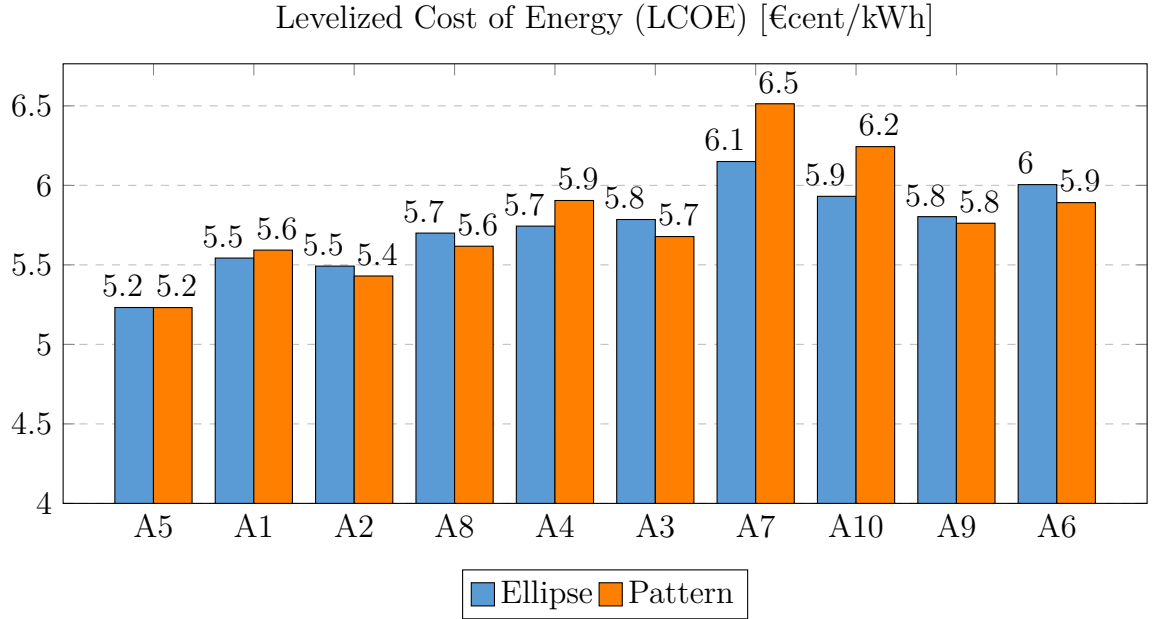


Figure 22: Comparison of LCOE of the ellipse and pattern based algorithm for each area.

The comparison in Figure 18 shows that the ellipse algorithm finds more turbines than the pattern based algorithm. This is natural since the pattern based algorithm can adjust worse to irregular area shapes.

Because of the relatively rigid grid of the pattern based algorithm, the wake loss is a lot lower for nearly all areas (see Figure 21). The wake loss for the hexagon pattern is about 10% or lower for all areas. The ellipse algorithm is more flexible leading to a less than 13% wake loss. Therefore, the wake loss is higher than the pattern algorithm, which is close to the targeted wake loss of about 10%.

The AEP is lower for the pattern based algorithm, but this is natural since the turbine count is lower. If we look at the AEP per turbine in Figure 20, the power per turbine with the ellipse algorithm is significantly higher for area A7 and A10. Otherwise, the AEP per turbine is roughly equal with a maximum deviation of about 0.2 GWh per year.

Looking at the LCOE, it is about the same or lower for the ellipse algorithm in all areas except for areas A6 and A3 where the price is 0.1 cent more expensive (see Figure 22). Note that a lower LCOE is better.

This leads me to the conclusion that the ellipse algorithm is generally better than the pattern based algorithm, since it returns a higher AEP and lower LCOE for most areas especially for irregular shapes.

5.11.1 Runtime

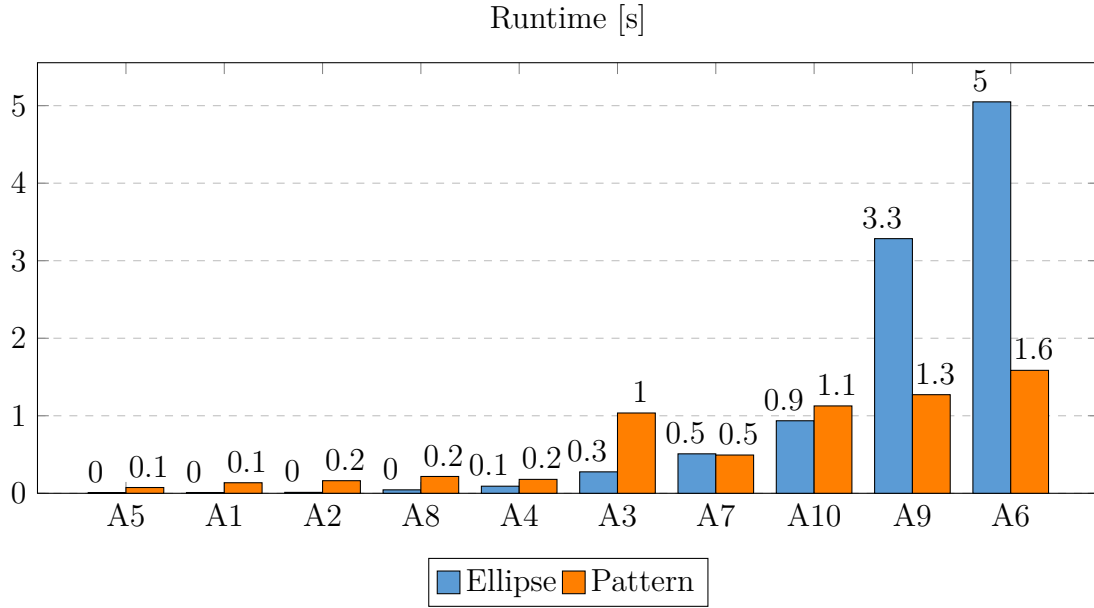


Figure 23: Comparison of the performance of the ellipse and pattern algorithm.

The runtime of the pattern annealing based and the greedy ellipse based algorithms is depicted in Figure 23. The pattern based algorithm is faster for all areas more than 5 turbines, except for A3.

The runtime complexity of the ellipse algorithm is $O(n \cdot \log(n))$ depending on the turbine count, which depends on the area size. The runtime of the pattern based algorithm is linear in the area size.

The majority of the runtime for the ellipse algorithm is spent at finding the best positioning by expanding the individual ellipse sizes in the last step, since a binary search ($O(\log(n))$) has to be performed for each turbine to increase the ellipse size. With an increase of the number of turbines, the binary search has to be performed more often, more turbines have to be placed and more grid points have to be checked for each placement.

For the pattern based algorithm, the increase in runtime is due to needing to align a bigger grid to the area. The annealing time is very short for small areas, because the saved position of the grid of the first annealing is very close to the optimal position. The bigger or the more irregular the area is, the more annealing steps are necessary to find the best alignment of the grid. This is why the runtime increases linearly excluding small areas.

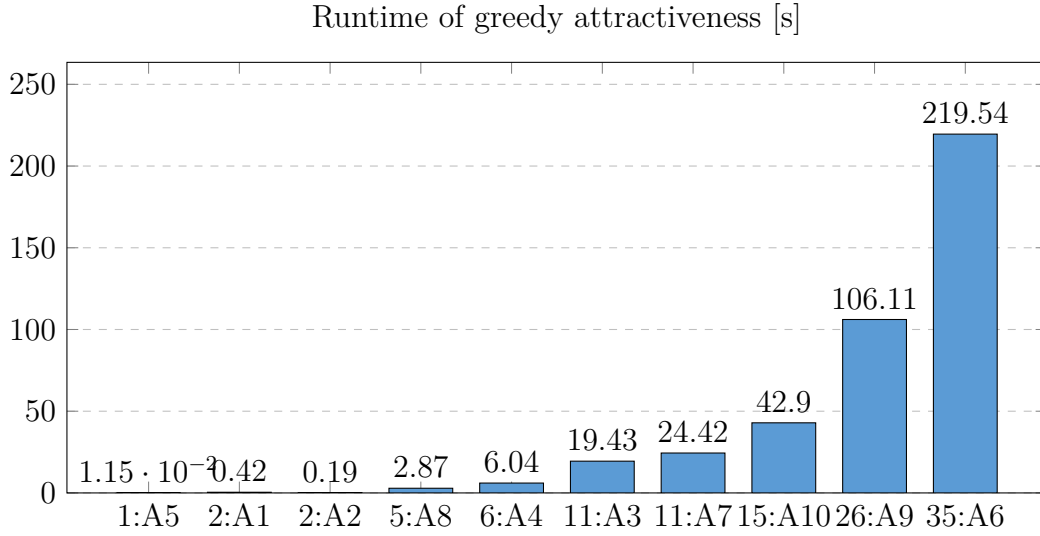


Figure 24: Performance of optimal positioning attractiveness.

The target of the ellipse and pattern algorithm is to achieve a runtime better than the pre-existing AEP greedy attractiveness matrix algorithm, which finds a solution very close to the optimal one, but is quite slow (see Figure 24). Considering that the turbine amount is automatically determined for the ellipse and pattern algorithm unlike the attractiveness algorithm, the runtime of the ellipse algorithm is significantly better.

The area size is also a compounding factor in the runtime of the greedy attractiveness algorithm. This is due to needing to recalculate the attractiveness matrix after each turbine is placed. Additionally, the more turbines are placed, the more wake effects have to be considered while calculating the attractiveness matrix, which also causes a significant slowdown.

For all runtimes of the attractiveness algorithm higher than 2 seconds, the ellipse and pattern algorithms are faster by at least 6 times. The runtime of the ellipse and pattern algorithm always remains under 10 seconds. At 35 turbines the attractiveness matrix algorithm reaches a runtime of more than 3 minutes. The runtime of the ellipse and pattern algorithms at 35 turbines is faster by a factor of over 50.

The greedy ellipse and pattern algorithms reached their goal by achieving a fast runtime, with every runtime being lower than about 5 seconds. Both the ellipse and the attractiveness algorithm have a significant slowdown depending on the area size, but the ellipse algorithm is much faster than the attractiveness matrix algorithm.

WindPRO calculates a previously selected turbine amount range and calculates the AEP for every turbine amount in this range. Calculating a single AEP takes a similar amount of time compared to the attractiveness matrix algorithm, which means it is also very slow.

The measured runtimes are for the individual potential areas of the project Herzogenrath. Calculating the total time for the project takes about 11 seconds for the ellipse algorithm, about 6 seconds for the pattern algorithm and about 7 minutes for

the attractiveness matrix algorithm, which is a huge difference. Therefore, the runtime of the ellipse and pattern algorithms is very fast in comparison.

5.12 Comparison to Greedy Ellipse Algorithm

5.12.1 Comparison of annual energy production

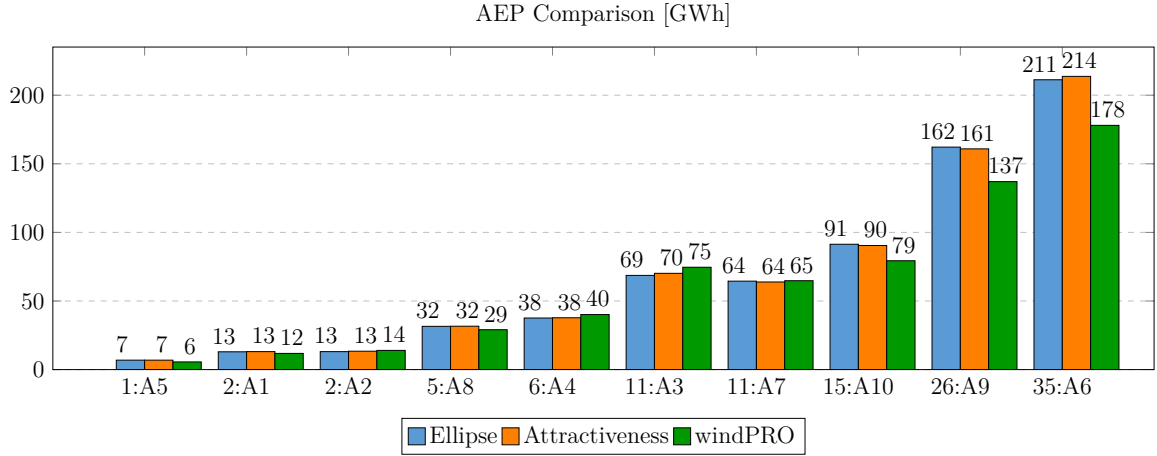


Figure 25: Comparison of Annual Energy Production of the ellipse and attractiveness algorithm and windPRO.

The optimizer has to find the best turbine amount and the best positioning for the turbines. Both the ellipse and the attractiveness algorithm reach very similar results, with the ellipse algorithm achieving higher yields than the attractiveness algorithm for some areas (7, 9, 10), which is visualized in Figure 25.

The AEP cannot be directly compared to windPRO, since windPRO uses different underlying wind data. Nonetheless, the AEP of windPRO is significantly lower than that of the ellipse and attractiveness algorithm for the areas 6 and 9. For the areas 2, 3 and 4 the AEP of windPRO is higher than that of the other algorithms. For the other areas the AEP is similar. Note that all deviations get amplified by the AEP metric because it is an annual metric and is multiplied by the hours in a year.

Figure 26 not only displays the AEP difference in MWh divided by the turbine amount, but also the total deviation in percent from the attractiveness algorithm. The areas 2 and 3 have a bigger AEP percentage deviation per turbine compared to the rest, with a difference of about 2%. The rest has a difference of about $\pm 1\%$. The average AEP difference is about 0.26% from the attractiveness result.

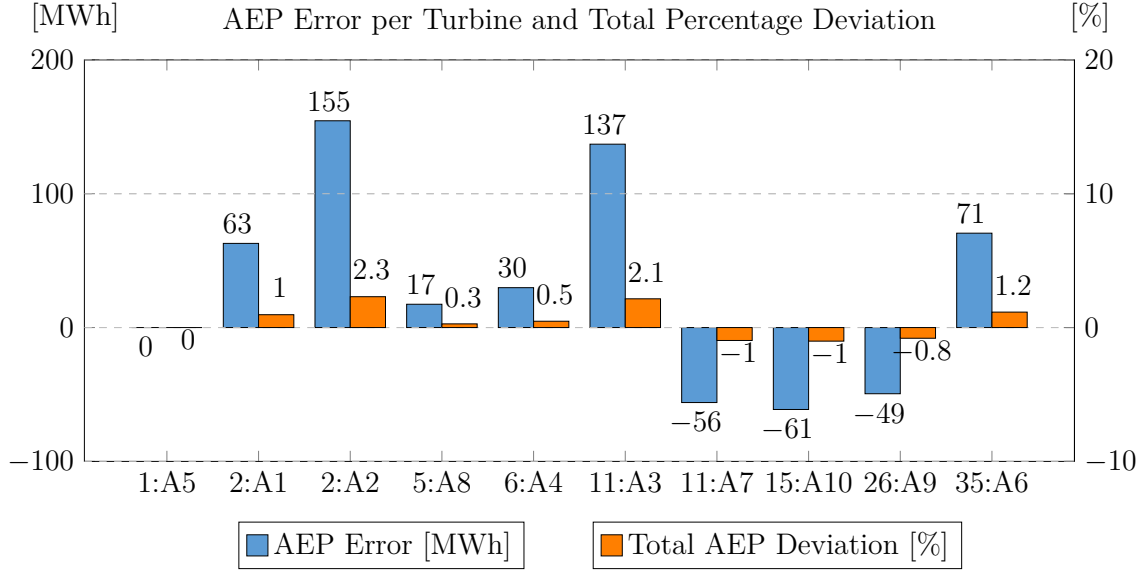


Figure 26: AEP error per turbine and total percentage deviation to the attractiveness matrix algorithm. Positive values mean that attractiveness is better.

5.12.2 Comparison of wake loss

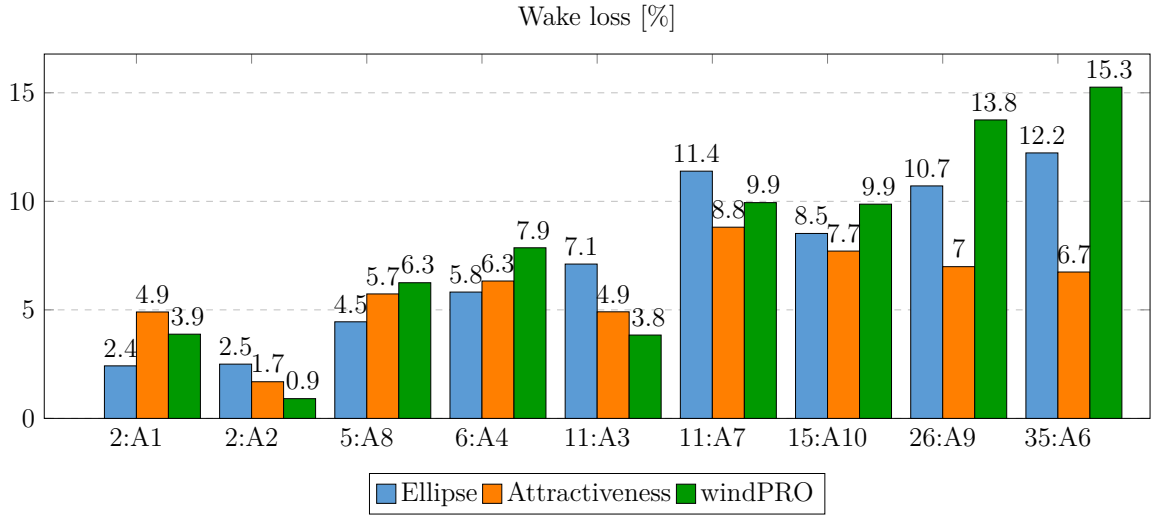


Figure 27: Wake loss of the placements

The target farm efficiency should be about 90% or higher [5]. The wake loss comparison in Figure 27 shows that the ellipse algorithm always finds a wind turbine amount, which reaches a wake loss of 13% or lower.

As seen in subsection 5.11, the hexagon pattern algorithm always finds a placement with a wake loss of about 10%. This indicates that the chosen ratio of 3:5 for the ellipse algorithm is a good choice, even though it might find a layout with a higher

wake loss trading it for a higher AEP.

The wake loss difference between the attractiveness matrix algorithm and windPRO is low except for areas 6 and 9, where the ellipse and windPRO wake loss is significantly higher. WindPRO is optimized for highest LCOE not the lowest wake loss.

The results of the ellipse algorithm achieve a similar wake loss to windPRO and attractiveness. Only areas 6, 7 and 9 have a significantly higher wake loss compared to the attractiveness result. Notably for areas 7, 9 and 10, the wake loss of the ellipse algorithm is higher, but the AEP is higher than that of the attractiveness algorithm.

5.12.3 Comparison of levelized cost of energy

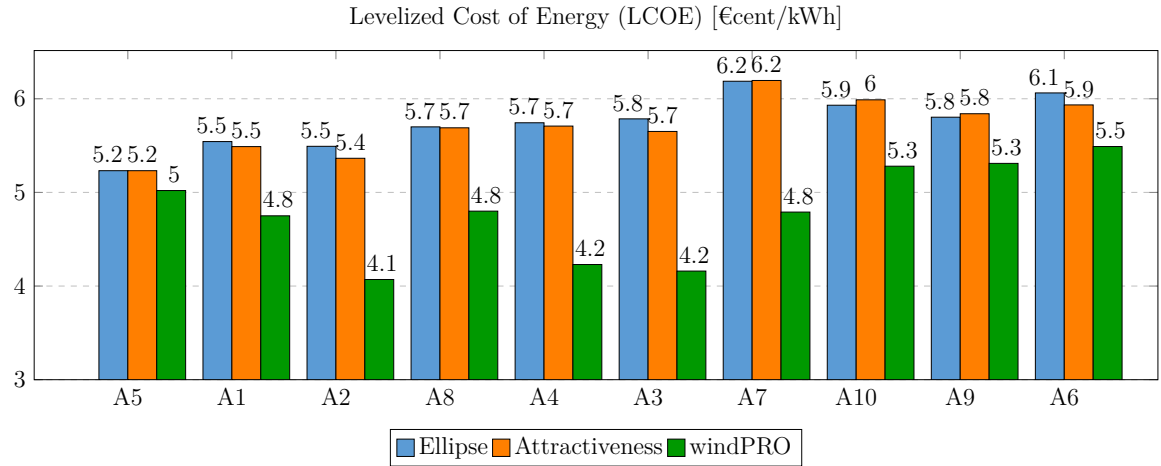


Figure 28: Comparison of LCOE of the ellipse and attractiveness algorithm and windPRO.

Figure 28 shows that the Levelized Cost of Energy of the ellipse algorithm is very similar to the LCOE of the attractiveness algorithm. The LCOE of the areas 2, 3, and 6 have the highest negative difference of up to 0.2 cent per kWh for the ellipse algorithm. The LCOE of the attractiveness algorithm is lower or equal for all areas except for areas 10 and 9.

The LCOE of windPRO is calculated with different costs compared to the LCOE calculation of the implementation. What is notable is that the LCOE of windPRO of the layouts 2, 3 and 4 is a lot lower than the other layouts of windPRO with about 0.6 cent per kWh less. This is because for area 2 the positions are closer together to minimize the cable cost and for areas 3 and 4, the turbines are placed in a line to minimize the cable cost. Unlike the other algorithms, windPRO was set to optimize for LCOE, because otherwise windPRO does not calculate the LCOE metric.

5.13 Positioning Comparison

5.13.1 Better placements than attractiveness

As seen in the previous section, the layouts 7, 9, and 10 of the ellipse algorithm have a higher AEP, but a higher wake loss compared to the attractiveness algorithm. For all these layouts, the LCOE is lower than that of the attractiveness algorithm. Therefore, the layouts of these areas are better than the layouts of the attractiveness algorithm.

5.13.2 Worse placements than attractiveness

Only layouts 2, 3, and 6 of the ellipse algorithm can be considered worse placements than the attractiveness layouts, since they have a high negative AEP deviation per turbine compared to the attractiveness algorithm. Of these three areas, only area 3 and 6 have a high wake loss, and layout 2 is skewed since it only has 2 turbines.

The LCOE of the areas 2, 3, and 6 have a high negative difference of 0.1 cent per kWh for the ellipse algorithm. Therefore, areas 2, 3 and 6 of the ellipse algorithm can be considered worse placements than the attractiveness layouts.

5.13.3 Direct positioning comparison

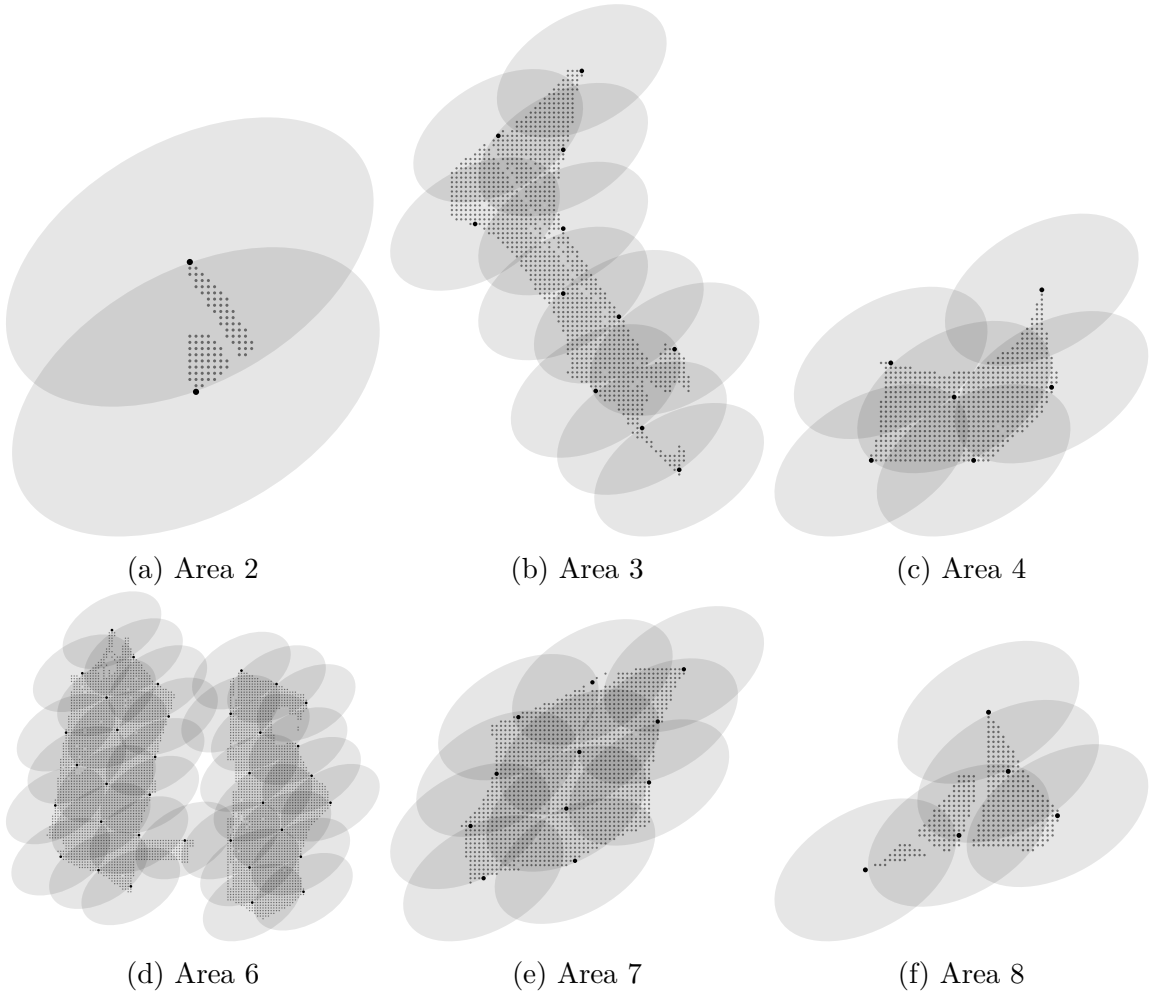


Figure 29: Optimal placements of the potential areas of Herzogenrath project. The displayed ellipse size is the global maximum ellipse size. The main wind direction is 30° .

The placements in the potential area by the greedy ellipse algorithm seen in Figure 29 show that the maximum amount of turbines are placed correctly in the area, while keeping the turbines as far apart as possible. The previous subsection showed that the areas 2, 3 and 6 of the ellipse algorithm are worse placements and areas 7, 9 and 10 are better placements compared to the attractiveness algorithm. The worse placements still only have a difference of about 2% AEP deviation compared to the attractiveness algorithm.

The placement of area 2 of the ellipse algorithm is not perpendicular to the wind direction due to the shape of the area. The expanding of the ellipses causes the second turbine to be pushed to the lowest left corner, which is less perpendicular to the wind direction. This causes the AEP to be lower and the LCOE to be higher than the attractiveness result.

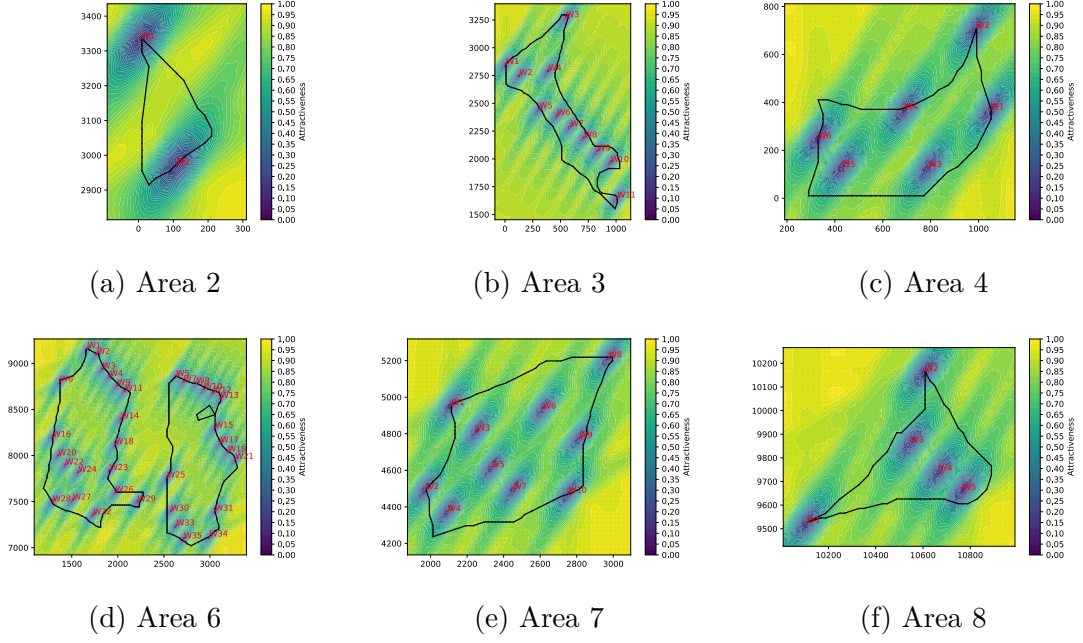


Figure 30: Optimal placements of the potential areas of the Herzogenrath project, calculated with greedy attractiveness algorithm. The color shows the attractiveness score.

In area 3, the expanding of the ellipses causes the turbines in the middle not to be placed in a row but on alternating edges, which is less optimal than placing them perpendicular to the wind direction in a line, like the attractiveness algorithm does.

In area 6, the turbines on the bottom right are expanded to the maximum, but placing them lower in the corner would be a better placement. This is a general problem with the ellipse approach.

In general, the ellipse algorithm finds similar placements to the attractiveness matrix (see Figure 30), except for areas 3 and 6. There, the attractiveness algorithm opts for a dense placement of turbines in the direction perpendicular to the main wind direction, which is less than the minimum ellipse size and keeps a big distance between the rows.

Some differences of the placements can be accounted for because of the big difference in alignment direction, as the ellipse greedy algorithm chooses the alignment direction as close as possible to the main wind direction.

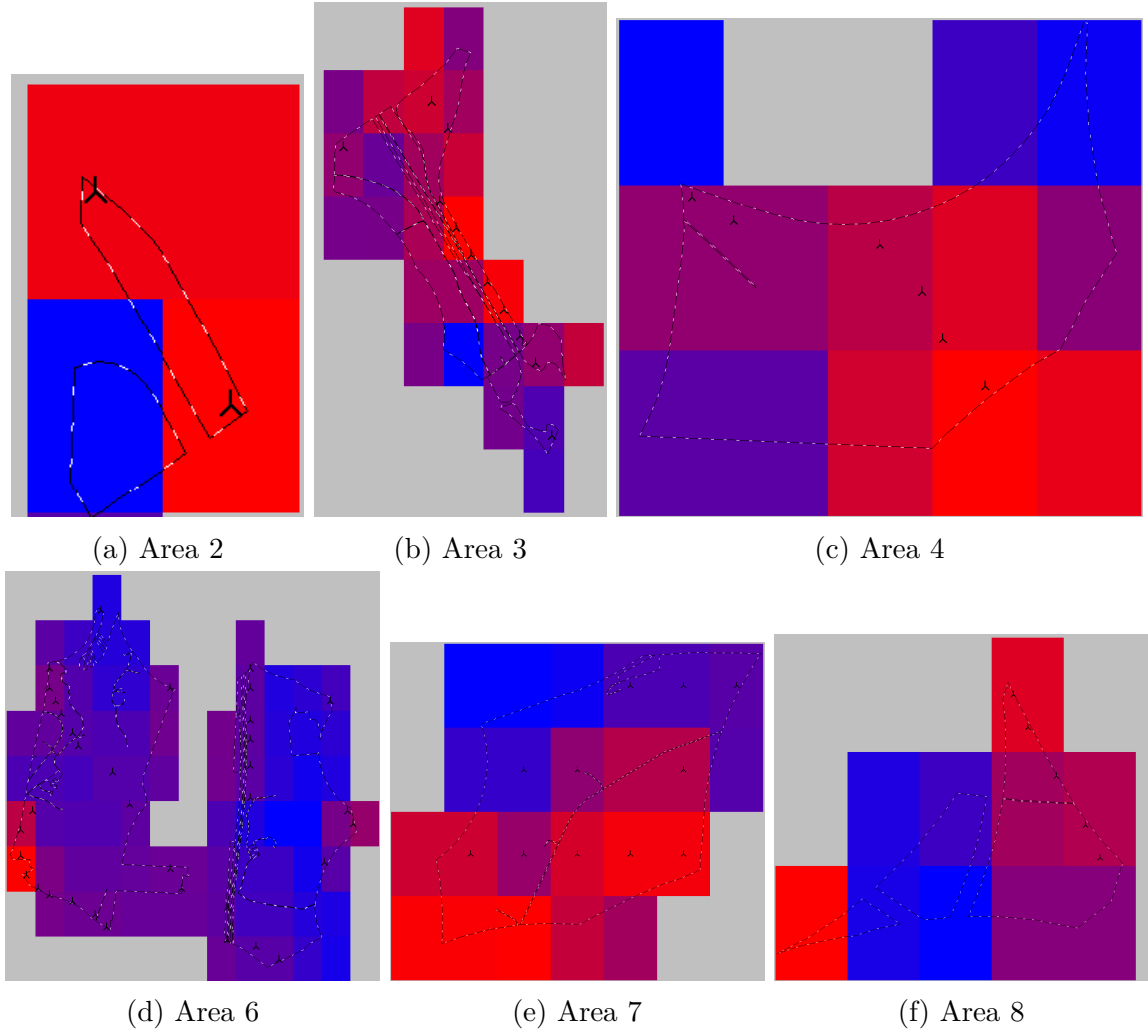


Figure 31: Optimal placements of the potential areas of the Herzogenrath project, calculated with AEP in windPRO. The color represents the average wind speed.

WindPRO places the turbines in area 3 and 8 similar to the ellipse algorithm along the right edge of the area. While area 6 is placed more similar to the attractiveness algorithm, as straight and as perpendicular as possible to the wind direction. Unlike the attractiveness placement, the wind turbines of the right area in area 6 are placed along the left edge because the wind speeds are higher there.

The placements of area 7 could be incorrect, because the turbines are all placed in the center of the wind squares. I would expect a placement more perpendicular to the wind direction. As the AEP and wake loss seems reasonable maybe only the visualization of windPRO is unreliable.

All in all, windPRO finds more similar placements to the greedy attractiveness algorithm, which tries to place the turbines as straight and as perpendicular as possible to the main wind direction.

5.14 Comparison Conclusion

The Levelized Cost of Energy (LCOE) and the annual energy production (AEP) of the greedy ellipse algorithm are very similar and sometimes better than the greedy attractiveness matrix algorithm. The LCOE deviation of the greedy algorithm to the attractiveness algorithm is about 0.2 cent per kWh at its highest. The LCOE of windPRO is better for some layouts, because it prioritizes putting the turbines closer together to minimize cabling costs.

The average AEP deviation of the ellipse algorithm is only 0.26% from the attractiveness result. The wake loss is about 1–4% higher for the ellipse algorithm compared to the attractiveness result, but it remains at close to the 10% wake loss target.

The runtime of the ellipse algorithm stays under 10 seconds, while the attractiveness matrix algorithm takes more than 3 minutes in the biggest potential area alone. In conclusion, the ellipse algorithm achieves a very good estimation of the optimal turbine placements and achieves a good balance between the farm efficiency and power output, while staying within a very short runtime.

6 Future Work

The state North Rhine-Westphalia has its own freely available wind data source, which could be integrated in the future.

Calculating the potential areas for an entire country like Germany is feasible with the current speed of the potential area calculation. For example, it takes about 5 minutes to calculate the whole of North Rhine-Westphalia. Therefore, I estimate that calculating the entire country would take approximately $4\text{min} \times 16\text{states} = 64\text{min}$. However, the GIS data for the whole of Germany is estimated to be around 100GB, which is too large to store as a JSON file and retrieve from rate limited APIs. To address this, I have implemented an iterative algorithm for potential areas, which retrieves the GIS objects by county and merges them iteratively to avoid saving the data into a single file. The problem is that there are API limits for the size and number of requests. Therefore, either the data has to be downloaded and self-hosted, or the API has to be exchanged with one without limits.

Instead of calculating the grid points of the potential areas at the end of the calculation, it would make sense to calculate them when they are needed, like for optimal positioning. This would allow for a more flexible grid depending on the required precision. For example, a more precise grid could lead to a more optimal positioning of the wind turbines or a less precise grid could lead to a faster calculation.

Instead of using the grid points for the greedy ellipse approach, the ellipses could be placed in the continuous area directly with an annealing approach, which could lead to better placements.

7 Conclusion

The wind data sources from various official German potential area studies were analyzed. The German Weather Service (DWD) and the Global Wind Atlas (GWA) data was used in the potential area studies. Therefore, the Global Wind Atlas was newly added as a data source. The implementations for extracting the wind data from DWD and GWA were both verified by comparing the extracted data with the online atlases. Although the data from the Global Wind Atlas is newer than the DWD data, it overestimates the wind speeds by about 1 m/s.

Integrating the official real estate cadasters and nature restriction areas of every German state improves the correctness of the potential area calculation, as the calculation directly depends on the underlying geometric information.

The direct merging approach has demonstrated that potential areas can be calculated with the highest possible accuracy and faster than the grid-based approach. A new option was added to both approaches to exclude areas with insufficient wind speeds with a customizable threshold. The implementation was verified by comparing the results of the two approaches to each other and to the Herzogenrath study.

Calculating the potential areas for the test project takes about 7 seconds with the new approach, compared to the 2 minutes of the initial implementation. The new second approach runs faster the higher the distances are and the sped up first approach runs faster the lower the distances of the distance regulations are. However, the second new approach already runs faster than the first when all regulations are set to a minimum distance of 20 m.

For the optimal positioning the greedy ellipse approach has achieved its goal to reach a good approximation of the optimal positioning, while being very performant. The ellipse approach can even outperform the attractiveness approach in some cases, where it yields a better annual energy production (AEP) and Levelized Cost of Energy (LCOE). The ellipse approach performs worse in very small areas, which are shaped irregularly, because the ellipse algorithm enforces a big minimum distance, which can be less optimal in small areas. In the worst case, the AEP is about 2% lower and the LCOE is about 0.2 cents per kWh higher than the attractiveness approach. On average, the AEP is very similar to that of the attractiveness algorithm and of the commercial software windPRO.

The ellipse approach has a quite fast runtime of about 10 seconds for the Herzogenrath project, compared to the greedy matrix attractiveness approach, which takes about 7 minutes for the project. WindPRO takes even longer than the attractiveness approach. The runtime for the attractiveness approach as well as for the ellipse approach increases significantly with the size of the area, which corresponds to the number of turbines that need to be placed. However, the runtime of the greedy ellipse algorithm is still several times faster.

This work provides a solution for quickly adapting to regulatory changes, ensuring that potential areas and optimal wind turbine positions can be recalculated swiftly, with easily modifiable parameters.

References

- [1] globalwindatlas.info. URL <https://globalwindatlas.info>.
- [2] Jake Badger, Andrea Hahmann, Xiaoli Guo Larsén, Merete Badger, Mark Kelly, Bjarke T Olsen, and Niels G Mortensen. The Global Wind Atlas: An EUDP project carried out by DTU Wind Energy. *Dtu*, 2015. URL https://backend.orbit.dtu.dk/ws/portalfiles/portal/238494910/GWA_64011_0347_FinalReport.pdf.
- [3] Claudia Guski and Jens Müller. Stadt Herzogenrath Potenzialstudie erneuerbare Energien. Technical report, Stadt Herzogenrath, Herzogenrath, 1 2024.
- [4] H S M Coxeter, M G Greening, Ron Graham Graham, and Jürg Rätz. E1921. *The American Mathematical Monthly*, 75(2):192, feb 1968. ISSN 00029890. doi: 10.2307/2315913. URL <http://www.jstor.org/stable/2315913><https://www.jstor.org/stable/10.2307/2315913?origin=crossref>.
- [5] DLG. Planung von Windenergieanlagen Worauf Land-und Forstwirte achten sollten DLG-Merkblatt 395. *Fachzentrum Land- und Ernährungswirtschaft*, 2014. URL https://www.dlg.org/fileadmin/downloads/Merkblaetter/dlg-merkblatt_395.pdf.
- [6] Dr. Sibylle Pawlowski. LANUV-Fachbericht 142. Technical report, LANUV NRW, 2023. URL <https://www.energieatlas.nrw.de/site/Media/Default/Dokumente/LANUV-Fachbericht142.pdf>.
- [7] Dr. Stefan Bofinger, Doron Callies, Michael Scheibe, Yves-Marie Saint-Drenan, and Dr. Kurt Rohrig. Potenzial der Windenergienutzung an Land. page 23, 2012. URL https://www.wind-energie.de/fileadmin/redaktion/dokumente/publikationen-oeffentlich/themen/01-mensch-und-umwelt/03-naturschutz/bwe_potenzialstudie_kurzfassung_2012-03.pdf.
- [8] Dr. Wolfgang Peters, Tim Herbeck, Silvio Hildebrandt, Dr. Carsten Pape, David Geiger, Christoph Zink, and Arne Füsers. WIN-NIEPOT. *Fraunhofer*, 2024. doi: 10.5771/9783748916505. URL <https://www.umwelt.niedersachsen.de/startseite/aktuelles/ergebniskarten-der-windflächenpotenzialanalyse-downloadmöglichkeit-220485.html>.
- [9] DWD. DWD NRW Poster, 2004. URL https://www.dwd.de/DE/leistungen/windkarten/deutschland_und_bundeslaender.html#buehneTop.
- [10] DWD. DWD DATENSATZBESCHREIBUNG, 2014. URL https://opendata.dwd.de/climate_environment/CDC/grids_germany/multi_annual/wind_parameters/resol_200x200/BESCHREIBUNG_gridsgermany_resol_200x200_de.pdf.

- [11] Tuhfe Göçmen, Paul Van Der Laan, Pierre Elouan Réthoré, Alfredo Peña Diaz, Gunner Chr Larsen, and Søren Ott. Wind turbine wake models developed at the technical university of Denmark: A review. *Renewable and Sustainable Energy Reviews*, 60:752–769, 2016. ISSN 18790690. doi: 10.1016/j.rser.2016.01.113. URL <http://dx.doi.org/10.1016/j.rser.2016.01.113>.
- [12] LANUV NRW. INSPIRE Wind map download, 12 2023. URL <https://www.gis-rest.nrw.de/atomFeed/rest/atom/0c682cfc-4dd1-4c21-96cb-06d571a8ecd1.html>.
- [13] Nicolai Radke, Patrick E.M. De Smet, and Erika Ábrahám. Numerical modeling and analysis of shadow flicker using solar path functions for enhanced predictive accuracy. *Energies*, 18, 1 2025. ISSN 19961073. doi: 10.3390/en18020352. URL <https://www.mdpi.com/1996-1073/18/2/352>.
- [14] Lalon Sander. Strommix Deutschland: Wie hoch ist der Anteil erneuerbarer Energien?, mar 2025. URL <https://www.ndr.de/nachrichten/info/Strommix-Deutschland-Wie-ist-der-Anteil-erneuerbarer-Energien,strommix102.html>.
- [15] Sebastian Dijks, Dr. Wolfgang Peters, Tim Herbeck, Dr. Silvio Hildebrandt, Dr. Carsten Pape, and Arne Füsers. Sammeldokument zur Windflächenpotenzialstudie Analyse der Flächenpotenziale. *Fraunhofer*, 3 2024. URL https://www.saarland.de/SharedDocs/Downloads/DE/mwide/energie/dld_windBG_SL_endbericht_wfps_2024.pdf?__blob=publicationFile&v=4.

A Figures

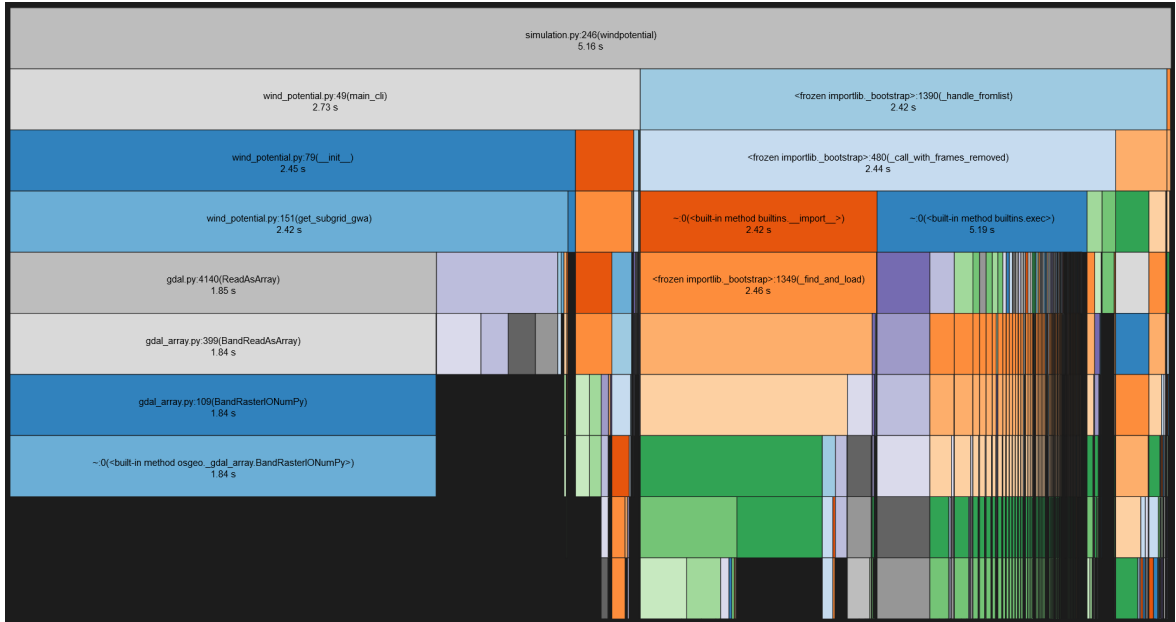


Figure 32: SnakeViz¹ visualization of the profiling² of the Implementation of the GWA data processing

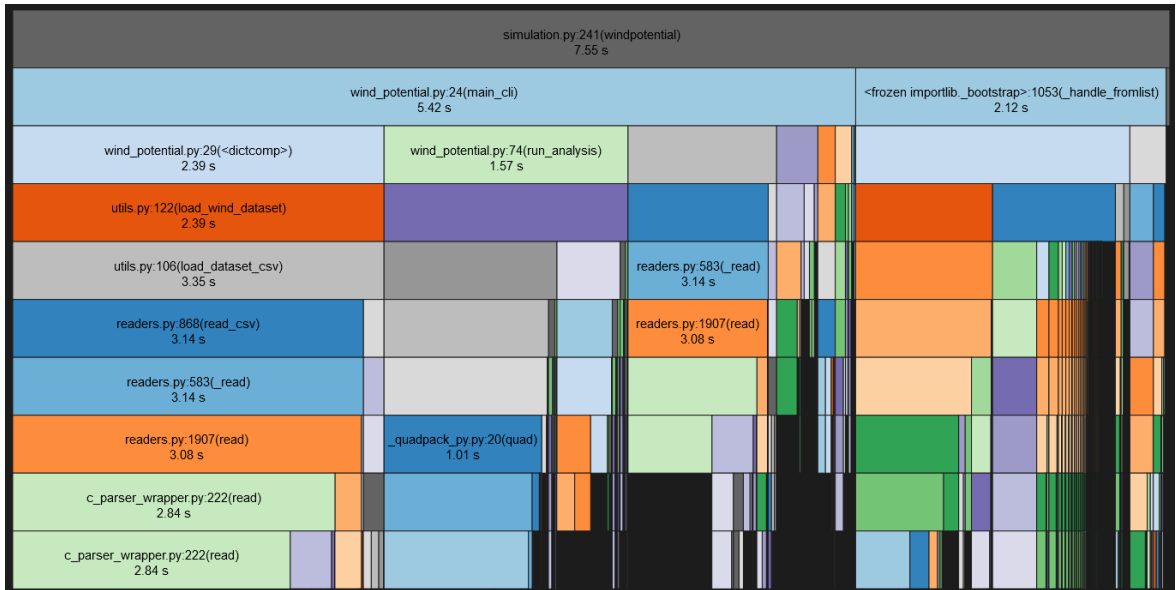
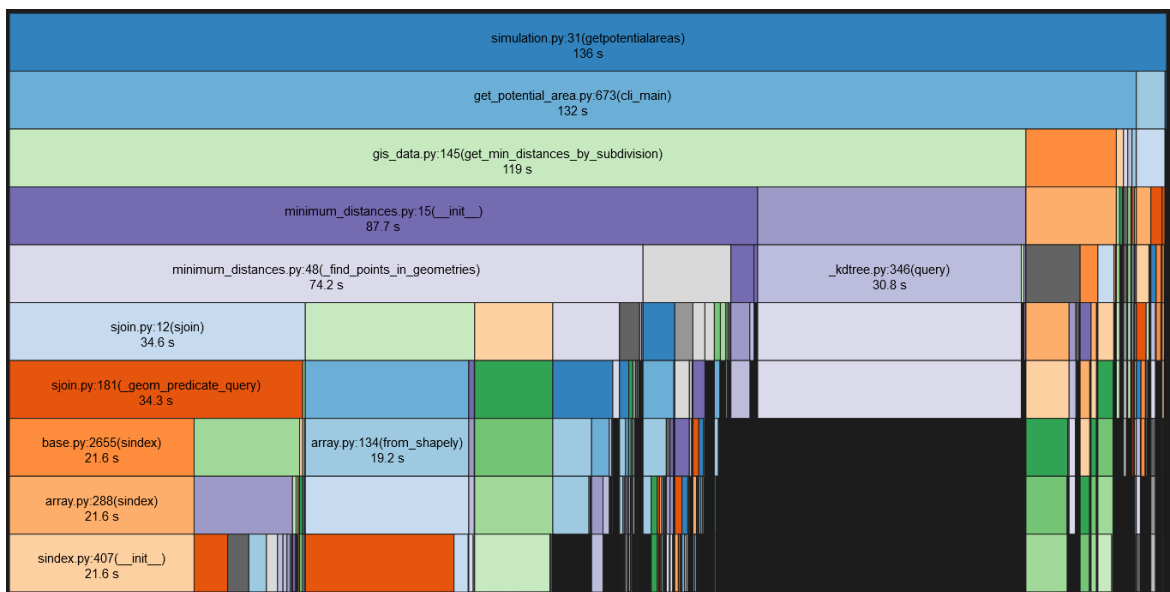
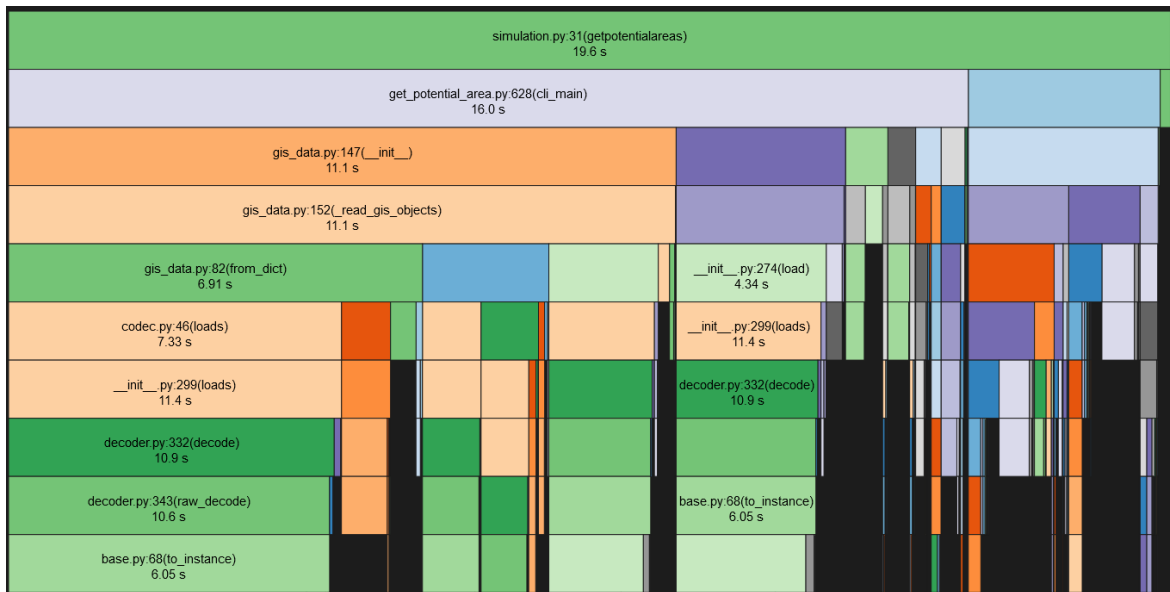


Figure 33: SnakeViz¹ visualization of the profiling² of the original DWD data processing

¹SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

²cProfile, URL <https://docs.python.org/3/library/profile.html>

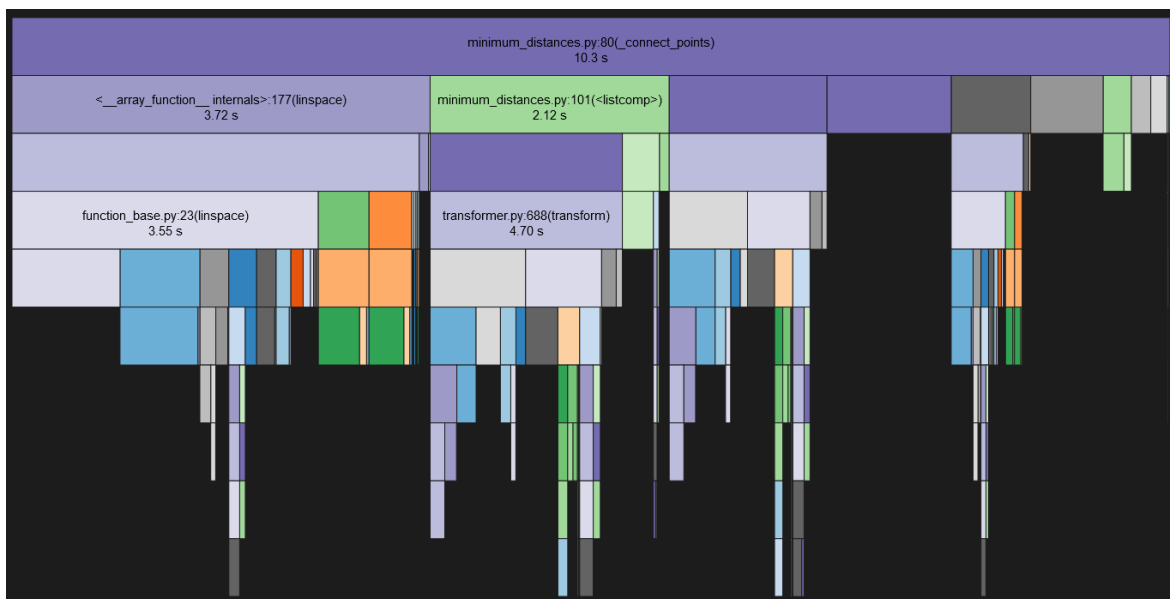
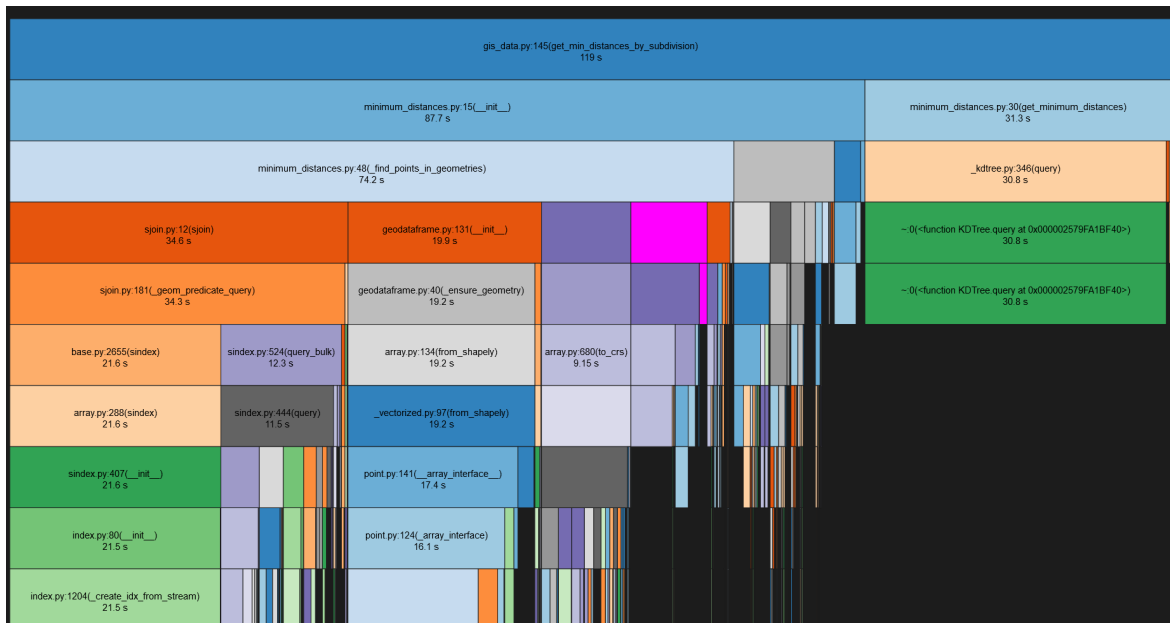


¹SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

²cProfile, URL <https://docs.python.org/3/library/profile.html>

¹SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

²cProfile, URL <https://docs.python.org/3/library/profile.html>



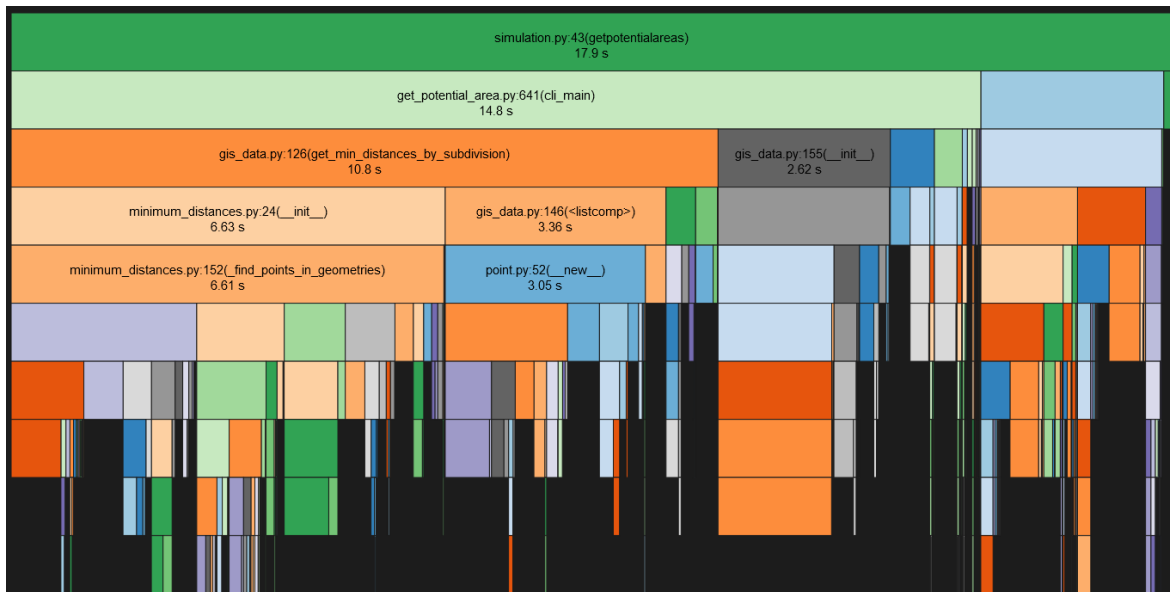


Figure 38: This is getpotentialareas after all performance improvements and fixes, visualized by SnakeViz¹.

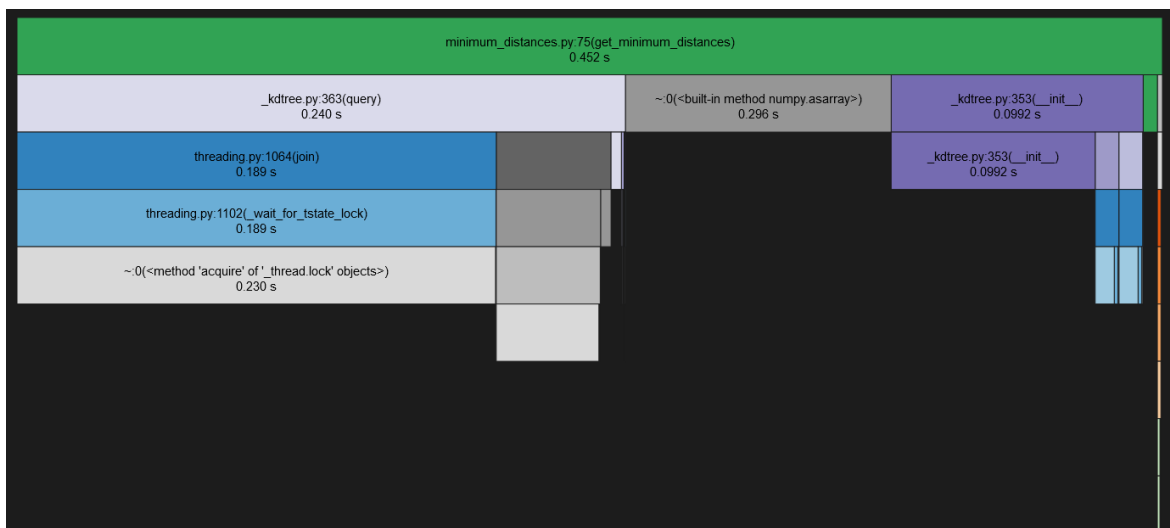


Figure 39: This is the function get_minimum_distances of getpotentialareas after all performance improvements and fixes, visualized by SnakeViz¹.

¹SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

²cProfile, URL <https://docs.python.org/3/library/profile.html>

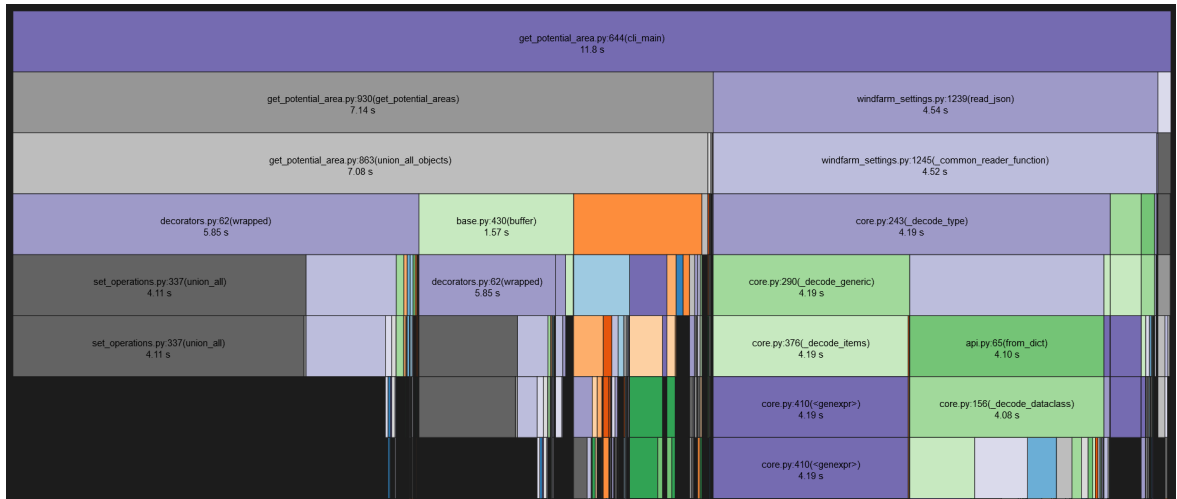


Figure 40: This is getpotentialareas with the union approach, visualized by SnakeViz¹.

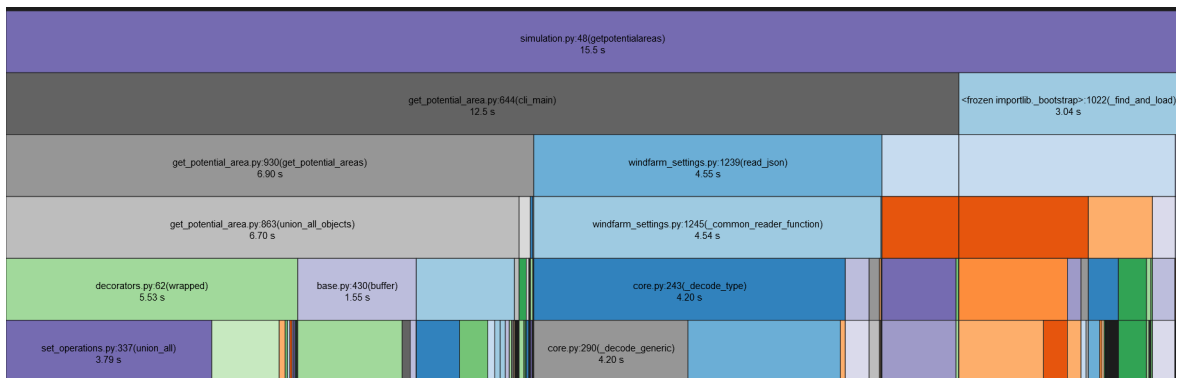


Figure 41: This is getpotentialareas with the second approach with 20m distance, visualized by SnakeViz¹.

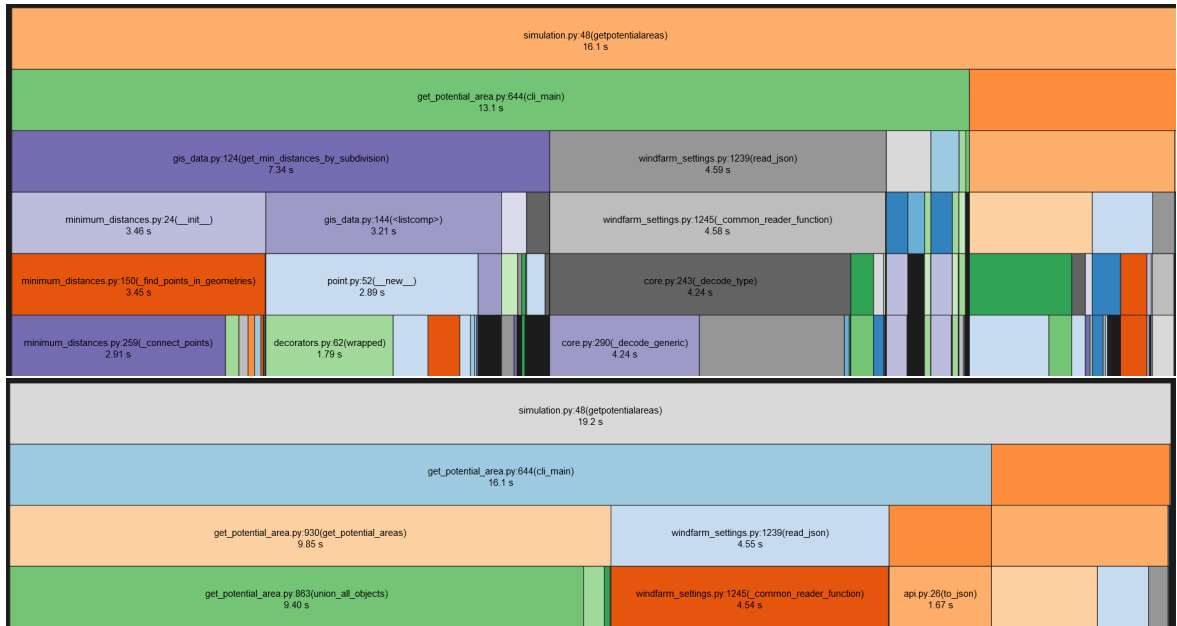


Figure 42: This is `getpotentialareas` with the first approach on the left and the second on the right, visualized by SnakeViz¹.

List of Tables

1	Wind data sources of the wind potential area studies of the individual states	8
2	All used GIS object classes for the potential area calculation	16
3	All real estate cadaster APIs for the individual states in Germany. The Data type ‘simplified’ stands for ‘ALKIS vereinfacht’, which is the simplified access to ALKIS.	18

List of Figures

4	The maps show the mean wind speed in 150 m height. Left is the GWA online atlas [1] and right the same area, but the data was processed and visualized by the implemented code.	12
5	The maps show the mean wind speed in 80 m height. Left is a PDF of the DWD data [9] and right you can see the same area, but the data was processed and visualized by the implemented code.	13
6	Overlaying different GIS class exclusion masks resulting in a potential areas layer. Figure from WINNIEPOT study [8].	20
10	Same area for both approaches to compare the precision.	24
11	Small GIS object example.	26
12	The potential area simulation with data from OpenStreetMap compared to the Herzogenrath potential area report [3]. Only areas with a square meter size higher than 10000 are shown.	27
13	Wind exclusion area showing winds lower than 7 m/s from GWA. . . .	28
14	Runtime comparison between all results.	31
15	Error in the optimal positioning algorithm, where the leftmost ellipse is not expanded to its maximum.	42
16	Comparison between hexagon pattern (left) and greedy ellipse (right). .	43
17	Comparison hexagon pattern to ellipse packing.	45
18	Comparison of found turbine amount of the ellipse and pattern based algorithm for each area.	48
19	Comparison of Annual Energy Production of the ellipse and pattern based algorithm for each area.	48
20	Comparison of Annual Energy Production per turbine of the ellipse and pattern based algorithm for each area.	49
21	Comparison of wake loss of the ellipse and pattern based algorithm for each area.	49
22	Comparison of LCOE of the ellipse and pattern based algorithm for each area.	50
23	Comparison of the performance of the ellipse and pattern algorithm. .	51
24	Performance of optimal positioning attractiveness.	52

25	Comparison of Annual Energy Production of the ellipse and attractiveness algorithm and windPRO.	53
26	AEP error per turbine and total percentage deviation to the attractiveness matrix algorithm. Positive values mean that attractiveness is better.	54
27	Wake loss of the placements	54
28	Comparison of LCOE of the ellipse and attractiveness algorithm and windPRO.	55
29	Optimal placements of the potential areas of Herzogenrath project. The displayed ellipse size is the global maximum ellipse size. The main wind direction is 30°.	57
30	Optimal placements of the potential areas of the Herzogenrath project, calculated with greedy attractiveness algorithm. The color shows the attractiveness score.	58
31	Optimal placements of the potential areas of the Herzogenrath project, calculated with AEP in windPRO. The color represents the average wind speed.	59

List of Listings

1	Pseudocode for the second approach of the potential area calculation. .	23
2	Finding hull of MultiPolygon	25
3	Clustering areas together, which are too close to each other	37
4	Optimal positioning algorithm shortened	39
5	Expanding the global ellipse size to its maximum	40
6	Expanding the individual ellipse sizes to their maximal size	41