Carl von Ossietzky Universität Oldenburg

Fraunhofer Institute for Wind Energy Systems

Wind Farm Optimization: Multi-objective problems

Bachelor Internship Report

01.03.2020 - 30.6.2020

Author: Alberto Jimenez Haro

Email: `alberto.jimenez.haro@uni-oldenburg.de`

Matrikel-Nr.: 3323417

Supervisor: Dr. rer. nat. Jonas Schmidt

University Supervisor: Dipl.-Ing. Andreas Hermann Schmidt

Abstract

The following paper describes the achievements and experience of my bachelor internship at the research institute Fraunhofer IWES in Oldenburg. During the internship I tried to implement minimum cable length calculations into the IWES wind farm modeling python code *Flappy*[1] and to calculate and analyze the Pareto front of the multi-objective wind farm layout optimization problem. To this aim firstly a program that searches to minimize the cable length between turbines was created using a minimum spanning tree. This program was later implemented inside of *Flappy* and used the calculations of the minimum spanning tree as the objective function for optimizing wind farm layouts of the turbines. A period of testing different optimization algorithms to understand better their performance and behaviour. Finally by using the above objective function plus a second objective function which calculates the maximum yield, and with the use of the multi-objective algorithm 'NSGA-II' I was able to create a multi-objective wind farm layout optimization problem, and calculate the Pareto front of this one plus an analysis of different Pareto fronts calculated under different conditions.

Contents

1 The Institute: Fraunhofer Institute for Wind Energy Systems IWES

Founded in 1949, the research organization of Fraunhofer undertakes applied research. Having international collaborations with excellent research partners and innovative companies around the world. At present, the Fraunhofer-Gesellschaft maintains 72 institutes and research units, with a staff of 25,000 qualified scientists and engineers.

Unlike research institutions that concentrate more heavily on basic research, direct assignments from industry are central for the organization's funding. Around 70 percent of the Fraunhofer-Gesellschaft's contract research revenue is derived from contracts with industry and from publicly financed research projects. The other 30 percent is contributed by the German federal and state governments in the form of base funding.

The Fraunhofer Institute for Wind Energy Systems IWES was founded in 2009 and emerged from the former Fraunhofer Center for Wind Energy and Marine Technology CWMT in Bremerhaven. It secures investments in further technological developments through validation, shortens innovation cycles, accelerates certification processes and increases planning accuracy through innovative measuring methods in the field of wind energy[2].

IWES collaborates with other institutes and industrial partners. The Research Alliance for Wind Energy (FVWE) consists of ForWind, DLR and IWES and forms a cooperation network of 600 skilled scientist. Having also close collaboration with the regional universities in Hanover, Oldenburg and Bremen and with the University of Applied Sciences in Bremerhaven. The close alliances with industrial partners are achieved through a *board of trustees* (representatives of rotor blade manufacturers, suppliers and politicians) which advises on its strategic orientation and its positioning in the growth markets for wind energy. And a *steering committee* for rotor blade testing ensures that test programmes and methods are tailored to the requirements of industry. The current manager director of Fraunhofer IWES is Prof. Dr.-Ing. Andreas Reuter, with a total of 220 scientific employees[3].

Here in the station of IWES in Oldenburg, work is done under the direction of Dr. Bernhard Stoevesandt who is head of the department of Aerodynamics, CFD and stochastic dynamics, and Prof. Joachim Peinke who is the site manager in Oldenburg. It works together with other groups in Oldenburg forming the TWiST group - turbulence, wind energy and stochastics. It is located inside the university campus at Küpkersweg 70, inside the WindLab - W33 building.

2 The proyect: Developing Multi-objective problems for Wind Farm Optimization

2.1 Introduction and aim of the project

Wind energy as a source for electricity generation has played a dominant role in the European Union and elsewhere, fueled by technology cost reductions and public promotion schemes. In turn, wind deployment has led to considerable socioeconomic and environmental benefits[4]. Being an increasingly competitive market, it is very important to minimize establishing costs and to increase production profits already in the design phase of new wind farms[5].

There are many many factors that influence the profits that can be obtained from wind energy, like the location of wind farms, wake interferences, structural materials and manteinance... Multi-objective optimization problems allow us to perform optimized trade-offs between energy production, capital investment and operational costs, which can significantly reduce costs. The aim of this project is to create and solve a multi-objective wind farm layout optimization problem. This means that firstly an objective function that searches to minimize the cable length between turbines has to be created, by use of a minimum spanning tree. These cable length calculation must be later implemented into the IWES wind farm modeling python code *Flappy*[1]. And finally the Pareto front of the multi-objective wind farm layout optimization problem must be calculated and analyzed with respect to this minimal cable length and the maximal yield (whose objective function was already implemented in *Flappy*).

2.2 Developing minimum cable length calculations

In order to achieve the final goal of creating and solving a multi-objective wind farm layout optimization problem, it is firstly needed to obtain our objective functions to be optimized. As mentioned before, the objective function which calculates the maximal yield has been already implemented in *Flappy*, therefore only the objective function which calculates the minimal cable length between the total number of turbines in the wind farm must be created.

2.2.1 Introduction to Graph Theory and Minimum Spanning Trees (MST)

Before being able to solve the minimal cable length problem of a set of turbines, a basic concept of Graph Theory and MSTs(Minimum Spanning Trees) must be understood in order to define the problem and be able to optimize it.

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph is defined to be a set of points (vertices) in space which are interconnected by a set of lines (edges) as shown in Figure 1a. For a graph G the vertex-set is denoted by V and the edge-set by E and write $G = (V, E)$[6]. E is a subset of the set $V^{(2)}$ of unordered pairs of V.[7]

There are two standard ways to represent a graph G. Firstly, as a collection of adjacency lists, where the graph G consists of an array *Adj* of $|V|$ lists, one for each vertex in V. For each $u \in V$, the adjacency list $Adj[u]$ contains (pointers to) all the vertices v such that there is an edge $(u, v) \in E$. That is, $Adj[u]$ consists of all the vertices adjacent to u in G. Figure 1b is the adjacency list of the undirected graph in Figure 1a.

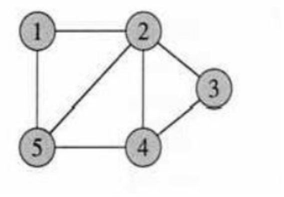
The second way is the adjacency-matrix representation of a graph, where the vertices are numbered $1, 2, ..., |V|$ in some arbitrary manner. The adjacency-matrix representation of a graph G then consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that,

$$a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

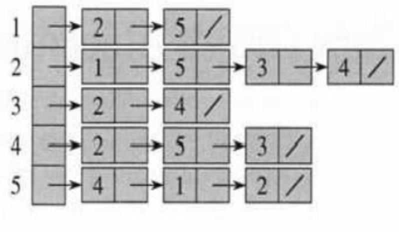
Figure 1c is the adjacency matrix of the undirected graph in Figure 1a. The adjacency-matrix representation will be chosen for our minimum cable length calculations.

The adjacency-list representation provides a compact way to represent sparse graphs for which $|E|$ is much less than $|V|^2$. An adjacency-matrix representation may be preferred, however, when the graph is dense for which $|E|$ is close to $|V|^2$ or when it is necessary to be able to tell quickly if there is an edge connecting two given vertices.

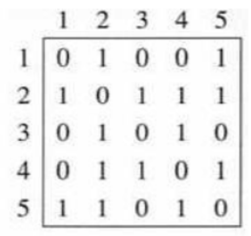
Graphs can either be undirected or directed. In a directed graph, each edge has a direction, and nodes on edges form ordered pairs. Edges can also be weighted. In weighted graphs, each edge has an associated weight, typically given by a weight function $w : E \to \mathbb{R}$, where each weight $w(u,v)$ is associated with an edge $(u,v) \in E$. A weighted undirected graph will be used for our calculations, with further explanation in section 2.2.3.



(a)



(b)



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Figure 1: Two representations of an undirected graph. (a) An undirected graph G having five vertices and seven edges. (b) An adjacency-list representation of G. (c) The adjacency matrix representation of G.[8]

So, for the case of undirected weighted graph, we have that for each edge $(u,v) \in E$, we have a weight $w(u,v)$ specifying the "cost" to connect u and v. We then wish to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

is minimized. Since T is acyclic and connects all of the vertices, it must form a tree (a connected, acyclic graph, i.e, a connected graph with no cycles), which we call a spanning tree since it "spans" the graph G. We call the problem of determining the tree T the "minimum-spanning-tree" problem (shortened for "minimum-weight spanning tree."). Figure 2 shows an example of a connected graph and its minimum spanning tree. All spanning trees have exactly $|V| - 1$ edges.

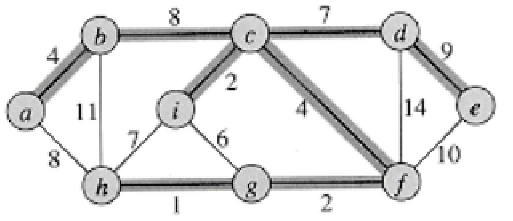


Figure 2: A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. The tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.[8]

Minimum-spanning-tree algorithms are a classic example of the greedy method. Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. A greedy algorithm always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. Greedy algorithms do not always yield optimal solutions, but for many problems they do, in which case it generally becomes the best method to solve that problem as they are usually more efficient than other techniques like Dynamic Programming or Brute Force algorithms.

There are a number of algorithms known to solve the MST problem for undirected graphs. The best known are the greedy algorithms of Prim and Kruskal.[6]. Kruskals' algorithm[9] treats the graph as a forest (a graph whose connected components are trees) and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

Prim's algorithm[10], in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph. This algorithm will be chosen for the calculations of our cable connections, and is therefore vital to understand it's procedure.

Firstly, lets imagine a set A that is always a subset of some minimum spanning tree. At each step, an edge (u, v) is determined that can be added to A without violating this invariant, in the sense that $A \cup \{(u, v)\}$ is also a subset of a minimum spanning tree. We call such an edge a *safe edge* for A, since it can be safely added to A without destroying the invariant. Also let C be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Prim's algorithm has the property that the edges in the set A always form a single tree. As is illustrated in Figure 3 the tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V. At each step, a light edge connecting a vertex in A to a vertex in $V - A$ is added to the tree. When the algorithm terminates, the edges in A form a minimum spanning tree. This strategy is "greedy" since the tree is augmented at each step with an edge that contributes the minimum amount possible to the tree's weight.

```
MST-PRIM(G, w, r)

$Q \leftarrow V[G]$
for each $u \in Q$
    do key[$u$] $\leftarrow \infty$
key [$r$] $\leftarrow 0$
$\Pi[r] \leftarrow$ NIL
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) <$ key[$v$]
                then $\Pi[v] \leftarrow u$
                    key[$v$] $\leftarrow w(u, v)$
```

In the pseudocode above, the connected graph G and the root r of the minimum spanning tree to be grown are inputs to the algorithm. During execution of the algorithm, all vertices that are not in the tree reside in a priority queue Q based on a key field. For each vertex v, key$[v]$ is the minimum weight of any edge connecting v to a vertex in the tree; by convention, key$[v] = \infty$ if there is no such edge. The field $\Pi[v]$ names the "parent" of v in the tree. During the algorithm, the set A is kept implicitly as $A = \{(v, \Pi[v]) : v \in V - \{r\} - Q\}$. When the algorithm terminates, the priority queue Q is empty; the minimum spanning tree A for G is thus $A = \{(v, \Pi[v]) : v \in V - \{r\}\}$.

So the process that can be visually seen in Figure 3, follows these steps shown in the pseudo-code:

- Lines **1-4** initialize the priority queue Q to contain all the vertices and set the key of each vertex to ∞, (except for the root r, whose key is set to 0)
- Line **5** initializes $\Pi[r]$ to NIL(empty set/list containing no entries), since the root r has no parent.
- Line **7** identifies a vertex u $\in$ Q incident on a light edge crossing the cut $(V - Q, Q)$ (except in the first iteration). Removing u from the set Q adds it to the set $V - Q$(contains the vertices in the tree being grown) of vertices in the tree.
- Lines **8-11** update the key and Π fields of every vertex v adjacent to u but not in the tree.

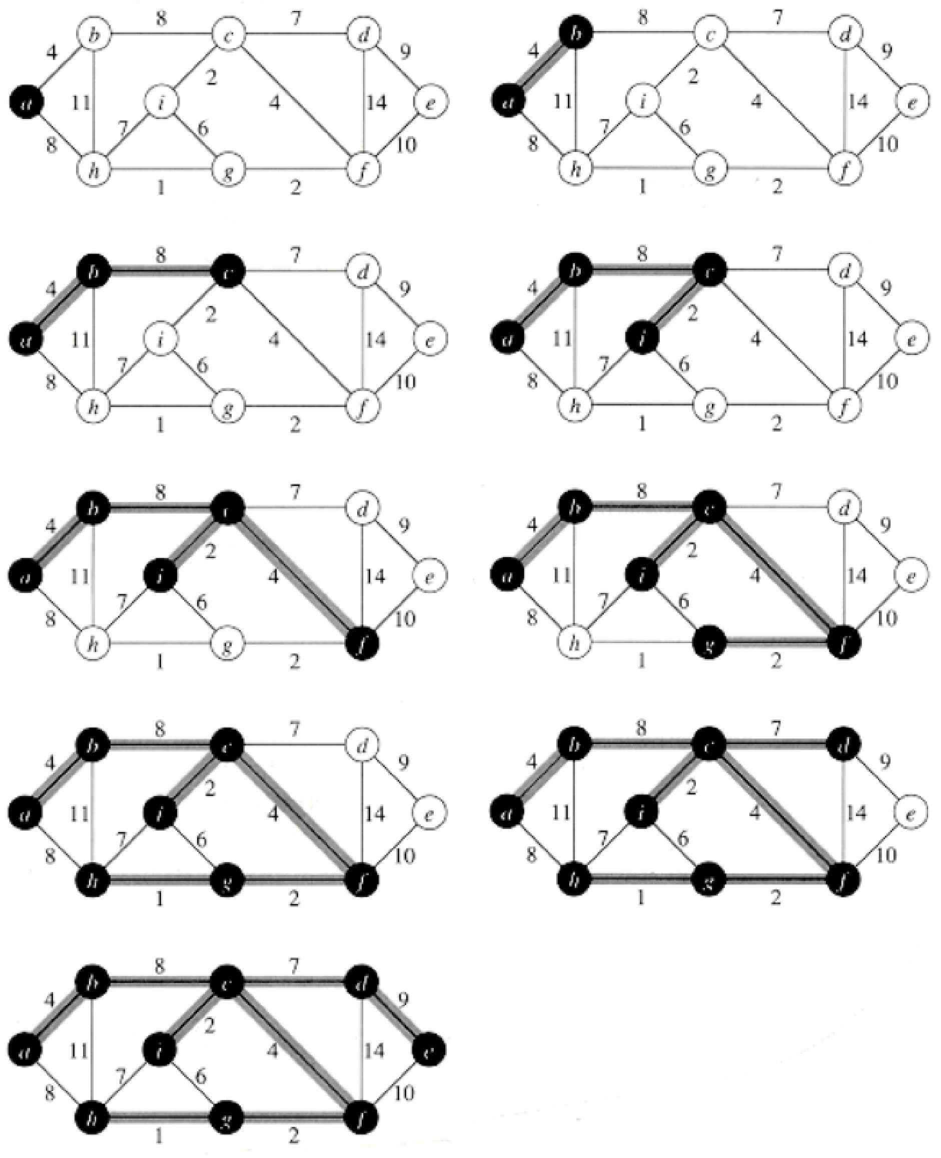


Figure 3: The execution of Prim's algorithm on the graph from Figure 2. The root vertex is at the top left. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut. [8]

2.2.2 Wind Farm Cable Connections

A typical wind farm contains a group of wind turbines, substations and grid points which are connected by a cabling system (see Figure 4). The turbines are driven by wind wheels and convert wind's kinetic energy into electrical power. This electrical energy is transferred by the cabling system to a substation collecting the electrical power of a group of turbines. The substation transforms the electrical energy to alternating current with a high voltage level to minimize transport energy loss. All substations are connected to a grid point, which is the entry of the general power grid.[11]

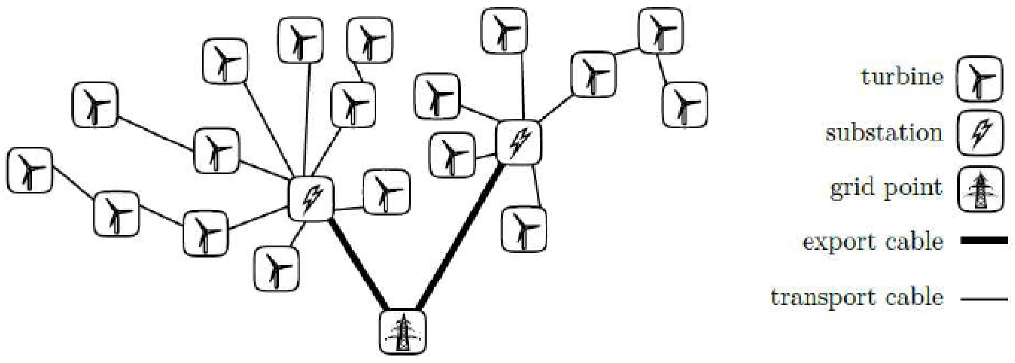


Figure 4: Cable connection structure of a wind farm.[11]

There are multiple types of transport cables, with different costs and capacities. There is also a limit on how many turbines can be connected to the same substation.[12] The challenge of the designers when dealing with the wind farm cabling problem, is to design the internal cabling of a wind farm such that all power from the turbines can be transmitted to the substations and the costs for the cabling are minimized.

The cost for the internal cabling accounts for approximately 17% of the total cost for planning and building a wind farm.[13] Therefore, it is essential to find a cost-efficient cabling. The cost of a cable will depend on several factors, being the amount of energy flow through it, the work needed for constructing the road and the burying of the cable as well as the terrain in which it is buried. Cables can also be chosen from several available cable types, each of which has a thermal capacity and cost. [14]

Wind turbines need to be connected to a transformer station and the existing external grid, and for this interconnecting cables buried underground are generally used. For connecting the transformer station to the external grid, an export cable with high capacity and cost is used. One must also consider the cost of power losses which is proportional to the length of a cable and the square of the current. Interconnecting cables of different dimensions can allow for different current levels and thus different number of connected wind turbines.[15]

Taking into account all of these factors, solving for an optimum layout becomes increasingly hard (infeasible in many cases). A simplified model of cable connections is therefore used for our calculations. There will be no grid points or export cables taken into account, not even substations per se. Just one type of cable will be used, without consideration for power losses. The terrains orography and any type of obstacles including roads will also not be taken into account.

2.2.3 Creating a program that calculates minimum cable length

Having now defined our wind farm cable characteristics in section 2.2.2, and the algorithm techniques in section 2.2.1(go back to this section if any of the vocabulary seems unfamiliar), we can finally create a program that calculates the minimum cable length of a set of fixed turbines.

To begin our problem must be correctly defined. We have a set of vertices which represent our turbines in a wind farm. The turbines are connected through edges, which are straight lines between vertices and represent the cable connections between the turbines (which are theoretically perfectly straight). The "weight" of the graph is the cable length in meters. Because we don't care about the direction flow of electricity in the cables we have an undirected graph problem. As we finally have an undirected weighted graph problem, we will use a minimum spanning tree to solve it, in this case Prim's algortithm.

To write the program, the programming language *Python* has been used. With the help of some standard *Python* code for calculating Prim's algortihtm[16] which was altered for our specific problem, the calculation of the shortest cable connection was possible. The code used follows the procedure of Prim's algorithm that was explained in the Pseudo-code from section 2.2.1 which can be graphically visualized in Figure 3. Being the program an adjacency matrix representation of the graph, the time complexity of this one is $O(V^2)$. After the calculation is done, a figure is created with all the information and printed into the screen, which can be seen in Figure 5.

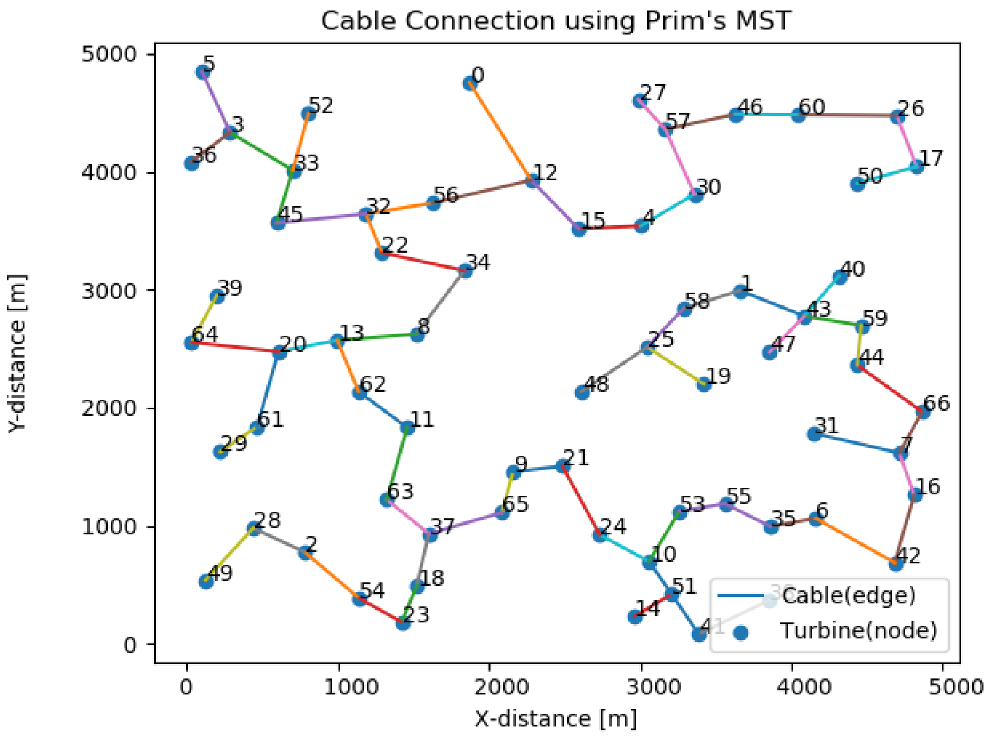


Figure 5: Calculation of the shortest cable connection between the set of turbines from a wind farm, by using a version of Prim's MST in *Python* and printing the final results into a figure.

2.3 Implementation of cable length calculations into software ***Flappy*** and testing of optimization algorithms

We have now developed a program that calculates the shortest cable connection of a set of fixed turbines by the use of a minimum spanning tree algorithm. Now this minimal cable length calculated by our program can be used as an objective function in a wind farm position layout optimization problem. In this case, an optimization algorithm will be used to re-position the turbines in the wind farm layout in a way it minimizes (or hopes to minimize) the objective function which in this case is the shortest cable connection achieved with Prim's MST algorithm.

2.3.1 Optimization Problems and Algorithms

Finding an optimal optimization algorithm for our layout and understanding the different methods and characteristics of these is core to solving the wind farm cabling problem. An optimization problem is the problem of finding the best solution from all feasible solutions.

In general, an optimization problem can be written as minimize

$$f_1(\mathbf{x}), ..., f_i(\mathbf{x}), ..., f_I(\mathbf{x}), \quad \mathbf{x} = (x_1, ..., x_d), \tag{1}$$

subject to

$$h_j(\mathbf{x}) = 0, (j = 1, 2, ..., J)$$
$$g_k(\mathbf{x}) \leq 0, (k = 1, 2, ..., K),$$

where $f_1, ..., f_I$ are the objectives, while h_j and g_k are the equality and inequality constraints, respectively.

when $I = 1$, it is called single-objective optimization. When $I \geq 2$, it becomes a multiobjective problem whose solution strategy is different from those for a single objective (we will deal with this later on).

To solve the optimization problem, efficient search or optimization algorithms are needed. There are many optimization algorithms which can be classified in different ways. They can be classified as gradient-based algorithms (use derivative information) or derivative-free or gradient-free algorithms (use instead the values of the function itself). Also as trajectory-based which uses a single agent or one solution at a time, or population-based algorithms which use multiple agents which will interact and trace out multiple paths. Stochastic or deterministic algorithms depending on their randomness in nature or lack of this one. And finally local or global search algorithms depending on if it converges towards a local optimum.

Algorithms with stochastic components were often referred to as heuristic algorithms, which are usually problem-dependent algorithms designed to solve a problem in a faster and more efficient fashion than traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed. Beyond these algorithms we encounter a crucial type of algorithms which we will use in our optimization strategies, and these are metaheuristic algorithms.

A metaheuristic is a high-level problem-independent algorithmic framework that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. All metaheuristic algorithms use a certain trade-off of randomization and local search. Quality solutions to difficult optimization problems can be found in a reasonable amount of time, but there is no guarantee that optimal solutions can be reached.

The two major components of a metaheuristic algorithm are 'Diversification' (to generate diverse solutions so as to explore the search space on a global scale allowing the search to escape from local optima) and 'Intensification' (to focus the search in a local region knowing that a current good solution is found in this region). A good combination of these will usually ensure that global optimality is achievable, which will later be our goal when trying to find the best metaheuristic algorithm and its parameters.[17][18]

2.3.2 Implementing minimum cable length calculations into software ***Flappy***

With an understanding of the optimization strategies that are available, the next step is to implement as an objective function the minimal cable length calculation and use it for our Wind Farm Layout Optimization problem. This one will be created inside of the software *Flappy*[1].

Flappy is a In-house wind farm and wake modelling tool developed here at Fraunhofer IWES. It stands for "*Farm Layout Program in Python*".

Flappy is used for fast calculation of wind farm yield results, and other wind farm related quantities. It can also make fast wind farm flow calculations (not CFD calculations), and can handle large time series and statistical wind input data. It has a broad modeling platform for wind farms and wake simulations and can be coupled to optimization libraries, for use, for example, in wind farm layout optimization in single and multi-objective problems. This last property will be necessary for our optimization calculations and the optimization library used called *Pygmo* will be further explained later in this section.[19]

This in-house software will create and fill a wind farm, by adding turbines using selected models. It will first fill a model book with all these selected model choices that are necessary to run the program. These models are:

– *Rotor models* which calculate effective data as seen by the wind turbine rotors: Effective wind vector, wind speed, turbulence intensity, air density.

– *Turbine models* calculate variables from current flappy variables and flow field.

– *Controllers* are responsible for running the turbine models.

– *Wake models* calculate wake deltas of wind vectors and turbulence intensity scalars, at any set of points.

– *Wake frames* calculate the coordinates (x, y, z) for input points, at which the wake models should be evaluated.

– *Wake superposition* combines wake effects.

Once our model book is filled and all the specific characteristics of the turbines have been fixed (number of turbines, hub height, rotor diameter...) and a wind farm boundary geometry is added (which defines the constraining area), the model book will be used to run the wind farm setup.

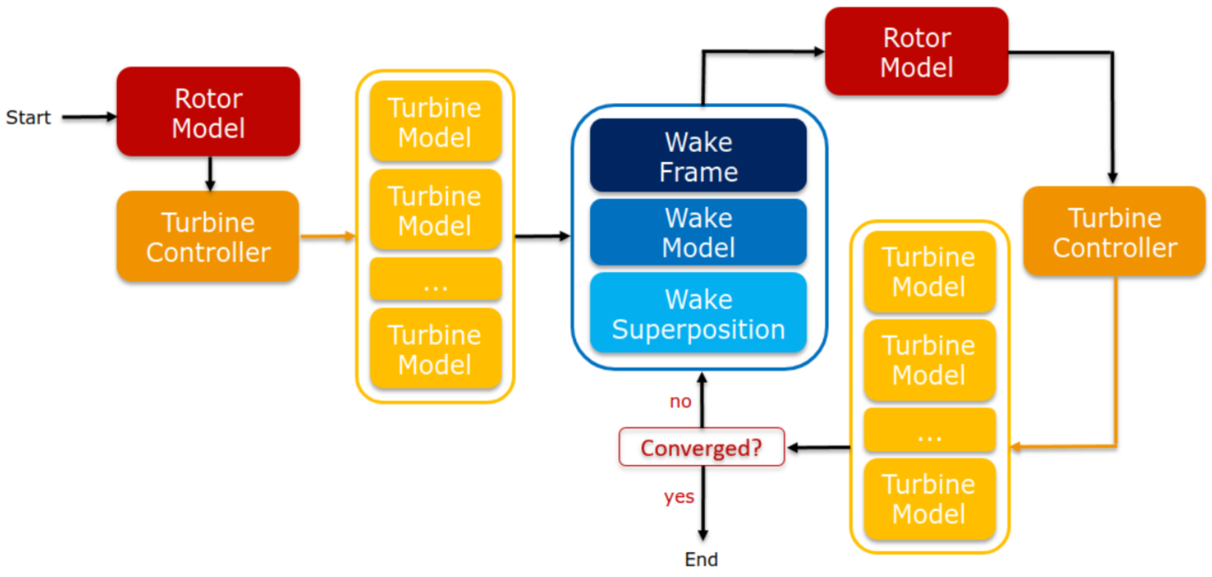


Figure 6: The working logic of *Flappy*[20]

Next step the ambient flow states will be created, each state is a single situation for which the wind farm is evaluated (a set of fixed values for wind speed, wind direction, turbulence intensity and air density for the case of a uniform inflow). In practice, flappy can evaluates many such states using time series and wind rose data which increases the computational cost of the calculations. After creating the wind farm and the ambient flow states, we are now in the position to run calculations of wind farm quantities for all states. The working logic can be seen in Figure 6.

This is the basic framework for any *Flappy* calculation. In our case when presented with our implementation of cable length calculations, we must also add a cable model. The ambient flow states calculations are not really relevant for our optimization of shortest cable connections, as it will depend more in the spatial positioning and constraints of the wind farm. Nevertheless, when optimizing maximum yield it then becomes crucial, therefore it is necessary for our future multi-objective problem.

From this point on the optimization elements come into action. *Flappy* will create and initialize the optimization problem, using the wind farm and the states. After that it will add and initialize the objective function(s) $f_1(\mathbf{x}), ..., f_I(\mathbf{x})$ from Equation 1. In this case our single objective function will minimize the total cable length between turbines.

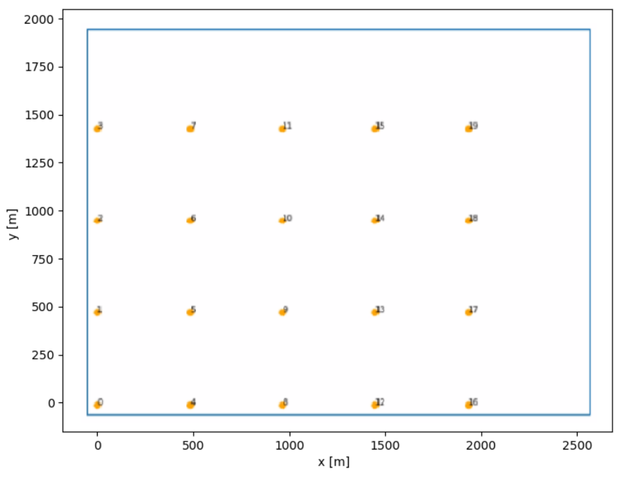
Next move it will also add and initialize the optimization constraints $g_k(\mathbf{x})$ also from Equation 1. We have two inequality constraints in our problem. Firstly the land delimitation where the wind turbines can reside inside of the wind farm. Turbines cannot reside outside of the encircling area. We refer to this one as 'area constraint' and it can be seen as the blue perimeter line surrounding the turbines in Figure 7.

The second inequality constraint is the so called 'distance constraint'. Wind passing through a wind turbine will decrease in speed, not to recover fully for quite some distance. Thus, two wind turbines standing not too far apart will have a negative effect on each others production, referred to as wake effects or array losses. This loss of wind speed can be calculated with our *Flappy* models and the optimal positioning of wind turbines will depend on this effect. This is the principle used in the already implemented objective function in *Flappy* which calculates the maximum yield by taking into account this wake effect between turbines. In our cable optimization program, because of this wake effect we reserve a minimum distance constraint between turbines, being this our second inequality constraint.

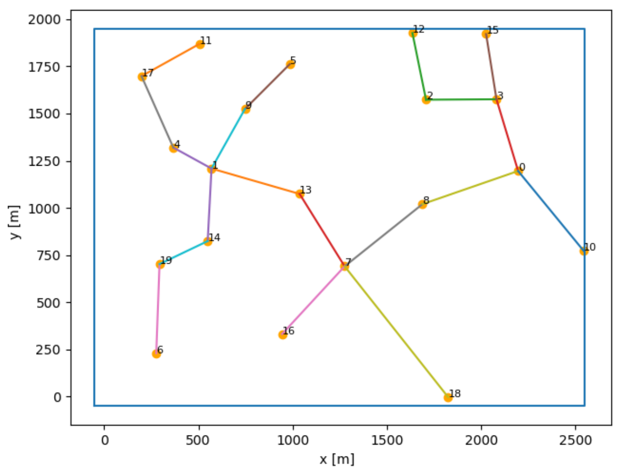
With our objective function and constraints defined, we can now create and initialize the optimization solver. By using the software *iwes-opt* from Fraunhofer IWES which is an interface to external optimization libraries, the scientific library for parallel optimization in Python and C++ *Pygmo* can be used[21]. Developed within the Advanced Concepts Team of the European Space Agency, *Pygmo* provides a large number of optimization problems and algorithms. These last can be used to solve our *Flappy* problems.

For our cable problem, we implemented two different optimizers, firstly a local optimizer called 'Ipopt' (Interior Point Optimizer) which is designed to find (local) solutions of mathematical optimization problems. And a metaheuristic optimizer (see section 2.3.1) called 'SGA' (Simple Genetic Algorithm), a biologically-inspired computer science technique that combine notions from Mendelian genetics and Darwinian evolution to search for good solutions to problems. The SGA works by generating a random population of solutions to a problem, evaluating those solutions and then using cloning, recombination and mutation to create new solutions to the problem.[22]

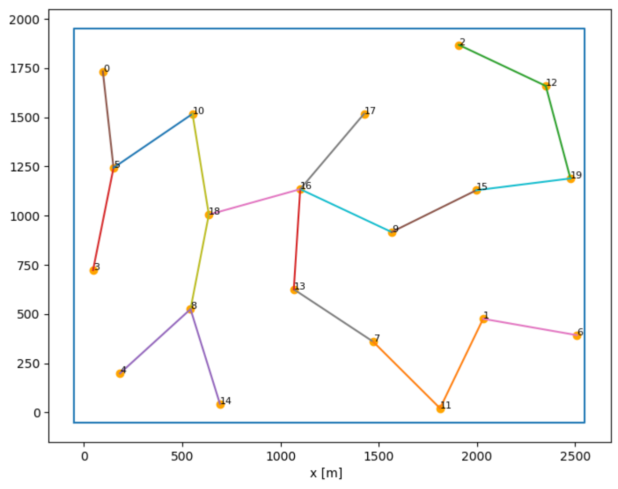
Using these optimization algorithms, we can run the optimization solver which will constantly re-position the turbines in the wind farm layout in a way it minimizes (or hopes to minimize) the objective function which in this case is the shortest cable connection achieved with Prim's MST algorithm.



(a)



(b)



(c)

Figure 7: Optimization solver using the SGA metaheuristic optimizer. At (a) the optimization process has not yet started (initial positions of turbines). (b) shows the optimized turbine positions and shortest cable connections at generation 10, and (c) at generation 200.

The results as this process is occurring can be seen in Figure 7. Once the optimization is finished, the evaluation of the optimization results is made. In our case, we obtain the figure of the wind farm layout with the minimum cable connection between them and a data file with the final cable length together with the optimized turbine positions and indication of any constraint failures and where.

2.3.3 Testing Optimization Algorithms for *Flappy*

Having achieved to create an optimizer for shortest cable connections, we must now ensure that the optimizer is efficient, effective and that there are no problems (or as little as possible) with the constraints we imposed. The importance of finding the correct optimization algorithm for a certain problem was already mentioned in section 2.3.1. The scientific library for parallel optimization *Pygmo* which has been already explained in section 2.3.2 provides a large number of optimization algorithms that can be used. We will explore the performance of some of these algorithms in hope to better assess future optimization problems in *Flappy*.

Five algorithms will be compared, two of them ('Ipopt' and 'SGA') have been already explained in section 2.3.2. The local optimizers are 'Ipopt' and 'SLSQP' which is a gradient-based and derivative-free optimizer which wraps a selection of solvers from the *NLopt*(nonlinear optimization) library. The other three are metaheuristic algorithms being 'SGA', 'PSO' (Particle Swarm Optimization) inspired from swarm behavior such as bird flocking and schooling in nature, where the new location of each particle is determined by a "velocity term" reflecting the attraction of global best and its own best during the history of the particle and random coefficients[23], and 'ABC'(Artificial Bee Colony) based on the behavior of honey bee, where a colony of artificial forager bees (agents) search for rich artificial food sources (good solutions for a given problem)[24].

For the comparison, a program was created in *Flappy* where all these algorithms or a selected number of them would be executed in series for a user determined number of cycles, during which various of their parameters would change, obtaining finally a data file with all information changes together with visual bar chart comparisons (see Figure 8) and all the layouts.

Over 300 different performance executions of different optimization problems were compared with different parameters or conditions. They will be first examined in different categories as per their local optimizer or metaheuristic nature described above.

The metaheuristic algorithms shared all many of the changes they experienced, being therefore very congruent with each other. Their performances could be described in the following way:

- Firstly *Repetition Tests* were performed to check if running repeatedly the exact same optimization problem scenarios resulted in similar outcomes. These tests were all successfully passed.
- The bigger the *population* of individuals the better the results are obtained for all 3 metaheuristics. Noticeable improvements can be observed until arriving to populations of over 100 individuals, where the improvement does not appear to be so clear anymore.
- The more *generations* of calculations are performed, evidently the better results are also obtained. In this case, there doesn't seem to be a clear upper limit for the number of generations (not forgetting that bigger numbers are more computationally expensive), although the rate of improvement of the fitness values (our minimal cable length or maximal yield) tends to slow down especially after 150 generations.
- When testing in *denser layouts* where the constraint area for positioning the turbines is smaller, the results are proportionally worsen in a consistent manner when compared to sparser layouts. The number of minimum distance constraint violations is also tripled or quadrupled compared with sparse layouts.
- In the case of more *complex area constraint shapes* (non-rectangular, more real like farm scenarios), the fitness values don't seem to be affected in a quantitative manner. Unfortunately some area constraint violations are sometimes found when the number of generations is lower than 150-200. 'SGA' seem to slightly outperformed the other two cases.
- The *number of turbines* also affects the results, with lower fitness values and more minimum distance constraint violations when the number of these is increased, which would be the expected results when added complexity.
- Finally the *constraint failures* found were very satisfactory in the case of area constraints, with almost no failures in any case except for low

generation complex farm field shapes. But in the case of minimum distance constraint violations there was for most cases a number between 1 - 7 constraint failures.

Another program was created with the aim of reducing the number of minimum distance constraint failures to a minimum. This program repeatedly modifies parameters that affect this constraint and then the results can be evaluated. There was no ideal parameter found giving zero constraint failures consistently, just a range of elite parameters.

Although the 3 algorithms used generated similar results for all cases, 'PSO' always performed slightly under 'SGA' and 'ABC'. It can be concluded that when using a high number of turbines, population and generations the best stable and consistent results of all are obtained, with very clear stepping and proportionality, as shown in Figure 8

The local optimizers 'Ipopt' and 'SLSQP' on the other hand performed in a different way to the metaheuristics. These algorithms always have a population of just one individual, and don't have many other parameters to be tuned apart from the finite difference step sizes which are not really improvable but they rather work or give really bad results.

They give in many cases good results compared with metaheuristics of lower populations and/or generations, and they almost never have any type of constraint failure of either area or minimum distance. Under the same conditions the perform exactly the same, because of their more deterministic nature. But they are unpredictable in some cases, giving a final farm layout in some cases similar to the user defined initial layout (see Figure 7a). Their results cannot really be improved as in the case of metaheuristics by increasing the number of population or generations, therefore at high number of these the metaheuristics take the lead. *Pygmo's* strength comes from their metaheuristic algorithms and the possibility of exploiting their parallel population dynamics where parallelization actually modifies the results of each population in each generation or every 'x' generations.

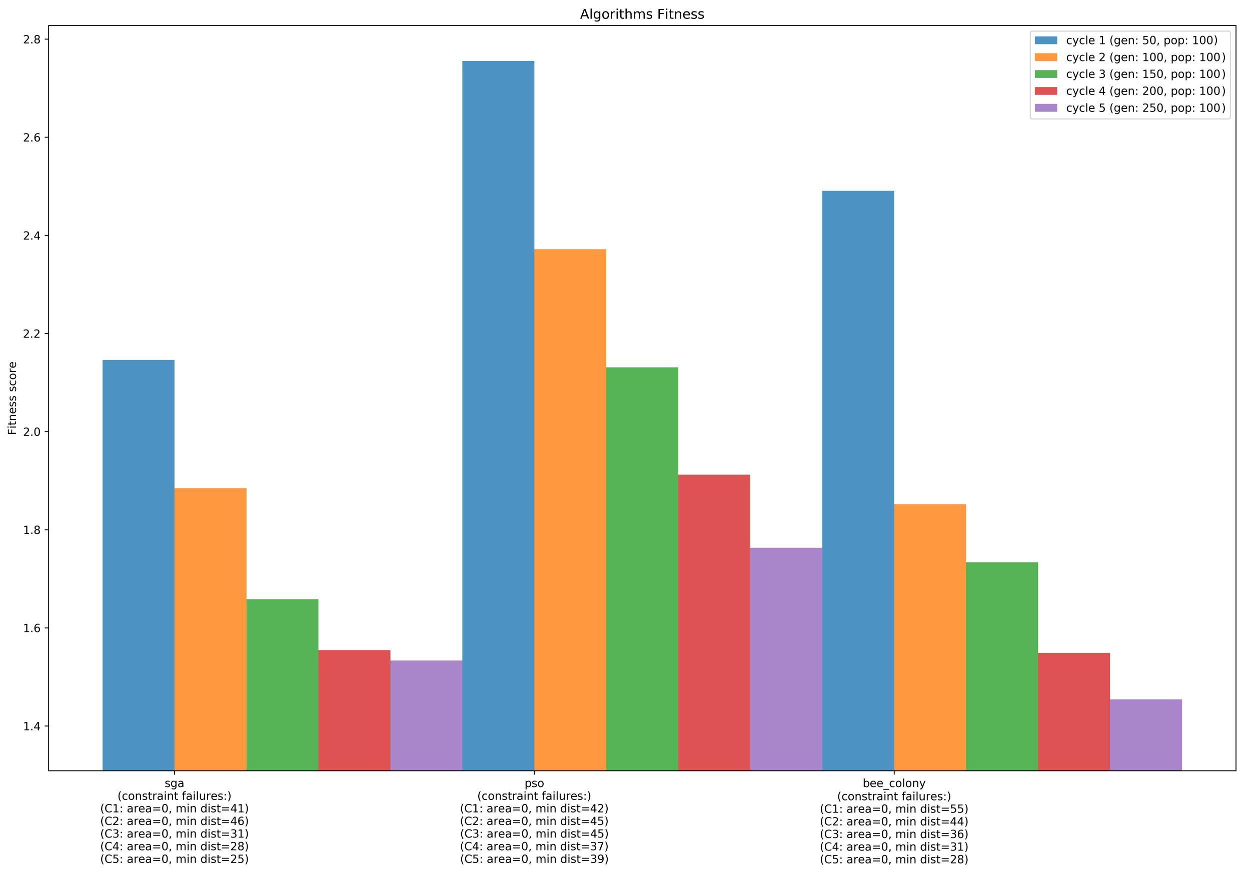


Figure 8: Bar chart comparison of the three metaheuristic algorithms 'SGA', 'PSO' and 'ABC', executed each 5 times(cycles) with 72 turbines, 100 population and 50 initial generations with an increase of 50 generations per cycle.

2.4 Creation of Multi-objective wind farm layout optimizaton problem

We have finally successfully completed all our chores, and gained the necessary knowledge to begin the creation of a multi-objective wind farm layout optimization problem for *Flappy*.

To create a multi-objective problem we just need to go back to the definition of an optimization problem given in Equation 1 in section 2.3.1 and use a value for $I \geq 2$. Optimization problems that have multiple (conflicting) objectives, essentially render the concept of optimality meaningless since the best solution for one objective may not be the best for another. In multi-objective optimization the concept of dominance is therefore introduced. A solution is said to dominate another solution if its quality is at least as good on every objective and better on at least one. To put this mathematically [25], if a vector $\mathbf{u} = (u_1, \ldots, u_k)$ is said to dominate $\mathbf{v} = (v_1, \ldots, v_k)$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) then,

$$\forall i \in \{1, \ldots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i.$$

The set of all non-dominated solutions of an optimization problem is called the Pareto set and the projection of this set onto the objective function space is called the Pareto front. This means that for an objective function $f(\mathbf{x})$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables and Ω the feasible region (set of all the values satisfying the constraints). The pareto optimal set is defined as

$$\mathfrak{P}* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x'} \in \Omega, f(\mathbf{x'}) \preceq f(\mathbf{x})\}$$

and the Pareto front

$$\mathfrak{P}\mathfrak{F}* = \{f(\mathbf{x}) \mid \mathbf{x} \in \mathfrak{P}*\} \tag{2}$$

Now we need a multi-objective algorithm for the problem. An evolutionary multi-objective metaheuristic called 'NSGA-II' which stands for non-dominated sorting genetic algorithm II [26] and currently seems to be the most popular[18] will be selected for our problem, as the 'SGA' (explained in section 2.3.2) has been one of the most used and successful algorithms we have used, having many similarities with 'NSGA-II'.

'NSGA-II' starts with a population of individuals P_t. From this one it creates a offspring population Q_t, which is created iteratively following three steps that will be repeated until the offspring population Q_t has the same individuals as the parent population P_t. In the first step called 'Tournament Selection' DNA of different members of the population are compared randomly with the strongest ones being used to reproduce in the next phase. In the 'Crossover' step, the DNA from the individuals selected from the phase before is mixed in some random or predefined way. After this step, random mutations in the genes can occur in the 'Mutation' step. These steps can also be found in the 'SGA' algorithm.

Once our offspring is created, we have a new population which is the sum of the individuals in P_t and Q_t, which we call R_t. Every individual is then ranked and sorted based on his performance on defined target indicators in the 'Non-dominated sorting' phase. This ranking occurs in different fronts F_i which are the actual Pareto frontier solutions sorted by their dominance. The new population is chosen by picking the best individuals from the best (dominating) fronts, until the size of the parent population P_t is reached. Because we have to exactly get a number of individuals equal to P_t, most of the times we have to choose the best individuals inside of a front, in this case we use the 'Crowding distance sorting' mechanism. This one measures the cuboid size defined by the locations of the closest neighbors (from the same rank) of a solution in the objective space, where larger values for the cuboid are preferred as this indicates that the solutions are located in areas of the search space that are not crowded. With the new population P_{t+1}, we then start the process again creating an offspring population Q_{t+1} and continuing the procedure as explained. This operation is shown in Figure 9.

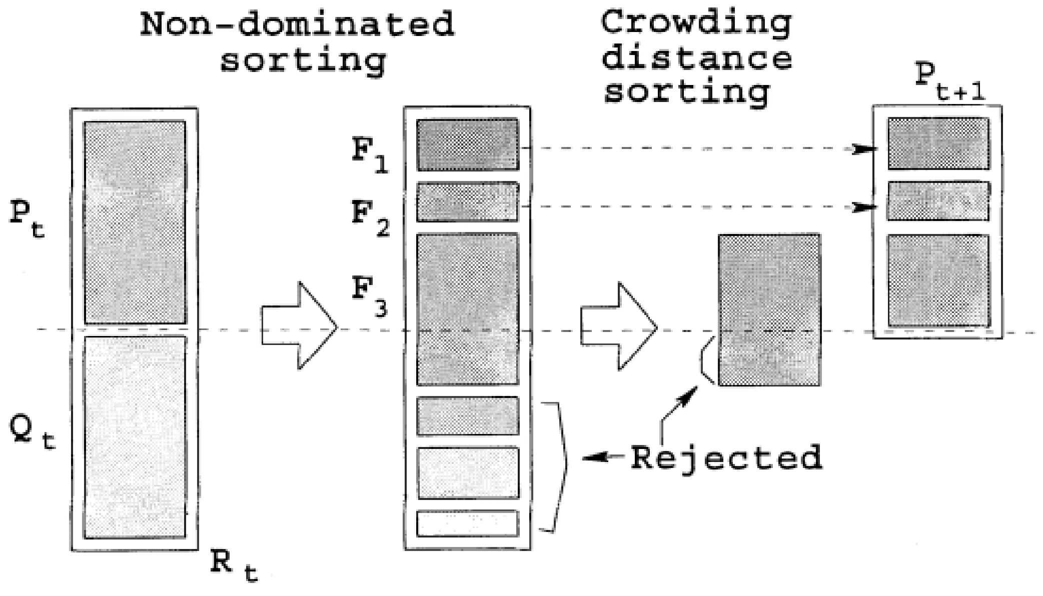


Figure 9: 'NSGA-II' procedure[26]

To develop our multi-objective problem in *Flappy*, we must now choose our objective functions $f(\mathbf{x})$. The first objective $f_1(\mathbf{x})$ will be the "minimum cable length" calculation that we developed in section 2.3.2 into *Flappy*. The second objective $f_2(\mathbf{x})$ will be the calculation of the "maximal yield" (following the "*Flappy* logic" explained in section 2.3.2) which was already implemented in *Flappy*.

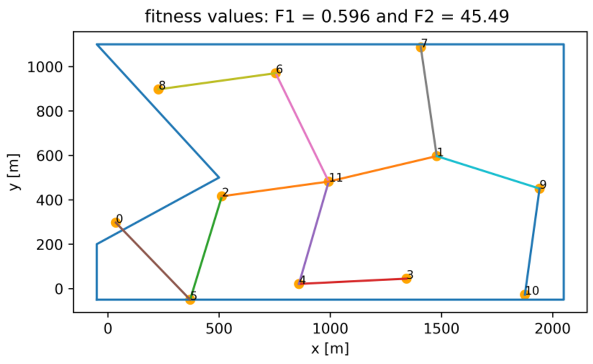
A new program will now be created in *Flappy* which follows the procedure of the *Flappy* basic structure that we saw in section 2.3.2. To this basic structure a second objective will be added with its constraints, together with other modifications in the optimizer of *iwes-opt*. Also, with the use of the optimization library *Pygmo*, the optimization algorithm 'NSGA-II' that was already explained above is selected for our program to use. This along with other multi-objective optimization utilities will allow us to calculate and analyze the Pareto front solutions of the multi-objective wind farm layout optimization problem with respect to the minimal cable length and the maximal yield, which is discussed in section 2.5.

2.5 Results and Discussion

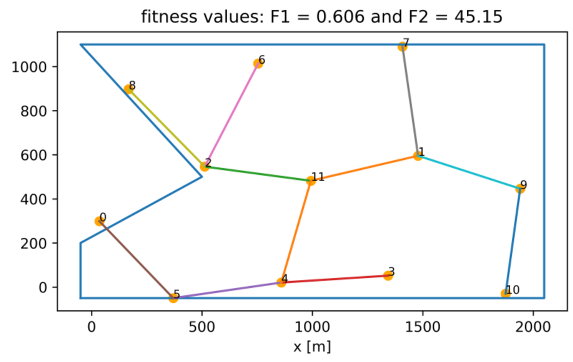
As mentioned in the introductory part, the objective of this internship is to create a program that minimizes cable length calculations, implement these into the IWES wind farm modeling python code *Flappy* and finally calculate and analyze the Pareto front of the multi-objective wind farm layout optimization problem with respect to maximal yield and minimal cable length. In the previous sections the results of many of these objectives have been already accomplished and explained, as the goals of the internship were more focused in "achieving an objective" rather than "analyzing data". Still an analysis of the performance of our Multi-objective program will be made, specially the Pareto front solutions produced for the wind farm layout optimization problem.

When running the multi-objective optimizer, *Flappy* is being fed0 with the objective functions to maximize yield and minimize cable length. The program will then use the 'NSGA-II' algorithm from *Pygmo* to create a Pareto Frontier of solutions (see section 2.4)

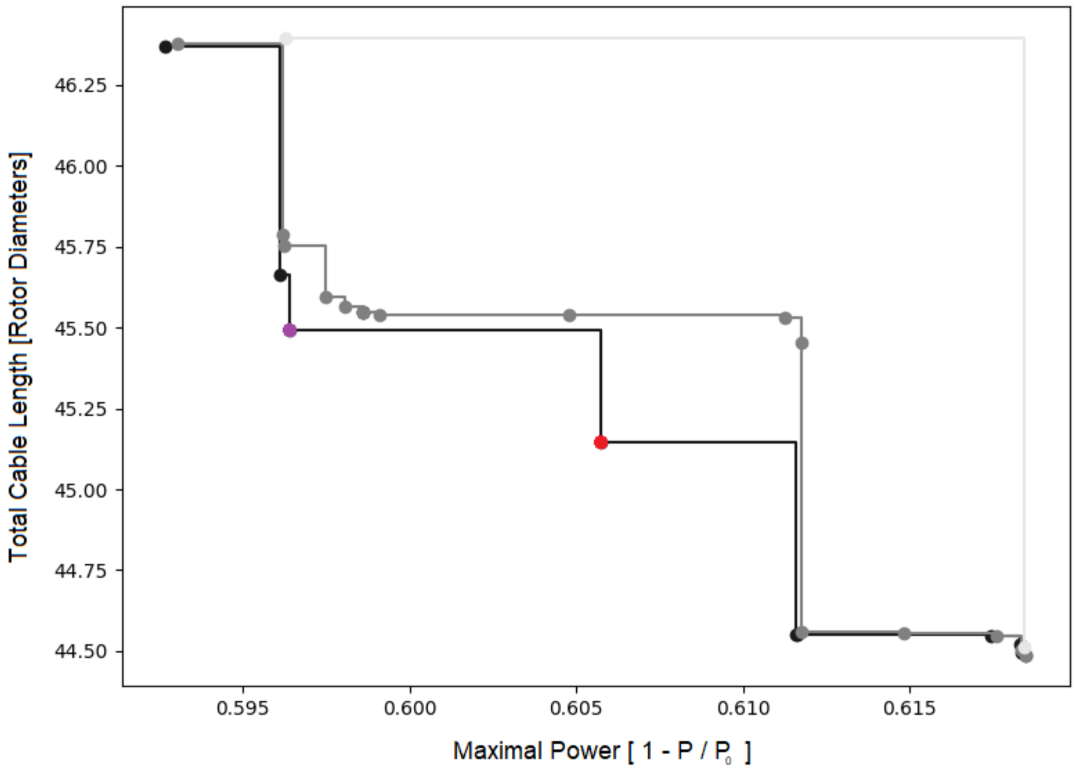
A Pareto front can be seen in Figure 10. In it multiple fronts can be seen. Each domination front is connected by straight lines which are all horizontal or vertical on the coordinate plane. The darkness of the lines indicates their ranking dominance over other fronts (the darkest line being the non-dominated Pareto front). This horizontal and vertical line connection has the purpose of better visualizing the dominance of the points of a front in each of the objective function directions (x and y axis) over other points outside this front. This also means that if a solution point from behind a Pareto front were to cross this line, it would become part of the Pareto front



(a)



(b)



(c)

Figure 10: Pareto front solutions plot of the multi-objective wind farm layout optimization problem with respect to maximum power (unit '1 - (Power dived by the Rated Power)') and minimal cable length (in units of 'Rotor Diameters'). Each individual (solution points in the graph) represents a particular wind farm layout (containing a specific decision vector). The layout of the violet point in the graph is shown in (a), and the layout of the red point in (b).

as it would not be completely dominated by all the points in this front.

Each point in the figure corresponds to an actual wind farm position layout, as the ones seen in Figure 10a and 10b. The relative positions of the turbines (vertices) in the x,y coordinates does not change in a great manner in this example, as usually happens with the layouts of evolved wind farm optimization problems. But as we can see in this example, these small changes in the positions of the turbines have changed the cable connections (edges) and therefore the structure of the minimum spanning tree in figures 10a and 10b.

After an analysis of the multiple multi-objective layouts and their corresponding Pareto fronts, the following results about the behavior of these can be concluded (and the results seen in Figure 11).

Our population defines the number of solutions (points in the graph) that will be evolved by the optizer and appear in the final graph. Because this number of population is user-defined, the aim would be to have all of these population individuals in just one Pareto front of solutions, instead of having multiple fronts (as dominated solution fronts are not going to be used), providing this way more optimal "trade-off" solutions, which is the final objective of multi-objective optimisation.

What can be observed in the results is how the manipulation of certain parameters affect the Pareto front solutions. Firstly and as could be expected, when the number of generations is modified this affects the quality of the results in the Pareto fronts. In the case of high number of generations, solution graphs with just one Pareto frontier tend to appear as seen in Figure 11b. In the case of low number of generations, a graph like the one shown in Figure 11a appears frequently, which shows that the optimization process has not yet reached the level where it approaches optimal solutions, generating many fronts which dominate each other.

Secondly, and as explained before, when we increase the population of solutions, generating more points in the graph, it becomes more difficult to position them all in the same Pareto front (requiring more generations for that purpose), so generally more pareto fronts can be seen as shown in Figure 11d. Finally also the number of turbines in our calculations, and the density of our wind farm layout will also affect the Pareto fronts solutions, in the way that it may be required a higher number of generations until better graph solutions appear approximating a one Pareto front scenario (see Figure 11c).

There is a range of different utilities that are implemented together with the program to better analyze solutions. Like returning the best N individuals out of a multi-objective population, computing the ideal point (contains the minimum value of the objective functions of the input points) and the nadir point (point that has the maximum value of the objective function in the points of the non-dominated front), or comparing two elements in the graph to obtain which one dominates which, between others utilities.

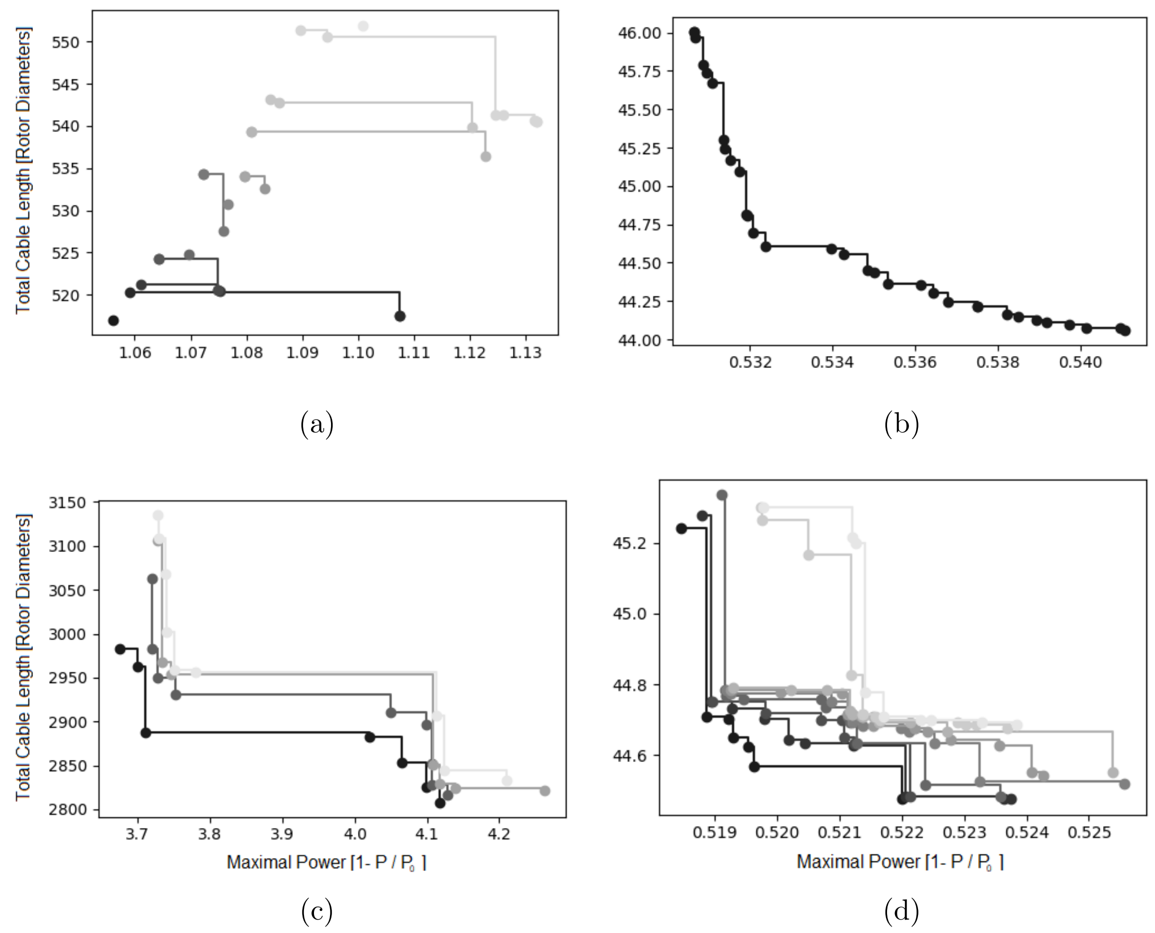


Figure 11: Pareto front solutions plots of the multi-objective wind farm layout optimization problem with respect to maximum power (unit '1 - (Power dived by the Rated Power)') and minimal cable length (in units of 'Rotor Diameters'). Graph (a) containing 12 turbines, 50 generations and 32 population, (b) containing 12 turbines, 700 generations and 32 population, (c) 48 turbines, 150 generations and 32 population, and (d) 12 turbines, 150 generations and 100 population.

3 Experience gained during the internship period

During my internship period there is much I have learned not only about the subject of optimization, wind energy and other programming and computing skills, but also about working environments, working and interacting with other and even my own capabilities.

My internship began unfortunately with the rise of the COVID-19 pandemic, so I was therefore forced to start my internship from home. But nevertheless, I was able to experience how a working environment at a research organization like Fraunhofer is. The need of good communication with staff members, making sure all formalities, legalities and organizational documents are being performed and provided. Communications with my supervisor, having a clear idea of what my working objectives are and asking for help with tasks when necessary. And finally communication with other research members of Fraunhofer and even ForWind (collaborating institute), which was unfortunately not possible by personal contact due to quarantine, but it was done in the form of weekly meetings which were held once a week with the CFD (computational fluid dynamics) group of Fraunhofer, and also once a week together with the ForWind group in the 'AG TWiSt' (Turbulenz Windenergie Stochastik) meetings. In these meetings we were required every certain time to present in front of the group our current work or to present contemporary research articles of interest, which makes the work experience more dynamic and a feeling that your work is being shared with others. I had a good experience overall at Fraunhofer.

I have acquired some experience with the programming language Python, and also its virtual environments and code editors, as *Flappy* is written all in Python. I have specially learned programming skills at a macro level using multiple files and libraries really exploiting the concept of inheritance, which was something new to me. Also I have learned about using the operating system Linux and the command-line interface shell.

Regarding the topic of my internship, I have learned about the world of algorithms and specially it's applications in computing. About the concept of single and multi-objective optimization and how are the ways it can be applied to real world problems. I gave a seminar presentation about the article "Heuristic Algorithms for the Wind Farm Cable Routing Problem"[27] where I presented several meta-heuristic techniques. I have also learned some added generals about Wind Energy, although my objectives were more centered around optimization algorithms.

My experience has taught me how to work in a big project together with others where I must continuously share and communicate my work, in our case by using the version control system 'Git' and the web-based repository 'GitLab'. On the other hand, I had to learn solve problems by myself. I was many times in my own (specially during this pandemic period), so I had to face problems that seemed scary about programming and computing, and learned many times it just requires more perseverance and hard work to get them solved, and that it is possible.

I would say that I have really enjoyed learning about optimization algorithms and programming with them in Python. It was a very creative environment which was most of the times satisfying (and frustrating), but I am still somewhat scared about getting into deeper waters with programming and algorithmic techniques.

4 Objectives and achievements of the internship

The objective of my internship was to implement cable length calculations into the IWES wind farm modeling python code *Flappy*, and to calculate and analyze the Pareto front of the multi-objective wind farm layout optimization problem with these. I believe that I was able to accomplish these objectives and that the final results are quite satisfactory. As seen along section 2, a minimum spanning tree that was able to calculate shortest cable connection paths was successfully developed using Prim's algorithm. And the following implementation into *Flappy* was also quite positive, having made some testing to see how it would behave with different algorithms. And finally it was possible to create a problem that calculates the Pareto fronts of a multi-objective layout problem with respect to maximal yield and minimal cable length, and an analysis of these Pareto front solutions was made to observe its performance which seemed to be quite decent.

The internship was a preparation for my thesis, which I am going to also do with Fraunhofer IWES and inside the same area of wind farm optimization, so I would say that the knowledge I have acquired of algorithms, optimization techniques and wind energy will be vital for my next step during my thesis. Also, although I haven't been able to work in an actual office due to the pandemic conditions, I think I've achieved the aim of getting to know a real working environment from a research institute like Frauhofer, and where I have felt really comfortable and warm after all.

For the future many objectives can be created for further developing the multi-objective optimization of wind farms. Performing more analyses with different objective functions (e.x. maximal turbine life, loads...), using different decision vectors (apart from turbine position coordinates), using other multi-objective algorithms and improving the parallel computation of these algorithms through exploiting parallel population dynamics and the possible interactions between them (e.x. 'The Generalized Island Model'[28]) are between the main future objectives.

5 Conclusion

I have been doing my Bachelor Internship for my studies in Engineering Physics at the Fraunhofer IWES institute in Oldenburg. Here I have been working developing a program that calculates the shortest cable connections between turbines, implementing these calculations into the wind farm modeling python code *Flappy*, and finally calculating and analyzing the Pareto front of the multi-objective wind farm layout optimization problem with respect to maximal yield and minimal cable length. I am very happy with my final experience at Fraunhofer, where I have learned and improved many skills during my internship, and I have been able to apply some of the knowledge that I gained during my main studies.

References

[1] Jonas Schmidt, Lukas Vollmer, and Martin Dörenkämper. *Flappy - farm layout program in python [Computer software]*. 2019. URL: https://gitlab.cc-asp.fraunhofer.de/iwes-cfsd/windsite-assessment/flappy/-/wikis/home. Fraunhofer IWES.

[2] *Fraunhofer IWES - About. YouTube*. 2020. URL: https://www.youtube.com/channel/UC7rUlm2KXrMT9fNOdtvJbKQ/about.

[3] *Fraunhofer Institute for Wind Energy Systems*. 2020. URL: https://www.iwes.fraunhofer.de/.

[4] Margarita Ortega-Izquierdo and Pablo del Rio. "An analysis of the socioeconomic and environmental benefits of wind energy deployment in Europe". In: *Renewable Energy* 160 (2020), pp. 1067–1080. Elsevier.

[5] Martina Fischetti. "Improving profitability of wind farms with operational research". In: *Impact* 2019.1 (2019), pp. 30–34. Taylor & Francis.

[6] Alan Gibbons. *Algorithmic graph theory*. Cambridge university press, 1985.

[7] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 2013.

[8] Thomas H Cormen et al. *Introduction to algorithms, Second Edition*. MIT press, 2001.

[9] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50. JSTOR.

[10] Robert Clay Prim. "Shortest connection networks and some generalizations". In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401. Nokia Bell Labs.

[11] Mohamed Tifroute and Hassane Bouzahir. "Optimization for Onshore Wind Farm Cable: Connection Layout using ACO-AIA algorithm". In: *MATEMATIKA: Malaysian Journal of Industrial and Applied Mathematics* 35.1 (2019), pp. 67–82.

[12] Constantin Berzan. "Algorithms for Cable Network Design on Large-scale Wind Farms". In: 2011.

[13] Pedro Santos Valverde, António JNA Sarmento, Marco Alves, et al. "Offshore wind farm layout optimization–state of the art". In: *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers. 2013.

[14] Sascha Gritzbach et al. “Towards negative cycle canceling in wind farm cable layout optimization”. In: *Energy Informatics* 1.1 (2018), pp. 183–193. SpringerOpen.

[15] Patrik Fagerfjäll. “Optimizing wind farm layout: more bang for the buck using mixed integer linear programming”. In: *Chalmers University of Technology and Gothenburg University* (2010), p. 111.

[16] *GeeksforGeeks - Prim’s Minimum Spanning Tree (MST) — Greedy Algo.* 2020. URL: `https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/?ref=lbp`.

[17] X. Yang. “Metaheuristic Optimization”. In: *Scholarpedia* 6.8 (2011). revision #91488, p. 11472. DOI: `10.4249/scholarpedia.11472`.

[18] F. Glover and K. Sörensen. “Metaheuristics”. In: *Scholarpedia* 10.4 (2015). revision #149834, p. 6532. DOI: `10.4249/scholarpedia.6532`.

[19] Dr. Jonas Schmidt. “Webinar: flappy v0.3”. In: (2020). Fraunhofer IWES [PowerPoint slides].

[20] Dr. Jonas Schmidt. “Flappy Logic”. In: (2020). Fraunhofer IWES [PNG file].

[21] Francesco Biscani and Dario Izzo. “A parallel global multiobjective framework for optimization: pagmo”. In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: `10.21105/joss.02338`. URL: `https://doi.org/10.21105/joss.02338`. The Open Journal.

[22] F. Stonedahl and U. Wilensky. *NetLogo Simple Genetic Algorithm model.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2008. URL: `http://ccl.northwestern.edu/netlogo/models/SimpleGeneticAlgorithm.`.

[23] Mohammed Ghasem Sahab, Vassili V Toropov, and Amir Hossein Gandomi. “A review on traditional and modern structural optimization: problems and techniques”. In: *Metaheuristic applications in structures and infrastructures* (2013), pp. 25–47. Elsevier.

[24] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization.* Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, Computer Engineering Department, Kayseri/Turkey, 2005.

[25] Gustavo R Zavala et al. “A survey of multi-objective metaheuristics applied to structural optimization”. In: *Structural and Multidisciplinary Optimization* 49.4 (2014), pp. 537–558. Springer.

[26] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197. IEEE.

[27] Davide Cazzaro, Martina Fischetti, and Matteo Fischetti. “Heuristic algorithms for the Wind Farm Cable Routing problem”. In: *Applied Energy* 278 (2020), p. 115617. Elsevier.

[28] Dario Izzo, Marek Ruciński, and Francesco Biscani. “The generalized island model”. In: *Parallel Architectures and Bioinspired Algorithms.* Springer, 2012, pp. 151–169.