Final-year Project on the Optimisation of Turbine Placements within Wind Farms

Bernadette Christine Browning

April 8, 2025

1 Introduction

My motivation for this project was the ever-present need for more efficient renewable energy in the wake of the global energy crisis and climate change [1]. Chen states that, by embracing more efficient energy production, greenhouse gases can be reduced by up to 40% [1]. The world economy relies so heavily on readily-available energy, that reducing our energy usage is often infeasible [1]. However, a compromise can be reached through an increased use of renewable energy and optimising renewable energy production.

The energy crisis is readily observed in some countries such as Libya [2] and Nigeria [3]. Without access to reliable electrical power, the economic growth of these countries is stunted, and the quality of life for the countries' citizens cannot improve [3] [2].

One way to improve the availability of energy in way the limits long-term environmental impact is to embrace wind energy production [2]. The cost of

doing so is always a concern, especially in less economically developed countries. As such, optimisation of wind farms to produce the highest possible power, at the lowest possible cost, is essential. One way to achieve this is by optimising the positioning of turbines within wind farms to decrease the number of necessary turbines [4]. To address the optimisation of turbine placements within wind farms, I wrote a function in Python that generates discrete representations of wind farm layouts in the form of a 10 x 10 matrix and a basic wake model. This code was fed into a genetic algorithms, NSGA-2 and SPEA2, using Pymoo to optimise the positioning of the turbines to achieve the best power output.

I began my reading with a comprehensive article, Multi-Objective Optimisation of the Benchmark Wind Farm Layout Problem, written by P.L. Manikowski et al. [4] This gave me an excellent overview of the topic. Another article that is hard to avoid when reading about this topic is the paper by Mosetti et al. entitled "Optimization of wind turbine positioning in large wind farms by means of a Genetic algorithm" [5]. It is cited in most articles that mention the wake effect because it was published early on within this field of research.

2 Betz's Law

Betz's law describes the conversion of the kinetic energy of wind flowing through turbine blades. Betz's law relies on several assumptions made about the wind turbine, as mentioned by Ragheb [6]:

1. "It does not possess a hub;

2. It possesses an infinite number of rotor blades which do not result in any drag resistance to the wind flowing through them;

3. Uniformity is assumed over the whole area swept by the rotor, and the speed of the air beyond the rotor is considered to be axial. The ideal wind rotor is taken at rest and is placed in a moving fluid atmosphere."

The velocity of air leaving the back of the rotor blades cannot equal the velocity of air entering the blades [7]. This is because a 'fully efficient' wind turbine would lead to a stagnation of air behind it, and thus no further air could pass through [6]. This means that there is an optimal velocity at which air should exit the back of the rotor blades to achieve an optimal energy conversion [7]. The ratio of optimal velocities for power conversion is defined by Blackwood [7] as

$$v_2 = \frac{1}{3} \cdot v_1, \tag{1}$$

where v_1 is the initial velocity of air entering the rotor blades, and v_2 is the velocity of air exiting the rotor blades.

It was useful to consider Betz's Law when calculating the potential velocities of turbines within my wind farm simulation, as it set an upper limit for each turbine's power output. There is a paper by Strauss [8] that claims to have 'broken' this limit written by disregarding "unneeded assumptions in Betz's Law". This theory is supported by other researchers [9] [10] who have taking the aerodynamics of turbine rotors into account, along with a holistic overview of the entire wind farm and its environmental conditions, as opposed to individual turbines in a potentially overly-simplified environment.

For the purposes of my project code and wind farm simulation, I have assumed that Betz's Law holds.

3 The Wake Effect

Arguably, the most important thing to consider for my project was the wake effect produced by each turbine. This is a concern in the design of wind farms because placing turbines too close to each other can reduce the overall efficacy of the wind farm by up to 30% [4]. I used the Jensen model to create my own wake model in Python because it is widely used and recommended for its performance [11] [12]. I kept my model simple by considering an adaptation of the single-wake model with the wind moving from North to South of the wind farm. Hence, the wake radius was calculated using the equation [4]

$$R = kx + r. \tag{2}$$

The full implementation of this, with the entirety of my project's code, can be seen via the link to my Google Colab notebook in Appendix A. The code became more elegant as my confidence in using Python grew. I coded two implementations of the Jensen wake model.

The initial implementation was around 1400 lines of code. In both implementations, the wind farms were represented by matrices of 0s and 1s, corresponding to the absence and presence of a turbine, respectively. For example, a 30-turbine wind farm may look as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Whilst velocity arrays (see Step 8 below) for the same problem, would like something like this:

$$\begin{bmatrix} v_1 & 0 & 0 & v_1 & 0 & 0 & 0 & 0 & v_1 & v_1 \\ v_2 & v_2 & 0 & 0 & 0 & 0 & v_1 & v_2 & 0 & v_2 \\ 0 & v_3 & v_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & v_4 & 0 & 0 & v_1 & v_1 & 0 & v_1 & 0 & v_1 \\ 0 & 0 & 0 & v_2 & 0 & v_2 & v_2 & 0 & 0 & 0 \\ v_1 & v_1 & 0 & 0 & 0 & 0 & 0 & 0 & v_1 & 0 \\ 0 & 0 & v_2 & 0 & 0 & 0 & v_1 & v_2 & 0 & 0 \\ 0 & v_3 & 0 & v_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_4 & v_4 & 0 & v_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

where i represents the number of turbines upwind from that turbine.

Notice that the velocities of turbines positioned more than one row and/or one column apart, are not affected. This is because of the incorrect calculation of the wake radius.

The code was structured as follows:

Step 1: Store the coordinates of turbine placements from a randomly generated array;

Step 2: Define variables used to calculate the wake region in their original units;

Step 3: Create lists for each row of the turbine array, populated with the turbine coordinates, then extract only their x-coordinates;

Step 4: Create lists for each row of the array, populated with the distances between individual turbines and each turbine in the proceeding row;

Step 5: Compare the distances between these turbines with a constant wake radius. Create lists populated with the x-coordinates of each turbine affected by wake in each row, and the distances between them;

Step 6: Create a 10×10 array populated with the positions of the affected turbines;

Step 7: Repeat steps 3 to 6 until there is, at most, one affected turbine;

Step 8: Calculate the velocity behind each 'layer' of affected turbines, and multiply the layout array by this value, to represent individual turbine power outputs;

Step 9: Calculate the wind farm's total power output by summing the values of the power-representation array.

I used the following turbine properties laid out by Mosetti [5] to calculate the wake radius of my turbines:

Property	Value
Thrust coefficient (C_t)	0.88
Axial induction factor (a)	0.3267949192
Rotor radius (r_0)	20m
Rotor diameter (D)	40m
Height of wind turbine (z)	60m
Roughness of terrain (z_0)	0.3m
Wake decay factor (k)	0.0943695829
Air density (ρ)	1.225kg/m^3
Power coefficient (C_p)	0.39
Swept area (A)	1256.64m^2

Table 1: This table is taken from Manikowski's paper [4], with the value for z_0 corrected from 0.03m to 0.3m (personal communication by Manikowski). Here, $k = \frac{0.5}{\ln \frac{z}{z_0}}$.

My second implementation contains around 60 lines of code. It uses a more accurate representation of how wake interactions affect turbine velocities in a few ways. First, it replaces the previous use of x as a constant with the variable x. This is the correct implementation, as used in Manikowski's paper [4]. This was necessary because the x-coordinate distance between turbines is not fixed, even if the minimum x-distance between any potential turbine coordinates is. Secondly, the wake radius is calculated differently. The variables are no longer defined directly in coordinate units - they are converted after the original wake radius is calculated. I attempted this in my original Jensen wake model implementation, but incorrectly. Thirdly, the wake radius is calculated using SciPy's 'cdist' function.

'cdist' is a SciPy function that calculates the pairwise Euclidean distances

between each turbine. That is to say,

$$E_{ij} = d(X_i, X_j) = \|X_i - Xj\|_2 = \sqrt{\sum_{k=1}^{n}(X_{ik} - X_{jk})^2}, \text{for all } i, j \in \{1, ..., m\}, \tag{3}$$

where X is an $m \times n$ matrix. For calculating pairwise Euclidean distances between turbines in a 10×10 array, $m = n = t$, where t is the number of turbines. One perk of using the E_{ij} matrix to calculate turbine distances, is the ease with which the matrix is read. E_{ij} is a symmetric matrix with zeros on its diagonal, called a zero-diagonal symmetric matrix. It's symmetry means $E_{ij} = E_{ji}, \forall\, i, j \in \{1, ..., t\}$, where E_{ij} is the matrix entry corresponding to the distance between turbines i and j.

In summary, this second implementation was structured as follows:

Step 1: Store the coordinates of turbine placements from a randomly generated array;

Step 2: Define variables used to calculate the wake region in their original units;

Step 3: Define the wake region and wake effect on velocity;

Step 4: Convert the wake region into 'coordinate units';

Step 5: Calculate the Euclidean distances between all turbines;

Step 6: Generate arrays of turbines affected by the wake from other turbines;

Step 7: Create a matrix of power outputs from each turbine;

Step 8: Calculate the wind farm's power output by summing all turbine outputs.

The changes I made to my code show an accurate representation of my progress in relearning Python. When creating it, there were a few things to consider. Choosing a suitable optimiser required realising the nature of my optimisation problem. Initially, as one can see in my code, I used an

optimisation package called 'Gekko' [1]. This was less suitable for my problem, as it does not handle non-convex problems.

4 On the Nature of Non-Convex Problems

Non-convex problems are those with local minima that can cause an optimiser to confuse them with a global minimum. Squared and square-root terms are in the definition of the wake radius, both of which can cause an optimisation problem to become non-convex. When the optimisation problem contains a square-root term, the optimiser can become 'stuck' at the local minima and consequently does not explore the whole of the available search space. Squared terms can also cause an optimisation problem to become computationally expensive ($\mathcal{O}(n^2)$), depending on the code's implementation, i.e. if the term is contained within nested loops. Freund [14] writes that "it is rare that an algorithm for NLP will compute a global minimum", where NLP refers to nonlinear programming. Fruend goes on to state that algorithms will generally approach the global minimum given sufficient iterations. This is a conclusion that one can interpret from the results of most nonlinear optimisation problems, including my own optimisation problem when comparing the final function values for an increasing number of iterations/generations.

I have included the function values produced by the NSGA-2 and SPEA2 algorithms (and their corresponding values for the power output of my wind farm simulations) for 10, 20, and 30 generations, respectively. Corresponding Pareto Front approximations and the best layouts found for each instance can

[1] `https://gekko.readthedocs.io/en/latest/`, citation: [13]

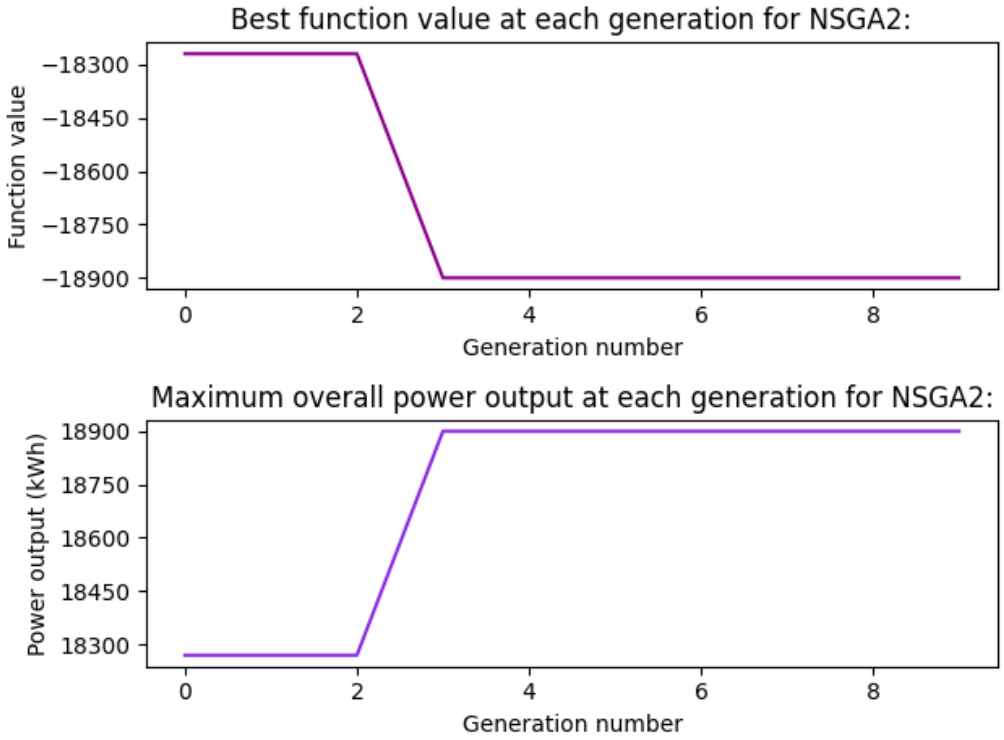

Figure 1: Best function values and power outputs obtained with a population of 50 and 10 generations, using NSGA-2.

be found in sections 6.1 and 6.2.

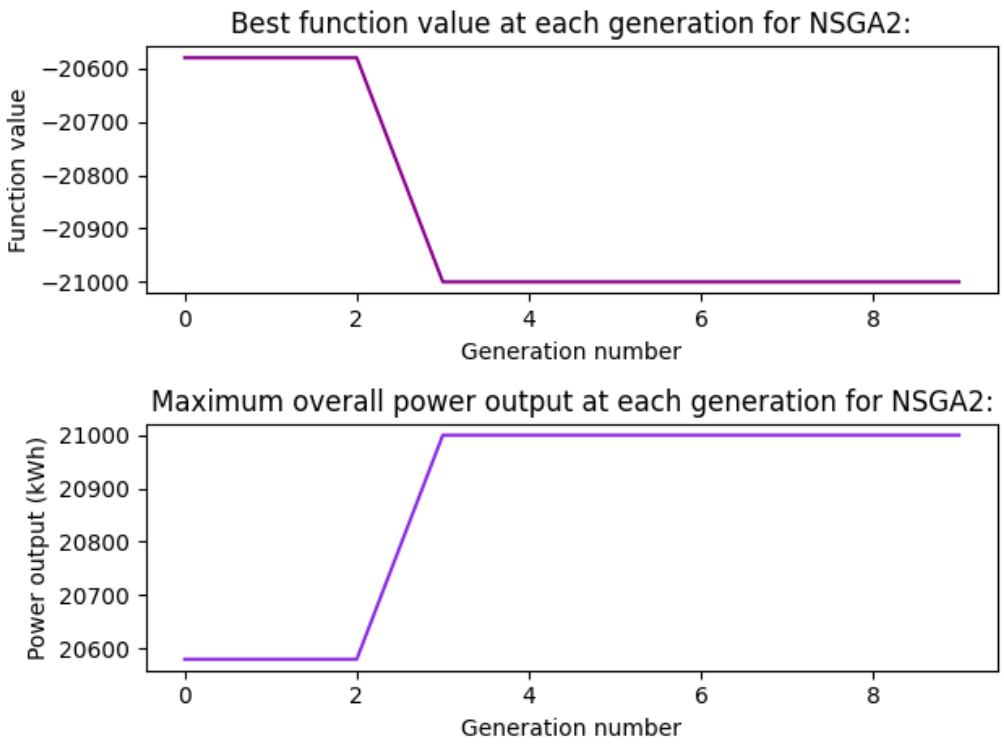

Figure 2: Best function values and power outputs obtained with a population of 100 and 10 generations, using NSGA-2.

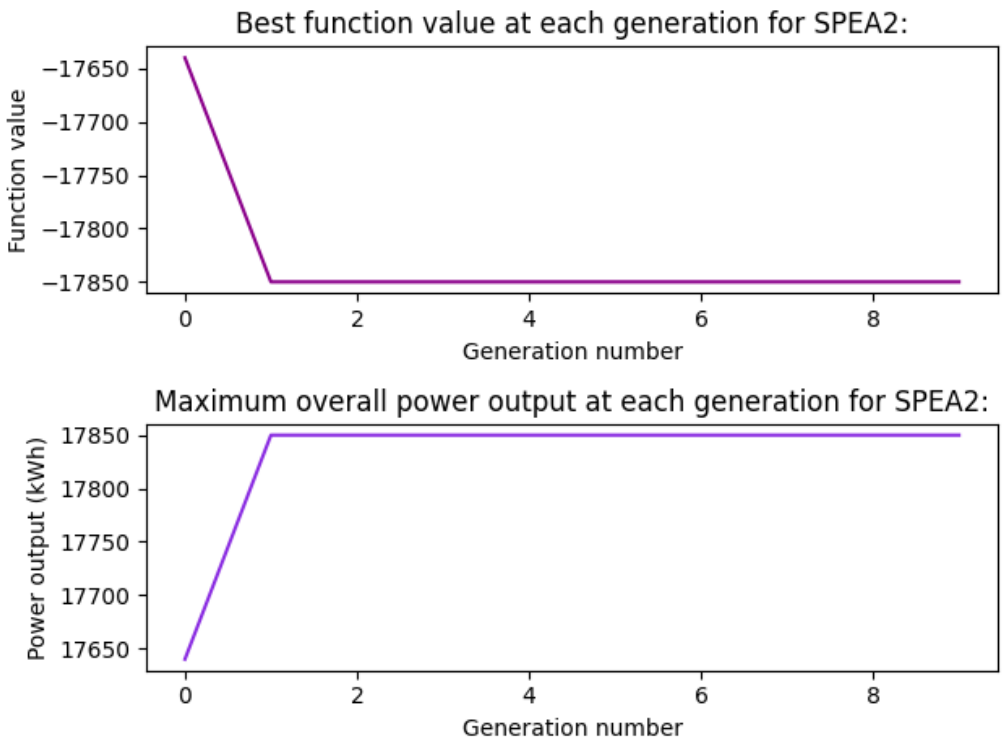

Figure 3: Best function values and power outputs obtained with a population of 50 and 10 generations, using SPEA2.

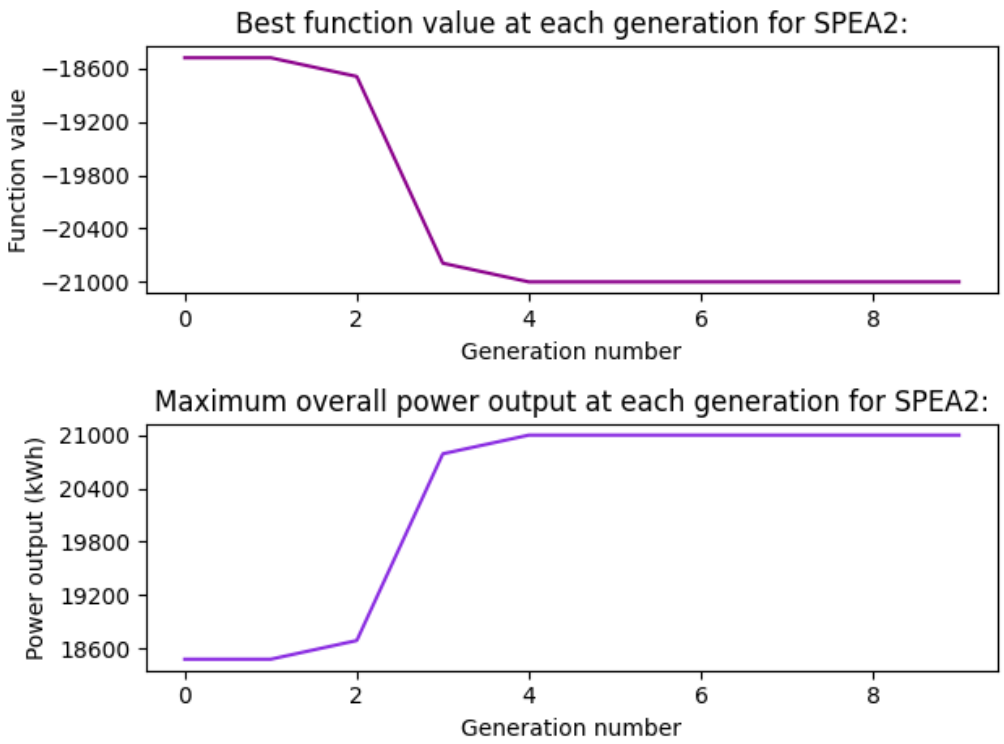

Figure 4: Best function values and power outputs obtained with a population of 100 and 10 generations, using SPEA2

At 10 generations, one can observe that the algorithms converge early. This is a sign that an insufficient number of generations is being used, and hence only a local optimum is reached because the search space is too small.

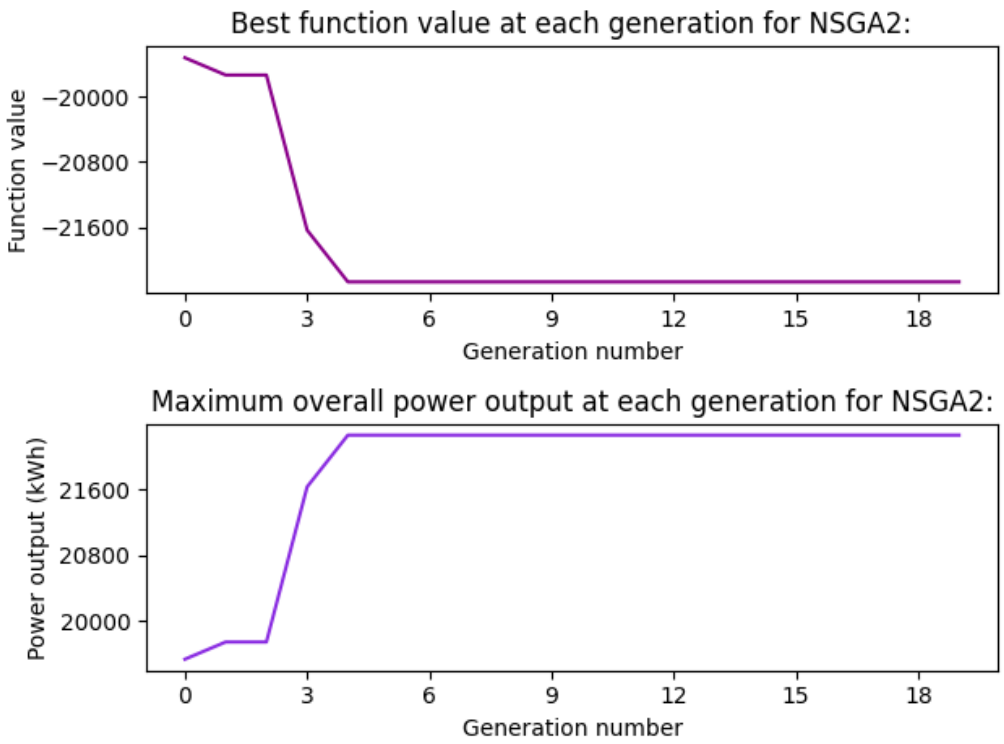

Figure 5: Best function values and power outputs obtained with a population of 200 and 20 generations, using NSGA-2.

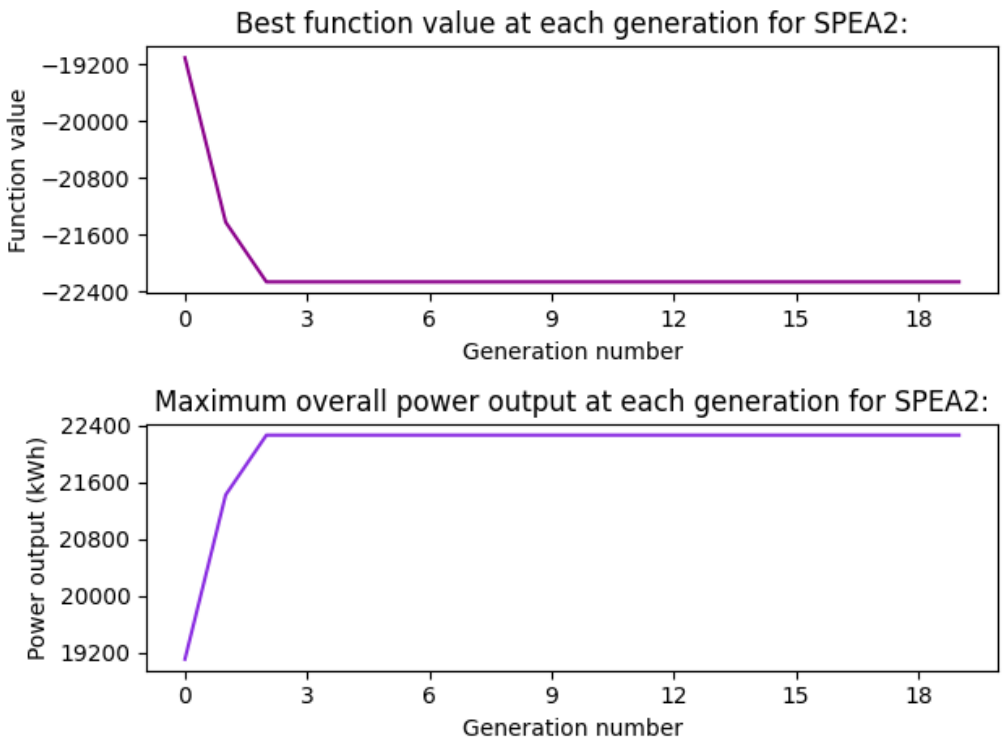

Figure 6: Best function values and power outputs obtained with a population of 200 and 20 generations, using SPEA2.

A short aside about these graphs is the nature of their increase in function value/power output over the number of iterations. There are steep 'jumps' in convergence as well as gradual increases. The 'jumps' occur because of the crossover and the gradual increases occur due to mutation. This is because more significant changes to the variables (the positions of the turbines) can occur during crossover than during mutation.

To understand the tendency for optimisers to become stuck at local minima, it is useful to recall a few definitions. Freund [14] defines the following equations 4, 5, and 6. The definition of a local minimum is

$$\bar{x} \in \digamma \text{ if } \exists \varepsilon > 0 | f(\bar{x}) \leq f(x) \forall x \in B(\bar{x}, \epsilon) \cap \digamma, \tag{4}$$

where $B(\bar{x}, \epsilon)$ is the closed ball which encircles the local minimum at its centre, with radius ϵ. This allows an optimiser to get arbitrarily close to the local minimum within its radius. The definition of a global minimum is

$$\bar{x} \in \digamma \text{ if } f(\bar{x}) \leq f(x) \forall x \in \digamma. \tag{5}$$

The feasible region, $\digamma$, for both the local minima and global minimum is defined as

$$\digamma = \{x | g_i(x) = 0 \text{ for } i \in \varepsilon, g_i(x) \leq 0 \text{ for } i \in I\}, \tag{6}$$

where ε is the set of indices in the equation for the problem's equality constraints, and where I is the set of indices in the equation for the problem's inequality constraints [14].

In equation 4, we see the space in which local minima exist contains the ball $B(\bar{x}, \epsilon)$ which allows the optimiser to get arbitrarily close to the local minimum, making it more likely that a local minimum is found before the global minimum.

How these definitions apply to my optimisation problem is seen by the equivalence between sums and integrals. Recall that

$$\int_a^b f(x)\,dx = \lim_{\delta x \to 0} sum_{x=a}^{b} f(x)\delta x. \tag{7}$$

Using this, we consider my implementation of the Jensen wake model and the calculations of wind velocity directly behind each turbine. In my wake model implementation, the total power output (in kWh) is calculated by summing the power output of each turbine in the array. This can be written as

$$\int_{x=0}^{x=9}\int_{y=0}^{y=9} f(x,y) = \lim_{\substack{\delta x \to 0,\\ \delta y \to 0}} \sum_{\substack{0 \le x \le 9,\\ 0 \le y \le 9}} f(x,y)\delta x \delta y, \tag{8}$$

where $f(x, y)$ represents the function "output(g)" in my project code, which calculates the wake experienced by turbines and their power output.

My wake model implementation contains only equality constraints. This means that the feasibility region is defined as,

$$F = \{x | g_i(x) = 0 \text{ for i} \in \varepsilon\}. \tag{9}$$

The local minima and global minimum are defined for my implementation as in equations 4 and 5, respectively.

5 Implementation of Optimisation Algorithms

I used Pymoo to implement my optimisation problem. Pymoo has a structure to its implementation which I was able to customise to my problem's needs, by customising its inbuilt methods. Using a custom sampling method ensured some element of randomness in the algorithms selection by preventing biased samples. Using custom mutation and crossover methods allowed the optimiser to escape local minima by increasing the search space.

5.1 Sampling Method

My sampling method produces a randomised array of 0s and 1s representing the absence and presence of turbines, respectively. The method used to achieve this is as follows:

Step 1: Choose a random number of turbines $t \in [30, 60]$;

Step 2: Choose a random x and y coordinate;

Step 3: Check if these coordinates have already been chosen;

Step 4: Assign non-duplicate coordinates to the array.

I chose the interval $t \in [30, 60]$ based on my requirements for the total power output of the wind farm. I wanted the output to be $\geq 10,000$ kWh. Assuming a maximum output per turbine of 630 kWh, the minimum number of turbines needed to achieve this (when rounded to the nearest integer) was 16. After calculation of the wake effect, not all turbines would be producing power and would effectively cease to exist in my calculations. To mitigate the limiting impact of this on the overall power output of the windfarm, I initialised the number of turbines at 30. I limited the number of turbines

to 60 because a turbine array that was too densely packed would cause the majority of turbines to experience a significant wake effect. I knew from reading Manikowski's [4] paper that the optimal number of turbines was in the range of 30 to 40, hence I would not be gaining analytically-important data by sampling populations that were significantly larger.

5.2 Mutation Method

I created a custom mutation method to move turbines to new positions depending on an improvement in total wind farm power output. This is an approach similar to that of a hill-climbing algorithm. The logic of my custom mutation method was as follows:

Step 1: Calculate the output power of the wind farm 'g';

Step 2: If the output of power (as calculated by my wake calculation function) is below a specified threshold, turbines are reassigned to available locations within the array;

Step 3: Recalculate the output power of the new wind farm 'g';

Step 4: If the new output exceeded a specified threshold, the loop is exited.

5.3 Crossover Method

My custom crossover method took two 'parent' arrays and from them, created two 'offspring' arrays. I compared each pair of parents, in order, to select for either an improvement in the total wind farm power output, or fewer turbines. This is an implementation of weak Pareto dominance, where one parent has one or both of the objective values considered more suitable than

the other parent.

The method was implemented as follows:

Step 1: If the algorithm is on it's first generation, even entries of each parent array are swapped;

Step 2: Otherwise, if parent 'a' has one or both objective function values better than parent 'b', both parent arrays are set equal to parent 'a';

Step 3: If parent 'a' is not weakly Pareto dominant, every other entry in each parent array is swapped such that the swapped entries are in position $100 - i$, where i is the index of the entry to be swapped.

5.4 Limitations

There were some limitations that emerged from the way my code was implemented.

The main issue was its runtime. Although I utilized parallel processing and pre-calculated computationally expensive functions, like scipy's 'cdist' function, the optimisation code called functions within functions, and contained nested while loops and for loops. This complexity meant that the optimisation took a long time to complete for higher population and generation sizes. In particular, their was a direct relationship between the generation number and the time that generation took to complete. There were ways I could have avoided this if I had more time to implement them; I shall go through these in the Future Considerations section: 7.5.

The way I got around this issue was by using low population sizes and

a low number of generations. The side effect of lower population sizes was less diverse individual solutions as there were fewer potential mating pairs. Fewer generations potentially caused the algorithms to terminate before they converged to an optimal solution.

6 Comparison of Optimisers

From Baptista's paper [15], we are given a succinct comparison of various optimisation algorithms applied to the problem of placement of wind turbines and power substations. I edited it for clarity below.

Year	Ref.	Algorithm	% off optimal solution	Wake Effect	Sub-stations	Power Losses
2023	[16]	MILP AOC AOCsp	0 +0.4 to +7.6 +0 to +1.4	No No No	Yes Yes Yes	Yes Yes Yes
2023	[17]	ILP	0	No	Yes	Yes
2022	[18]	PSO	Doesn't guarantee optimal solution	Yes	No	No
2022	[19],[20]	MILP	0	Yes	No	No
2022	[21]	ML/DNN	Doesn't guarantee optimal solution	Yes	No	No
2021	[22]	ILP	0	No	Yes	Yes
2019	[23]	MILP	0	Yes	No	Yes
2018	[24]	MILP	0	Yes	Yes	Yes
2018	[25]	GA	Doesn't guarantee optimal solution	Yes	No	No
2017	[26]	GA/PSO	Doesn't guarantee optimal solution	Yes	Yes	Yes
2016	[27]	ILP	0	No	Yes	Yes

Table 2: Comparison of papers reviewed by Baptista [15].

The columns on "substations" and "power losses" can be ignored for this project write-up, as they are not included in my optimisation problem.

The optimisers I used to achieve my final results in Python were NSGA-2 and SPEA2.

6.1 NSGA-2

NSGA-2 stands for Non-dominated Sorting Genetic Algorithm. I used NSGA-2 for a couple of reasons. When creating my optimisation problem, I noticed the similarity of my objectives to that of the binary knapsack problem. Owing to its binary variables, the knapsack problem can be solved using Pymoo's elementwise problem definition, and NSGA-2 was suitable for implementing this because of its ability to effectively optimise for two min/max functions.

As described by Holland [28], genetic algorithms apply the biological concepts of gene mutation and inheritance to optimisation problems, with the fittest solutions that survive into the next generation. Holland describes the advantage of NSGA-2's use of mutation in preventing the optimiser from becoming 'stuck' in local optima. Mutations achieve this by increasing the search area of solutions, thus 'jumping' over local optima.

That NSGA-2 is non-dominated means that it sorts its solutions based on how far they are from the problem's Pareto Front [29]. A Pareto Front is the collection of solutions that produce optimal trade-offs between two potentially conflicting optimisation functions. It often forms a quadratic curve when the problem is two-dimensional. When sorting its solutions, NSGA-2 ranks them in groups but retains only those closest to the problem's

Pareto Front for use in the next generation. These solutions are described as 'non-dominated' solutions.

A solution x_1 is described as 'non-dominated' if it fulfils the following condition,

$$f_i(x_1) > f_i(x_2) \text{ and } f_i(x_1) \geq f_i(x_2), \text{ for } i, j = 1, 2, i \neq j, \tag{10}$$

where f is an optimisation function. I have only defined a non-dominated solution for functions f_1, f_2, since NSGA-2 is most efficient when optimising for two functions.

After NSGA-2 sorts for non-dominated solutions, it sorts them again, according to crowding distance [29] [4]. This value is determined by the hypervolume indicator [4]. As in Manikoski's paper, the reference points for the most and least optimal solutions are the maximum number of turbines and the least number of turbines, respectively. In contrast to Manikowski's reference points, which are calculated by optimising cost and power, my reference points are calculated by optimising the number of turbines and power.

When these reference points are known or can be estimated, one can produce a graph showing the Pareto Front approximation and, using this curve, calculate the area of the feasible region as defined in the "On the Nature of Non-Convex Problems" section: 4. Points at the extreme ends of the Pareto Front approximation are kept in all generations [29].

Figure 12 shows a Perto Front approximation that roughly follows the curve expected for two competing objectives. However, most of the points are

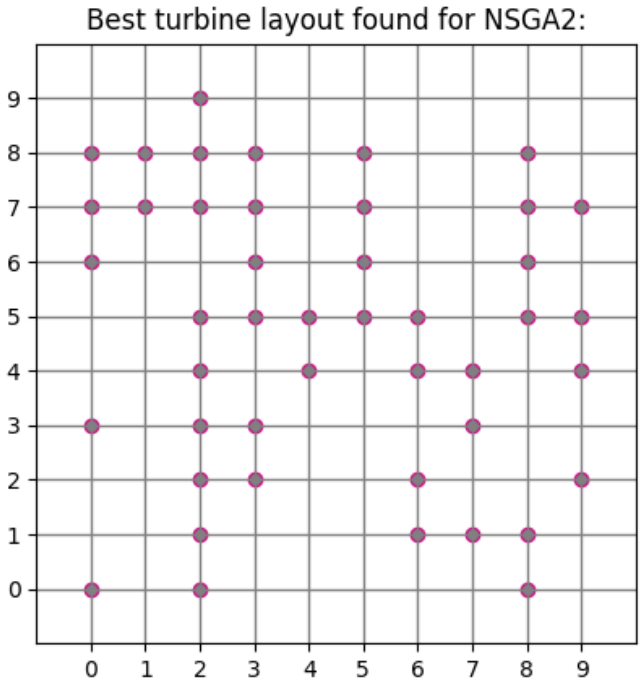

Figure 7: Best layout obtained with a population of 50 and 10 generations, using NSGA-2.

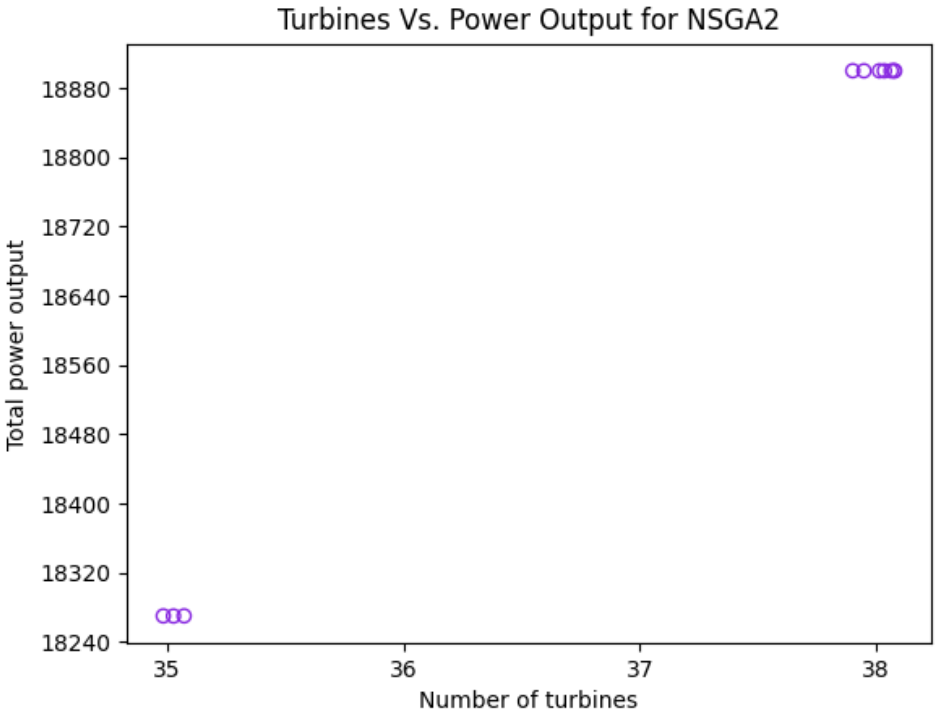

Figure 8: Pareto Front approximation obtained with a population of 50 and 10 generations, using NSGA-2.

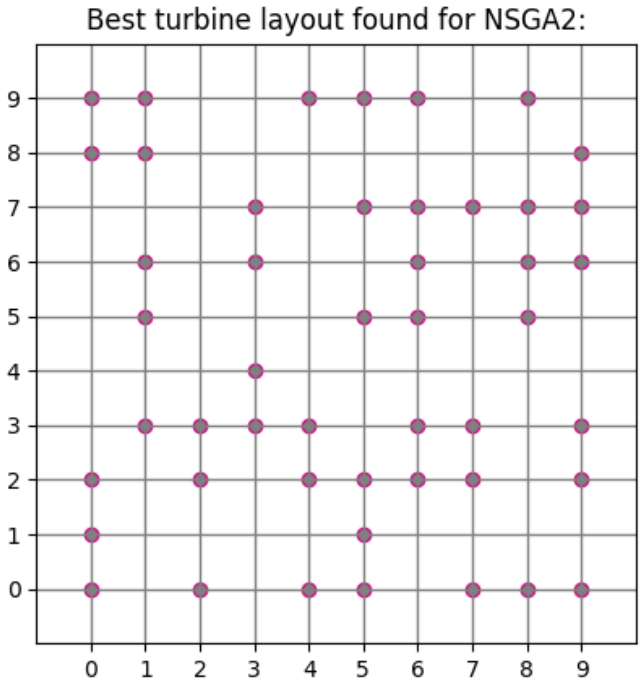

Figure 9: Best layout obtained with a population of 100 and 10 generations, using NSGA-2.

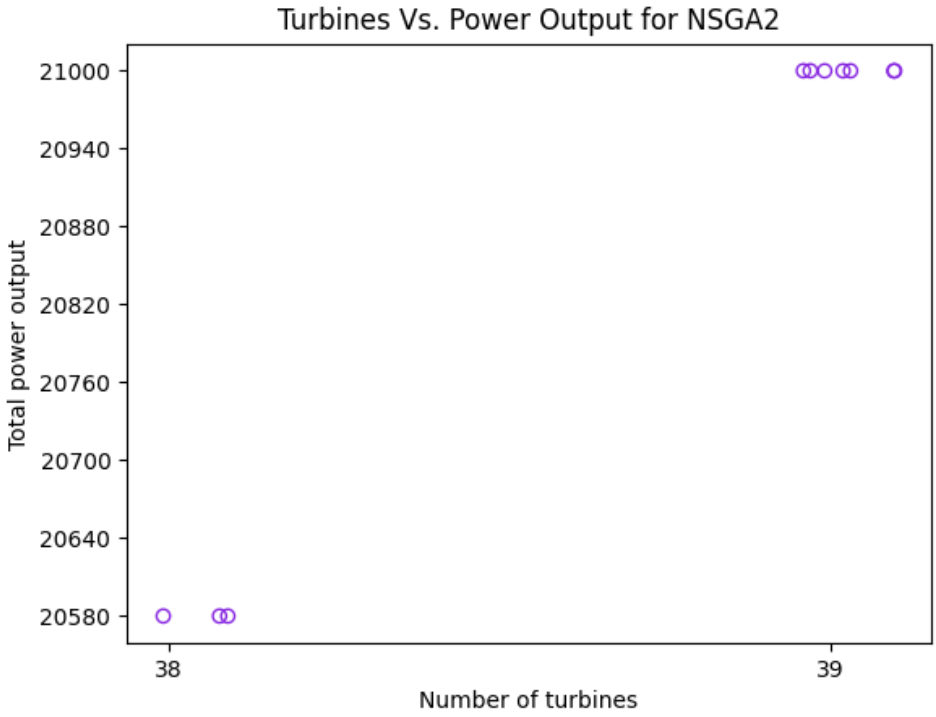

Figure 10: Pareto Front approximation obtained with a population of 100 and 10 generations, using NSGA-2.

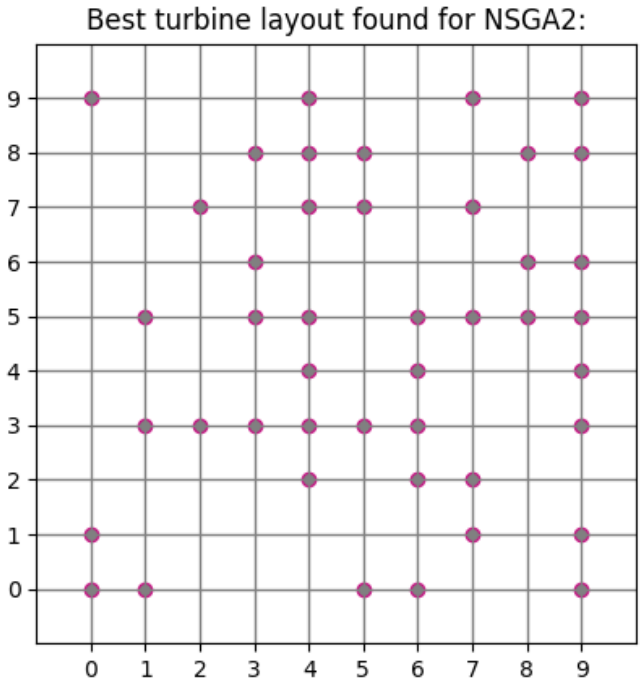

Figure 11: Best layout obtained with a population of 200 and 20 generations, using NSGA-2.

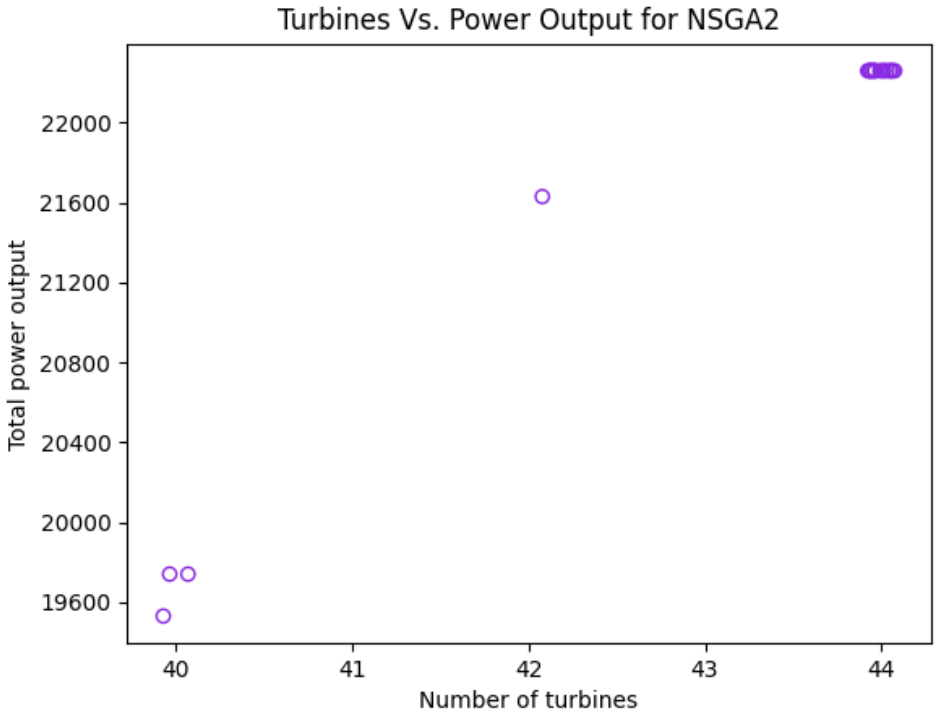

Figure 12: Pareto Front approximation obtained with a population of 200 and 20 generations, using NSGA-2.

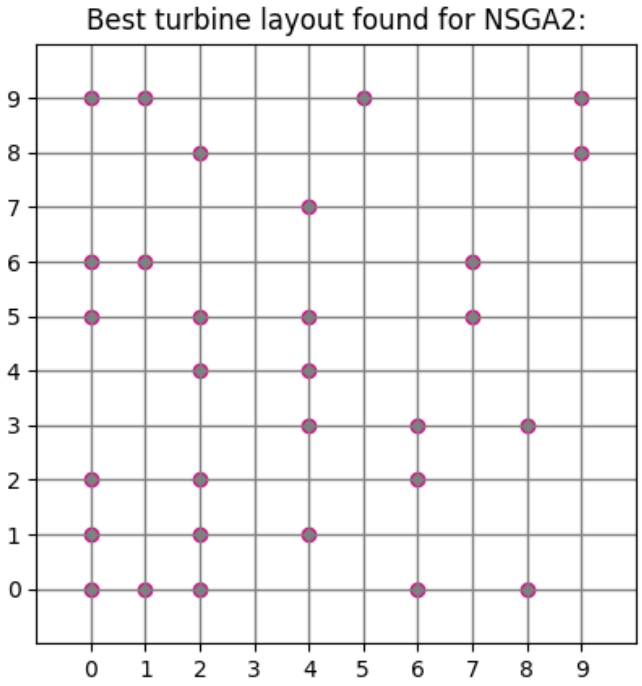

Figure 13: Best layout found obtained using a population of 50 and 30 generations, using NSGA-2.

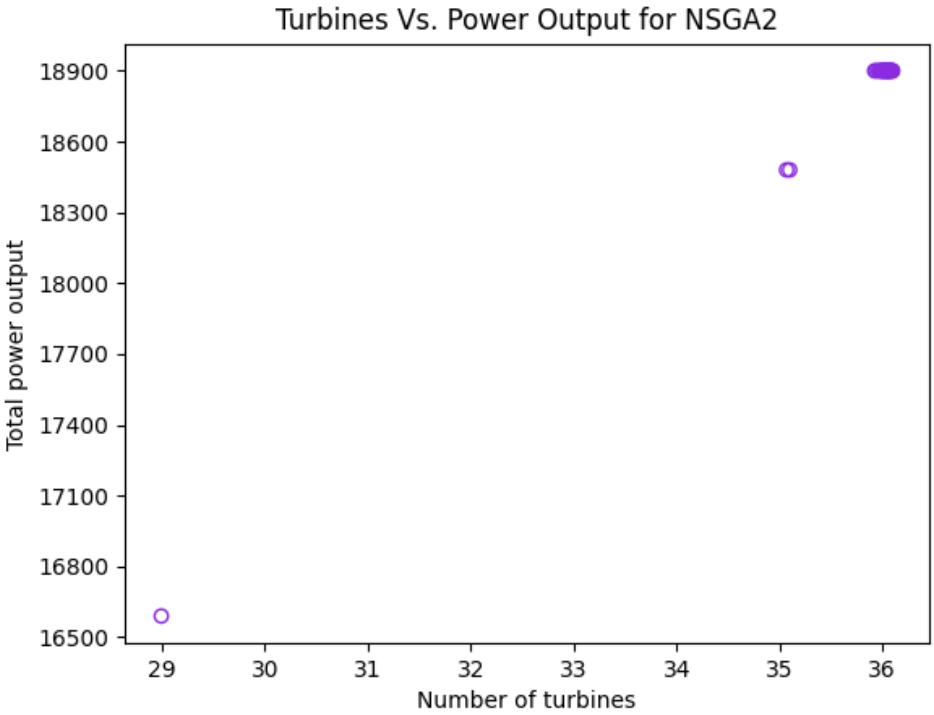

Figure 14: Pareto Front approximation obtained with a population of 50 and 30 generations, using NSGA-2.

clustered at the top-right of the graph. This can be explained by considering the effect of local minima on the search ability of NSGA-2. Figures 10 and 8 have their points clustered at the bottom-left and top-right of their graphs. This is not overly surprising, given the small range of turbines on their x-axes.

Figure 13 shows a layout with the most similarities to one found by Emami et.al. [30]. This is a positive sign that my optimisation problem is being implemented suitably. Perhaps with further refinement, I could recreate similar layouts more consistently.

6.2 SPEA2

SPEA2 stands for Strength Pareto Evolutionary Algorithm, and as the name implies, uses Pareto dominance in its selection of individuals. It has three main stages in its selection process [31]:

1. Fitness assignment using a strength value based on the number of solutions over which an individual has Pareto dominance;

2. density estimation (similar to crowding distance employed in the NSGA-2 algorithm);

3. comparison against an archive of other solutions.

The biggest difference between SPEA2 and NSGA-2 algorithms, is the archive of non-dominated solutions at each generation performed by SPEA2, that exists alongside the regular population. This allows for individual solutions to survive even if they do not go on to reproduce [32]. During selection,

the regular population is compared with the archive. This accounts for the effects of randomness introduced by crossover and mutation that may produce offspring with a lower fitness value.

Although NSGA-2 and SPEA2 both have their own predefined crossover and mutation methods, due to my use of custom crossover and mutation classes, my optimisation problem will not have been affected by these differences.

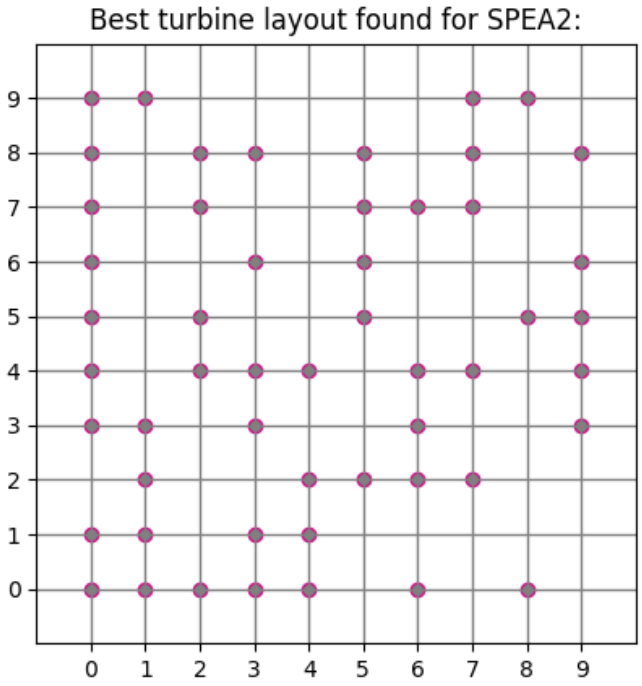

Figure 15: Best layout obtained with a population of 50 and 10 generations using SPEA2.

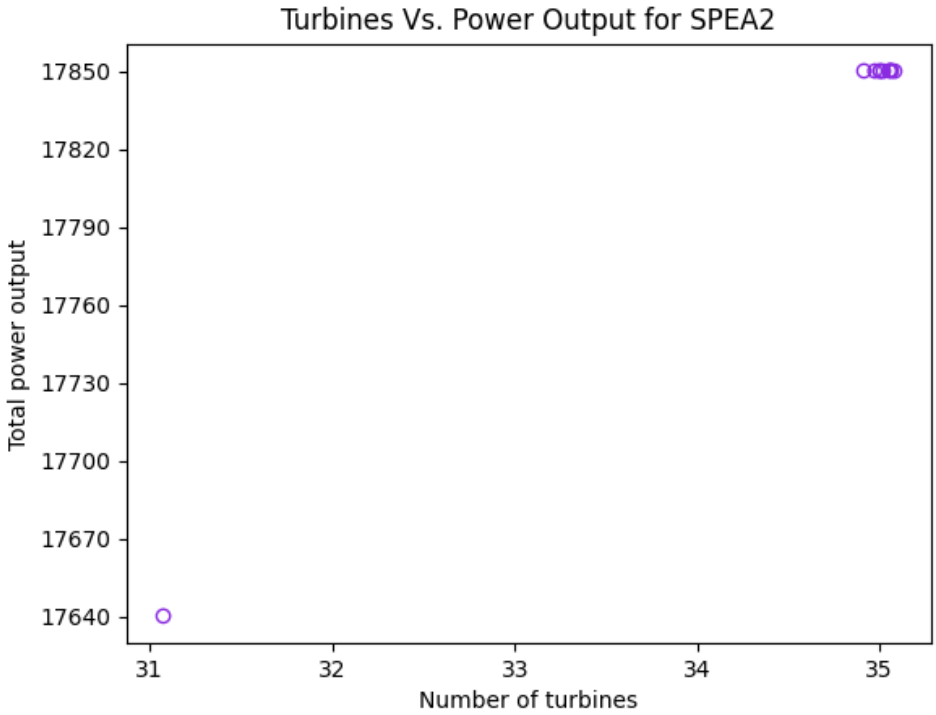

Figure 16: Pareto Front approximation obtained with a population of 50 and 10 generations using SPEA2.

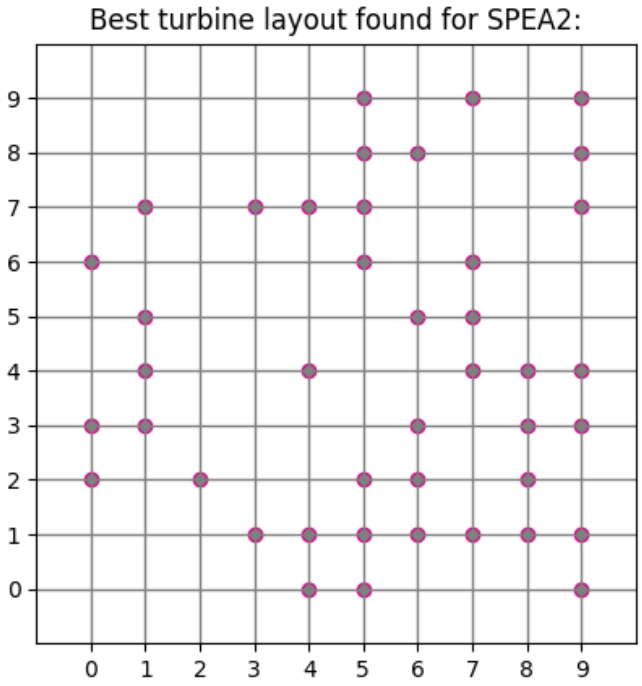

Figure 17: Best layout obtained with a population of 100 and 10 generations, using SPEA2.

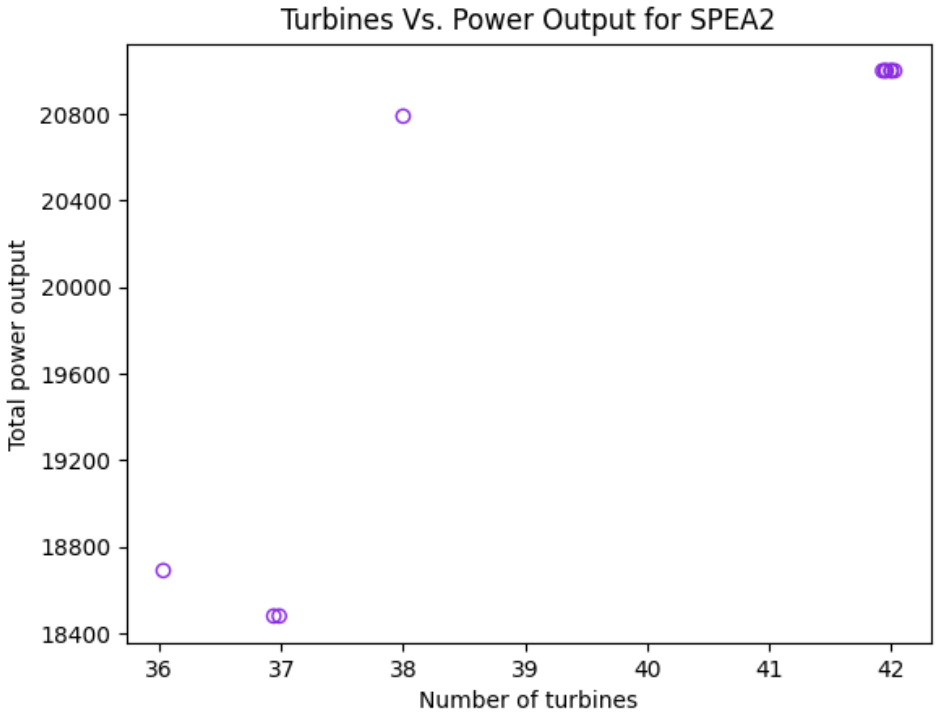

Figure 18: Pareto Front approximation obtained with a population of 100 and 10 generations using SPEA2.

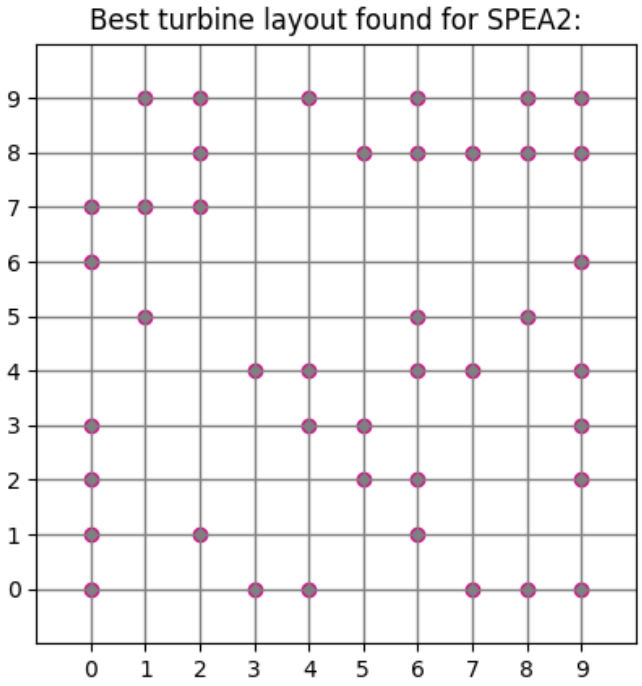

Figure 19: Best layout obtained with a population of 200 and 20 generations using SPEA2.

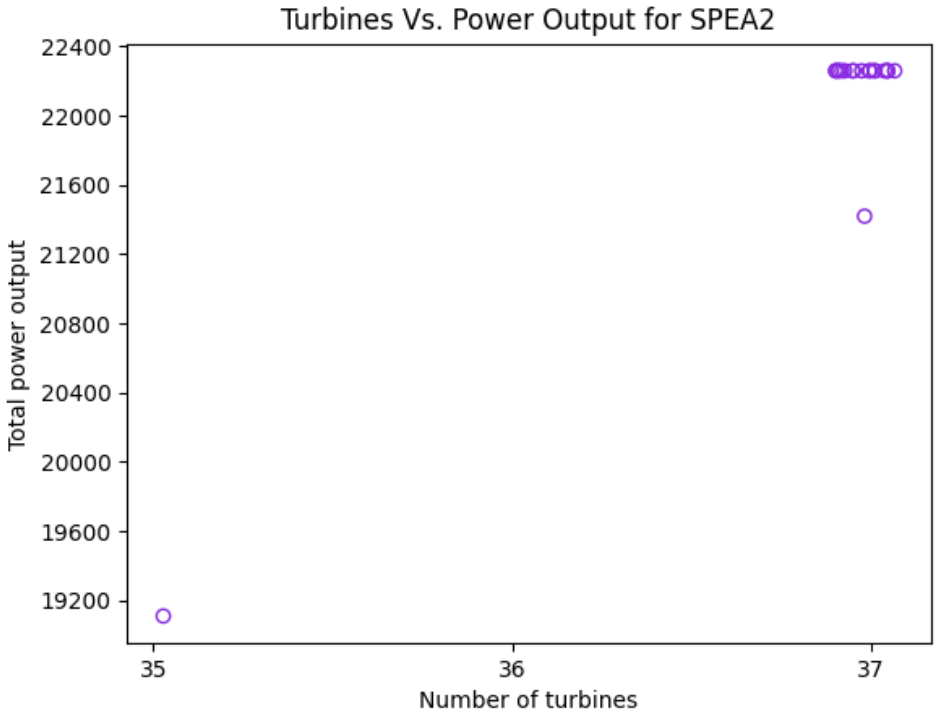

Figure 20: Pareto Front approximation obtained with a population of 200 and 20 generations using SPEA2.

The Pareto Front approximations using SPEA2 are obtained with a small range of values for the number of turbines. There are two possible reasons for this. The first is the dynamic constraints on the number of turbines in my evaluation class. I wrote the evaluation class of my optimisation problem such that solutions with too few turbines would be discarded, based on a definition linked to the current generation number expressed as a percentage of the maximum number of generations. This definition allowed me to tighten the constraint as the algorithm progressed, allowing more solutions to survive initially, thus improving their diversity. The unintended consequence of this constraint was solutions were less diverse due to their similar turbine counts.

Related to this is the second possible reason relating to the inherent issues of SPEA2 with local search [33]. When the search area is too small (i.e. the range of turbine counts is small), the algorithm is set up to expand this range, not to work within it [33]. Therefore, the algorithm does not play to its strengths and an approximation to the Pareto front of the problem will be poor [33].

What is encouraging is that the Pareto Front approximations in figures 1820 follow the general curve expected in the objective space for two competing optimisation functions. Figure 16 only includes the endpoints of such a curve. This is likely due to the limited number of generations and hence the limited number of points with which the Pareto Front can be estimated.

7 Future Considerations

The amount of learning required to write my project code limited the features that I could include. I have written a few sections below detailing the changes I would make to my project, if given the time.

7.1 Expanding the size of wind farm

I would have liked to adapt my existing code to solve the same optimisation problem using a 100×100 layout array. The obvious advantages to increasing the array size are two-fold. First, it increases the upper limit of turbines that can be placed within the array. This will lead to an increase in the upper limit of power the wind farm can produce.

Secondly, it allows for more separation between turbines. This means there will be less of an effect on any given turbine due to wake if the turbines are optimally placed.

7.2 Modelling for multi-directional wind and varying wind speeds.

In real-world conditions, although there will usually be a prevailing wind, wind comes from more than one direction throughout the day. This can be graphed using a wind rose.

I also applied the assumption in my wake model that the wind velocity is 12 m/s. The reason for this is that at speeds greater than or equal to 12.8 m/s, a turbine produces its maximum power output of 630 kWh [4]. This

simplified my wake model. It would have been advantageous to consider varying wind speeds as would be found in practical applications of wind farm modelling.

7.3 Better implementation of the Jensen wake model

Although my code does account for the wake effect from multiple turbines on any one given turbine, it is not a true implementation of the multiple wake model. My model calculated wake using the equations for the single wake model and the following logic:

1. First, calculate the number of turbines $\{t_1...t_n\} \setminus t_i$, where n is the total number of turbines in grid 'g', in which turbine t_i lies within the wake radius;

2. Second, calculate the wake effect upon turbines $t_1...t_n$. This influences the wake effect upon turbine t_i in an additive manner.

In contrast, a multiple wake model should use the Katic et.al. sum of squares method, which allows the velocity of a turbine t to be calculated from [4]:

$$v_i = v_0 \times (1 - \frac{1 - \sqrt{1 - C_t}}{1 + \frac{kx^2}{r}} \times \frac{A_0}{A_i}) \tag{11}$$

where v_i is the affected wind velocity exiting turbine t_i; v_0 is the initial velocity of air entering the rotor blades of turbine t_i, as unaffected by wake; A_0 is the area in which the rotor of turbine t_i and its wake area; and A_i is the area swept by turbine t_i. [4].

7.4 Comparing with commercial packages

Comparison with commercial packages such as PyWake would have been advantageous for my research. It would have allowed me to compare my wake model with PyWake's wake model by comparing the power output of wind farms using both models. PyWake has been used by many in their modelling of wind farms for the purpose of comparison [34][35].

7.5 Improvements for efficiency of code

There were improvements I could have made such that my project code was more efficient. I would have liked to streamline the calculations made in my wake model to require fewer iterations in its for and while loops. I could have achieved this by using a different method to account for the wake effect causing zero-output of turbines. Pre-calculating this possibility across the 10×10 array would have allowed the sampling method to avoid placing turbines in those areas, as opposed to my random sampling method.

References

[1] Chen W.; Alharthi M.; Zhang J; Khan I. "The need for energy efficiency and economic prosperity in a sustainable environment". In: *Gondwana Research* 127 (2024), pp. 22–35.

[2] Kassem Y.; Çamur H.; Aateg R. "Exploring Solar and Wind Energy as a Power Generation Source for Solving the Electricity Crisis in Libya". In: *Energies* 13.14 (2020), p. 3708.

[3] Awogbemi O.; Ojo A.O. “Harnessing Wind Energy to Solve Nigeria’s Energy Crisis”. In: *Journal of Engineering and Applied Sciences* 4.3 (2009), pp. 197–204.

[4] Manikowski P.L.; Walker D.J.; Craven M.J. “Multi-Objective Optimisation of the Benchmark Wind Farm Layout Problem”. In: *Journal of Marine Science and Engineering* 9.12 (2021), p. 1376. DOI: https://doi.org/10.3390/jmse9121376.

[5] Mosetti G.; Poloni C.; Diviacco D. “Optimization of wind turbine positioning in large wind farms by means of a Genetic algorithm”. In: *Journal of wind engineering and industrial aerodynamics* 51 (1994), pp. 105–116.

[6] Ragheb M. “Wind Energy Conversion Theory, Betz Equation”. 2022. URL: \url{http://ragheb.co/NPRE%20475%20Wind%20Power%20Systems/Wind%20Energy%20Conversion%20Theory%20Betz%20Equation..pdf}[Accessed11thMarch2025].

[7] Blackwood M. “Maximum Efficiency of a Wind Turbine”. In: *Undergraduate Journal of Mathematical Modeling: One + Two* 6 (2 2016).

[8] Strauss C.E.M. *Beating Betz’s Law: A Larger Fundamental Upper Bound For Wind Energy Harvesting*. Tech. rep. Browndogs Polytechnic Research Institute, 2021.

[9] Liew J.; Heck K.S.; Howland M.F. “Unified momentum model for rotor aerodynamics across operating regimes”. In: *Nature Communications* (2024).

[10] Howland M.F; Quesada J.B.; Martínez J.J.P.; Larrañaga F.P.; Chawla J.S.; Sivaram V.; Dabiri J.O. "Collective wind farm operation based on a predictive model increases utility-scale energy production". In: *Nature Energy* (2022).

[11] Archer C.L; Vasel-Be-Hagh A.; Yan C.; Wu S.; Pan Y.; Brodie J.F.; Macguire A.E. "Review and Evaluation of Wake Loss Models for Wind Energy Applications". In: *Applied Energy* 226 (2018), pp. 1187–1207.

[12] Al Shereiqi A.; Mohandes B.; Al-Hinai A.; Bakhtvar M.; Al-Abri R.; El Moursi M.; Albadi M. "Co-optimisation of wind farm micro-siting and cabling layouts". In: *IET Renewable Power Generation* 15 (2021), 1848–1860.

[13] Beal; Logan et al. "Gekko Optimisation Suite". In: *Processes* 6.8 (2018), p. 106. DOI: 10.3390/pr6080106.

[14] Freund R.M. "Issues in Non-Convex Optimization". With assistance from Brain W. Anthony. 2004. URL: \url{https://ocw.mit.edu/courses/15-094j-systems-optimization-models-and-computation-sma-5223-spring2004/abc71c7c3ec71d29f1943679b7370272_non_convex_prob.pdf}[Accessed10thMarch2025].

[15] Baptista J.; Jesus B.; Cerviera B.; Pires E.J.S. "Offshore Wind Farm Layout Optimisation Considering Wake Effects and Power Losses". In: *MDPI - Sustainability* 15 (2023), p. 9893. DOI: https://doi.org/10.3390/su15139893.

[16] Taylor P.; Yue H.; Campos-Gaona D.; Anaya-Lara O.; Jia C. "Wind farm array cable layout optimisation for complex offshore sites—A de-

composition based heuristic approach." In: *Iet Renew. Power Gener.* 17 (2023), pp. 243–259.

[17] Cerveira A.; de Sousa A.; Pires E.J.S.; Baptista J. "Optimizing wind farm cable layout considering ditch sharing." In: *International Transactions in Operational Research* 31(1) (2024), pp. 88–114.

[18] Balakrishnan R.K.; Hur S.H. "Maximization of the Power Production of an Offshore Wind Farm." In: *Appl. Sci.* 12 (2022), p. 4013.

[19] Baptista J.; Lima F.; Cerveira A. "Optimization of Wind Turbines Placement in Offshore Wind Farms: Wake Effects Concerns." In: *In International Conference on Optimization, Learning Algorithms and Applications, OL2A 2021* (2021), pp. 102–109.

[20] Jesus B.; Cerveira A.; Baptista J. "Optimization of Offshore Wind Farms Configuration Minimizing the Wake Effect." In: *In Proceedings of the 20th International Conference on Renewable Energies and Power Quality (ICREPQ'22), Vigo, Spain, 27–29 July 2022* (2022).

[21] Anagnostopoulos S.; Piggott M. "Offshore wind farm wake modelling using deep feed forward neural networks for active yaw control and layout optimisation." In: *In Journal of Physics: Conference Series; IOP Publishing: Bristol, UK, 2022* 2151 (2022), p. 012011.

[22] Cerveira A.; Pires E.J.S.; Baptista J. "Wind farm cable connection layout optimization with several substations." In: *Energies* 14 (2021), p. 3615.

[23] Fischetti M.; Pisinger D. “Mathematical optimization and algorithms for offshore wind farm design: An overview.” In: *Bus. Inf. Syst. Eng.* 61 (2019), pp. 469–485.

[24] Fischetti M.; Pisinger D. “Optimizing wind farm cable routing considering power losses.” In: *Eur. J. Oper. Res.* 270 (2018), pp. 917–930.

[25] Horn H.H. “Otimização do Parque Eólico de Osório Utilizando Algorítmo Genético.” PhD thesis. University of Vale do Rio dos Sinos, São Leopoldo, Brazil, 2018.

[26] Amaral L.; Castro R. “Offshore wind farm layout optimization regarding wake effects and electrical losses.” In: *Eng. Appl. Artif. Intell.* 60 (2017), pp. 26–34.

[27] Cerveira A.; de Sousa A.; Pires E.S.; Baptista J. “Optimal cable design of wind farms: The infrastructure and losses cost minimization case.” In: *IEEE Trans. Power Syst.* 31 (2016), pp. 4319–4329.

[28] Holland J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1975.

[29] Blank J. From the Pymoo documentation page on NSGA-2. URL: `\url{https://pymoo.org/algorithms/moo/nsga2.html}[Accessed15thMarch2025]`.

[30] Emami A.; Noghreh P. “New approach on optimization in placement of wind turbines within wind farm by genetic algorithms.” In: *Renewable Energy* 35 (2010), pp. 1559–1564.

[31] Ah King R.T.F.; Deb K.; Rughooputh H.C.S. “Comparison of NSGA-II and SPEA2 on the Multiobjective Environmental/Economic Dispatch Problem”. In: *University of Mauritius Research Journal* 16 (2010).

[32] Zitzler E.; Laumanns M.; Thiele L. *SPEA2: Improving the strength pareto evolutionary algorithm.* Tech. rep. Swiss Federal Institute of Technology (ETH) Zurich, 2001.

[33] Liu X.; Zhang D. “An Improved SPEA2 Algorithm with Local Search for Multi-Objective Investment Decision-Making”. In: *Journal of Applied Sciences* (2019).

[34] al Halabi M. “Comparing wind farm production data to engineering wake model simulations”. MA thesis. University of South-Eastern Norway, 2023.

[35] Sørensen S.J. “Offshore Wind Farms Modelled for Simulations with PyWake”. MA thesis. University of South-Eastern Norway, 2024.

Appendices

A Link to project code

Runtime can be over an hour depending on population size and number of generations. To test that the core functionality works and graphs are properly generated, use a minimum population size of 200 using 20 generations. Allow up to 40 minutes for the execution of each algorithm.

https://colab.research.google.com/drive/1F8aaAVJMW7HHQcgp31G7UWE8E2HDcdwS?usp=sharing