wflo: A New Standard for Wind Farm Layout Optimization in R

Carsten Croonenbroeck
Rostock University

David Hennecke
Rostock University

Abstract

We provide a new and unified standard for the field of wind farm layout optimization (WFLO). The problem of optimally arranging turbines within a given area (e.g. irregularly shaped land owned by the potential wind farm operator) has been researched for quite some time. Dependent on the wind direction, a single turbine A may or may not be located within the wake of another turbine B, leading to a (potentially) unnecessary loss of electricity produced at A. Arranging the farm in a way that rules out potential wake losses is a difficult problem because of the interdependencies of the turbines. A comprehensive methodology to solve this problem is yet to be found. Our R package providing both, a high-quality data set as well as useful functionality, will enable researchers to focus on their actual methodology contributions. The package also serves as a benchmark to empirically compare the outcomes of contributed solutions.

Keywords: WFLO, wind farm layout optimization, NP-hard, package, data set, benchmark, R.

1. Introduction

Wind farm layout optimization (WFLO), i.e. the question of how to optimally arrange a set of wind turbines inside a wind farm, is a problem that has been analyzed for quite some time. In mathematics, this problem is typically seen as a constraint optimization (i.e. maximization or minimization) task. However, while the objective space encompasses a typically rather simple surface in $\mathbb{R}^3$, "feature space" is a high-dimensional, mostly non-continuous (hence, non-differentiable), multimodal space of ex ante unknown complexity. Furthermore, as several points in that space are sought-after and these points possess a strongly interdependent behavior due to wake effects in the wind farm, the WFLO problem is considered to be $\mathcal{NP}-$hard (see Garey and Johnson 1979 for a definition).
Approaches to solving this problem can not only be classified a) according to the wake model used, but also b) according to the class of optimization approaches (gradient-based approaches and gradient-free algorithms), or c) according to the target function class. Most approaches to WFLO incorporate a physical model for inner-farm wake effects. The model by Jensen (1983) is most popular, as it provides a quite accurate approximation of measured wakes in actual farms, despite the model's simplicity. See Shakoor, Hassan, Raheem, and Wu (2016) and the references therein for a comprehensive overview of WFLO literature employing the Jensen model. However, some contributions employ other wake models. For instance, Kirchner-Bossi and Porte-Agel (2018) use a Gaussian wake model.

Most optimizers employ gradient-free methods, as these methods (e.g. metaheuristic algorithms such as genetic algorithms (GA) or simulated annealing) perform fairly well at exploring the feature space, given that this space is not of too high dimension. For example, Chen, Li, Jin, and Song (2013), Park, An, Lee, Jung, and Lee (2019), and Yang, Kwak, Cho, and Huh (2019) use these methods. Higher-dimensional spaces however are the domain of gradient-based approaches, as Thomas and Ning (2018) point out.
Many researchers use wind farm efficiency or annual energy production (AEP) as their target function, e.g. Park *et al.* (2019). However, a more realistic approach considers what wind farm operators are really after: Earning money. Therefore, as Giovanni (2019) points out, AEP maximization is not an immediately meaningful goal. Instead, researchers recently tend to use economically driven target functions. Wu, Zhang, Wang, Wang, and Feng (2020) use a profit-driven function, which also enables them to flexibly incorporate cost or revenue influencing drivers of the optimization problem, e.g. inner-farm wiring. As an alternative, Yang *et al.* (2019) discuss wake effect uniformity instead of AEP or profit maximization as an alternative target.
Antonini, Romero, and Amon (2020) point out the complex terrain aspects of the WFLO problem, an issue that Thomas and Ning (2018) try to relax. However, as this is just a means of modifying the problem at hand to ease its handling, it even emphasizes the necessity for a unified setting and goal in order to focus on the methodology to solve that goal. Thus, researchers even start WFLO competitions to have participants compete on a unified setting. Baker, Stanley, Thomas, Ning, and Dykes (2019) show the outcome of such a competition. Their competition does not only provide a unified target and unified data, it also serves as a benchmark that every participant can use to evaluate his/her own methodology idea/contribution.
Using `R` by R Core Team (2017), package **windfarmGA** provides a genetic algorithm implementation to deal with WFLO. The methodology can be explored at `https://windfarmga.shinyapps.io/windga_shiny/` as well. However, as this is an approach at solving the problem, it still does not fully describe the problem itself, nor does it provide a data set or a benchmark. At this point, package **wflo** (see `https://CRAN.R-project.org/package=wflo`) comes into place: It provides high-quality and highly accurate data, which can be used for both, developing a methodology as well as benchmarking it. It does not, however, provide an actual optimization solution. **wflo** is merely a sandbox for solution researchers. The package also provides a set of modular functions around the WFLO problem, providing an economically driven target function implementing Jensen's multiple wake effects model, plug-in functions for cost and revenue and a set of parameters. In this article, we discuss the components of package **wflo** and its intended usage in great detail. Section 2 presents the data and sheds light on the functions and their utility. Section 3 provides usage examples and section 4 draws a brief conclusion.

2. Data and Functions

2.1. Data Description

Package **wflo** deals with a high-quality data set available in a list object `FarmData` after the package is installed and loaded within `R`. The data stem from DWD (Deutscher Wetterdienst, German Meteorological Service) and NASA (National Aeronautics and Space Administration,

SRTM data set, Shuttle Radar Topography Mission) and have been preprocessed, i.e. the meteorologic measurement station data are spatially interpolated using Geographic Information System (GIS) software[1] and the wind direction data are temporally averaged by transforming polar coordinates to cartesian coordinates and then computing the vector averages.
The package comes with a subset of the entire data set. Due to CRAN[2] package size limitations, it is not possible to enclose the entire data set within the package. However, the full data set can be downloaded conveniently using the built-in function `AcquireData()`. As a parameter, the function accepts the directory to where the file will be saved. For example, the user may choose to use the current working directory obtained via `getwd()` as a target directory. After file `FarmData.RData` is downloaded, it is loaded automatically. While the reduced, built-in data set is accessible via list object `FarmData`, once the full data set is available, it replaces the built-in data set by being embedded into user writeable environment `e`, easily accessible via `e$FarmData`. Therefore, one object does not mask the other. All subsequent functions check for the full data set being present in environment `e` accordingly. On load, the package checks whether the file is present in the current working directory and if so, loads the file automatically. Otherwise, **wflo** automatically falls back to basic mode, using the built-in data set. The functionality described in this article works perfectly with the built-in data set. Downloading the full file is only necessary for cross-checks or advanced tests of a self-implemented optimizer. WFLO research is perfectly possible with the built-in data set.
List object `FarmData` (or `e$FarmData`) has seven slots. Each of them contains a matrix that has 4,400 $\times$ 3,250 elements in full mode and 25 $\times$ 25 elements in basic mode (built-in data set). The subset in basic mode is "carved" out of the entire data set at position [2000:2024, 2000:2024], so it is nested in the full data set. The 4,400 $\times$ 3,250 elements in full mode cover the entire area of Germany. They represent a raster data set at a rather accurate resolution of 200 $\times$ 200 meters. Slot 1 of the list object, named `AdjustedYield`, contains adjusted potential AEP ("yield") for German onshore sites. AEP is computed based on FGW technical guidelines (Fördergesellschaft Windenergie und andere Dezentrale Energien, support organization for wind energy and other types of decentralized energy). Those technical guidelines are mandatory to use in Germany, are stipulated in the German Renewable Energy Act (EEG 2017), and are obtainable at `https://wind-fgw.de/shop/technical-guidelines/?lang=en`. The guidelines themselves are based on IEC 61400-12-1, the international standard on "power performance measurements of electricity producing wind turbines" by the International Electrotechnical Commission, IEC. According to EEG (2017), AEP computed that way must be multiplied by a correction factor. That factor is based on a specific location's quality in order to compensate for several market regulations in Germany. Users can interpret the contained values immediately as AEP in "megawatt hours per year" (MWh/a), however we decide to name that variable *adjusted* yield, as researchers computing AEP on their own may obtain slightly different values. Figure 1 visualizes the data.
Slot 2, named `WindSpeed`, contains temporally averaged wind speed in meters per second (m/s), while slot 3, named `WindDirection`, contains temporally averaged wind direction in degrees (azimuth). Figure 2 presents both in a joint image. Slot 4, named `SDDirection`, contains the standard deviations of wind directions after temporally averaging. So far, stan-

[1] We use the inverse distance weighted (IDW) method, see `https://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-idw-works.htm` for documentation.
[2] Comprehensive R Archive Network, the worldwide repository network for R packages.

dard deviations are not used anywhere in the package. However, they may provide important insight into the volatility of wind directions at each point and may thus be relevant for, e.g., computing confidence bands. Slot 5 (`Elevation`) holds the terrain elevation in meters, slot 6 (`Slope`) provides terrain slope in degrees, and slot 7 (`SlopeDirection`) gives the direction of hillside, in degrees as well. Figure 3 presents a joint image plot for the data in the seven slots (built-in data set).

The second data object in the `e` environment installed by **wflo** is `e$FarmVars`. This list object contains a set of variables required for the functions of **wflo** and, among other things, controls the "data window" of consideration if the full data set is present. `$StartPoint`, `$EndPoint` and `$Width` control which area from the data set is used as the current wind farm area. In general, `$Width` describes width (and height, as the area is assumed to be a square) of the area used, in raster points. As the raster resolution is 200 m, the default value of `$Width` = 25 results in a square of 5 × 5 km. For convenience, `e$FarmVars` also contains the farm size in meters in its variable `$MeterWidth`, which is computed as

```
e$FarmVars$MeterWidth <- e$FarmVars$Width * 200.
```
(1)

In full mode (downloaded data set), `$StartPoint` = 2000, while in basic mode (built-in data set), `$StartPoint` = 1. In both cases, `$EndPoint` is computed based on `$StartPoint` and `$Width` following

```
e$FarmVars$EndPoint <- e$FarmVars$StartPoint + e$FarmVars$Width - 1,
```
(2)

which results in `e$FarmVars$EndPoint` = 2024 in full mode and `e$FarmVars$EndPoint` = 24 in basic mode. Furthermore, `e$FarmVars` contains `$MeterMinDist`, which holds the minimum distance between any two turbines in the wind farm. Its default value is 500 meters. As internally, the wind farm is considered to be a unit square (automatically and irrespective of the current `$StartPoint`, `$Width`, `$EndPoint`, and `$MeterWidth`), `e$FarmVars` also contains a variable `$MinDist` for the minimum distance in unit space as

```
e$FarmVars$MinDist <- e$FarmVars$MeterMinDist / e$FarmVars$MeterWidth,
```
(3)

which consequently defaults to 500 / 5000 = 0.1, i.e. one tenth of the farm square's height or width. `e$FarmVars` also contains variables `$z`, `$z0`, and `$r0`, which contain the turbines' hub height in meters (defaults to `e$FarmVars$z` = 100), the terrain's roughness length required for Jensen's wake model (defaults to `e$FarmVars$z0` = 0.1), and the turbines' rotor radius in meters (defaults to `e$FarmVars$r0` = 45). Users should adjust these values according to their turbine specifications or use the default values for benchmarking. Furthermore, `e$FarmVars` contains the variables `$UnitCost` and `$Price`. `$UnitCost` is meant to carry a single turbine's yearly cost. If, for example, total turbine installation costs are 2 mio. EUR and the projected life span of the turbine is 20 years, then the yearly cost is roughly 100,000 EUR, which is also the default value for this item. Additional cost components such as wiring, (non-linear) amortization, maintenance, opportunity cost, or others can be modeled using an alternative cost function which can be plugged in easily (see below). Similarly, `$Price` contains the sale price per megawatt hour. This defaults to an empirically reasonable average of 100 EUR. However, if revenue is to be modeled in a more sophisticated fashion than using "constant price times sale volume", the necessary adjustments are easily performed (once again, see below). Finally, `e$FarmVars` contains a vector object `$BenchmarkSolution`, which holds 40 values

(20 pairs of x and y coordinates) representing a setup of 20 turbines within the standard data set terrain. It has been obtained using optimizer `genoud()` of package **rgenoud** by Mebane and Sekhon (2011) and provides a standard of comparison for future optimizers or can be used as starting points for further exploring the feature space.
Changing, say, `$r0` or `$z`, but more immediately, `$StartPoint` and `$EndPoint`, turns the problem into an entirely new setup. For instance, assuming the full data set is present, selecting

```
R> e$FarmVars$StartPoint <- 1800
R> e$FarmVars$Width <- 100
R> e$FarmVars$EndPoint <- e$FarmVars$StartPoint + e$FarmVars$Width - 1
R> e$FarmVars$MeterWidth <- 200 * e$FarmVars$Width
R> e$FarmVars$MinDist <- e$FarmVars$MeterMinDist / e$FarmVars$MeterWidth
```

provides an entirely new (and larger) testing ground. A huge multitude of optimization areas is accessible that way, even allowing to immediately select costal regions, mountain regions, or areas known to be especially apt or unsuitable to wind power plants.

2.2. Functions

The central function to **wflo** is `Profit()`. It takes a set of points in the unit square (turbine locations) and based on the adjusted AEP in the farm specified via the `e$FarmVars` settings object, checks whether the layout is valid (i.e. minimum distances are met for all turbines), computes multiple Jensen wake penalties, generates the total marketable power production of the specified layout, takes cost and sale price into consideration and finally computes the farm's economic profit. Since most numeric optimizers by default operate as a minimizer, `Profit()` returns the negative profit, i.e. a negative number for positive actual profit values and a positive number if cost is greater than revenue (negative actual profit).
`Profit()` calls `Cost()`, which is a stub function only returning the `$UnitCost` value from the `FarmVars` object. For more sophisticated cost components, users should replace the `Cost()` function by their own cost model. This can be done by embedding another function `Cost()` in the `e` environment. `Profit()` will then operate on it immediately. Note that `Cost()` must be a function that is provided with `x` and `y` (the turbine coordinates in the unit square) and both, `x` and `y`, must be $\in \mathbb{R}^n$, i.e. both variables are vectors of length `n` (the number of turbines). An example for such a workflow is given in Section 3. Similarly, `Yield()` is a stub looking up the adjusted AEP based on a given turbine location. If yield or revenue should follow a more sophisticated model, replacing `Yield()` by that model would be the way to go. Again, a plug-in yield function must be embedded in `e`, and `Profit()` will use it accordingly. In contrast to `Cost()`, `Yield()` must be a function provided with `x`, `y`, and `AEP`. Here, `x` and `y` must be $\in \mathbb{R}$, i.e. single-valued variables for one turbine at a time. `AEP` will carry the AEP values for the wind farm. The user function is not required to evaluate this, the object is just provided for convenience. Note, however, that according to the calling convention, the function must have a prototype declaration that expects that object.
`PlotResult()` is a convenience function that takes an optimizer result, i.e. an object as returned by `optim()`, visualizes the adjusted yield "landscape" in the current wind farm setting, superimposes a contour plot and a vector field representing the wind directions and then draws the provided points (turbines). For example, a usual optimization run can be performed in the first place via

```
R> library(wflo)
R> NumTurbines <- 4
R> set.seed(2763)
R> Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
+    method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
+    upper = rep(1, NumTurbines * 2))
R> Result

$par
[1] 0.004135149 0.182979427 0.833207494 0.364683764 0.038710320
[6] 0.036169698 0.700000355 0.101751237

$value
[1] -2103136

$counts
function gradient
      63       63

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Afterward, a call to `PlotResult(Result)` will produce the image in Figure 4. Lighter shades of blue indicate larger values of AEP, darker ones indicate low AEP values. The actual AEP values are also indicated via the contour lines. The area selected represents a rather complex AEP "terrain", a challenging task to optimizers. The light arrows indicate the main wind directions, while the grey arrows add/subtract half the standard deviation to/from the main direction to indicate the direction volatility. The gold dots, finally, show the turbines' arrangement. Figure 5 shows the alternative testing ground presented in Section 2.1, together with a (not necessarily optimal) solution of 20 turbines (gold dots).

Another "post-optimization" result inspection functionality is provided by `ShowWakePenalizers()`. A call to `ShowWakePenalizers(Result)`, where again `Result` is the optimizer's output (list object containing at least a "`$par`" slot), draws a reduced field (neglecting terrain and contour, yet imposing wind directions) and in gold, draws all points (turbines) that are in the wake of other turbines (called "sufferers" in the plot's legend) while in grey, all points are drawn causing wake effects on other points (called "causers"). The Jensen cones imposed on the sufferers are drawn in red. Figure 6 shows an example. Using `ShowWakePenalizers()`, it can be easily inspected whether the optimizer puts emphasis on avoiding wake effects or, vice versa, whether an optimizer finds an optimal arrangement *although* it means that some points cause wake effects on others.

Function `ProfitContributors()` computes the profit contribution of each point (turbine) as a part of an optimization result. That way, the setup location and wake influence of each point can be analyzed in greater detail. Calling `ProfitContributors(Result)` returns a matrix with two columns and as many rows as points present in the provided list object. The first

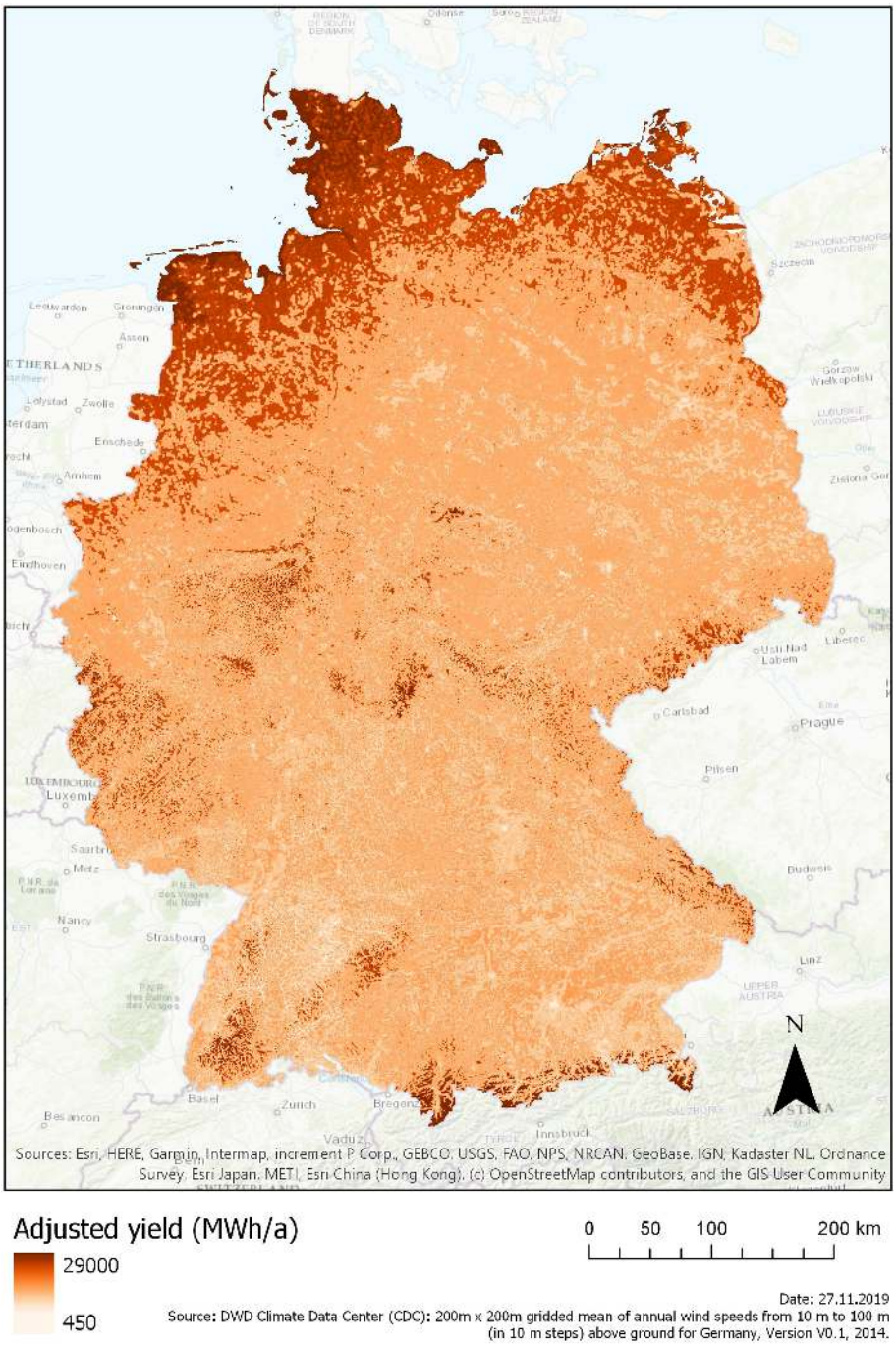

Figure 1: Adjusted yield (AEP) in Germany.

column represents turbine IDs, the second column contains the respective profit contributions of each point. If the solution provided is valid in terms of the minimum distance criterion, the sum of values in the second column will be identical to the (absolute value of) the returned value of `Profit()`.

3. Use Cases

It is the foremost intention of **wflo** to be compatible to the `R` optimization quasi-standard, i.e. optimizers are minimizers, they expect start values and the function to optimize as their parameters and that function is to be provided with nothing but a vector object of values. The optimizer returns a list which contains at least `$par` and `$value`. With **wflo**, researchers in WFLO can focus on developing and providing a standard-compliant optimizer and use it to measure its performance on the wind farm setting provided by the package. An actual **wflo** workflow of course implies that the user utilizes a self-developed plug-in optimization function. Since none is available here, we merely show a set of sample workflows based on pre-existing optimizers. As can be seen, a **wflo** use case is as simple as just minimizing `Profit()` using an optimizer. Post-optimization analysis can be performed using `PlotResult()`, `ShowWakePenalizers()`, `ProfitContributors()`, and all arbitrary analysis procedures for optimizers.
In `R`, most optimizers observe the quasi-standard described above. The only additional requirement for **wflo** is that the optimizer respects box constraints. However, for optimizers that do not, a crude wrapper is presented below. As a first use case, package **nloptr** by Johnson and Ypma (n.d.) provides a controlled random search optimizer (CRS). It can be utilized like this:

```
R> library(nloptr)
R> set.seed(1357)
R> Result <- crs2lm(x0 = runif(NumTurbines * 2), fn = Profit,
+    lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
R> Result

$par
[1] 0.2394255 1.0000000 0.3670594 0.1329830 0.5197366 0.0000000
[7] 0.0000000 1.0000000

$value
[1] -2359115

$iter
[1] 10000

$convergence
[1] 5

$message
[1] "NLOPT_MAXEVAL_REACHED: Optimization stopped because maxeval (above) was reached."
```

Many WFLO researchers use genetic algorithms to optimize their wind farms. In `R`, a sophisticated GA implementation is provided via package **rgenoud**:

```
R> library(rgenoud)
R> set.seed(1357)
```

```
R> Dom = cbind(rep(0, 2 * NumTurbines), rep(1, 2 * NumTurbines))
R> Result <- genoud(fn = Profit, nvars = 2 * NumTurbines,
+    starting.values = runif(NumTurbines * 2), Domains = Dom,
+    boundary.enforcement = 2, print.level = 0)
R> Result

$value
[1] -2445920

$par
[1] 0.63250725 0.04927494 0.74812584 0.59172686 0.58084278 0.41179594
[7] 0.08776379 0.10433266

$gradients
[1]   0   0   0   0   0   0 NaN   0

$generations
[1] 13

$peakgeneration
[1] 2

$popsize
[1] 1000

$operators
[1] 122 125 125 125 125 126 125 126   0
```

Finally, particle swarm optimization (PSO) is another metaheuristic frequently used in WFLO. Package **pso** by Bendtsen (n.d.) provides a straight-forward implementation:

```
R> library(pso)
R> set.seed(1357)
R> Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
+    lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
R> Result

$par
[1] 0.9153697 0.3092070 0.9651576 0.0000000 0.5485764 0.0000000
[7] 0.6916834 0.5013425

$value
[1] -2445920

$counts
 function iteration  restarts
    15000      1000         0
```

```
$convergence
[1] 2

$message
[1] "Maximal number of iterations reached"
```

For optimizers that do not provide box-constraint compliance, e.g. the SANN (simulated annealing) optimizer that is part of `optim()`, a simple wrapper can look like this:

```
R> lower <- rep(0, NumTurbines * 2)
R> upper <- rep(1, NumTurbines * 2)
R> Wrapper <- function(X)
+  {
+          xSel <- seq(from = 1, to = length(X) - 1, by = 2)
+          x <- X[xSel]
+          y <- X[xSel + 1]
+
+          if (any(x < lower) | any(x > upper) | any(y < lower) | any(y > upper))
+          {
+                  return(sum(rep(e$FarmVars$UnitCost, length(x))))
+          }
+
+          return(Profit(X))
+  }
R> set.seed(1357)
R> Result <- optim(par = runif(NumTurbines * 2), fn = Wrapper,
+    method = "SANN")
R> Result

$par
[1] 0.16629696 0.96090420 0.55578489 0.02630332 0.83219195 0.34124568
[7] 0.69785617 0.03611307

$value
[1] -2291240

$counts
function gradient
   10000       NA

$convergence
[1] 0

$message
NULL
```

Returning the sum of `$UnitCost` if any point violates the constraints may seem crude. More sophisticated ways to deal with that are up to the researcher. For now, this simple wrapper gets the job done.
Replacing `Cost()` and `Yield()` functions by user defined code is straightforward and enables the researcher to incorporate more sophisticated models for these components, still using the integrated **wlfo** workflow. For example,

```
R> e$Cost <- function(x, y) #x, y \in R^n
+  {
+          retVal <- rep(e$FarmVars$UnitCost, min(length(x), length(y)))
+          retVal[x > 0.5] <- retVal[x > 0.5] * 2
+          return(retVal)
+  }
```

embeds a cost function that overly punishes turbine locations on the right (or eastern) half of the unit square. There is no fundamental reason to this as it is just a crude and simple proof-of-concept example. In practice, users may want to setup a function that instead encompasses, e.g., (possibly non-linear) maintenance or opportunity cost. As to be expected, an optimizer run

```
R> set.seed(1357)
R> Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
+    lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
R> Result

$par
[1] 0.0527673590 0.1154515810 0.3739443954 0.1494660420 0.3895410254
[6] 0.4089666057 0.0000707834 1.0000000000

$value
[1] -2445920

$counts
 function iteration  restarts
    15000      1000         0

$convergence
[1] 2

$message
[1] "Maximal number of iterations reached"

R> rm(Cost, envir = e)
```

returns locations on the left (or western) half of the wind farm. Note the last code line to remove the superimposed cost function. A more realistic idea for a cost based model could

be that in many cases, turbine locations should a) keep their minimum distances, b) possess as little wake as possible, and c) still be as close to one another as possible to at the same time minimize wiring cost. A cost function that obtains that may be

```
R> e$Cost <- function(x, y)
+  {
+          n <- min(length(x), length(y))
+
+          retVal <- rep(e$FarmVars$UnitCost, n)
+
+          DistMat <- matrix(ncol = n, nrow = n)
+          for (i in 1:n)
+          {
+                  for (j in 1:n)
+                  {
+                          DistMat[i, j] <- sqrt((x[i] - x[j]) ^ 2 + (y[i] - y[j]) ^ 2)
+                  }
+          }
+          SumDist <- as.numeric()
+          for (i in 1:n) SumDist[i] <- sum(DistMat[i, ])
+
+          retVal <- retVal * SumDist
+          return(retVal)
+  }
R> rm(Cost, envir = e)
```

Similarly to `Cost()`, users can incorporate an alternative yield function that provides yield for each turbine individually. `Profit()` will automatically impose Jensen wake penalties on the result. For instance,

```
R> e$Yield <- function(x, y, AEP) #x, y \in R
+  {
+          return(x + y)
+  }
```

is again a crude example. The function does not even evaluate AEP and provides a strong tendency of the turbine layout result to locate turbines in the topright corner, where both, x and y, take their largest values, however still keeping their minimum distance and still subject to wake penalties. A run

```
R> set.seed(1357)
R> Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
+    lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
R> Result

$par
[1] 0.8 1.0 0.9 1.0 1.0 0.9 1.0 1.0
```

```
$value
[1] 399992.4

$counts
 function iteration  restarts
    15000      1000         0

$convergence
[1] 2

$message
[1] "Maximal number of iterations reached"
```

```
R> rm(Yield, envir = e)
```

returns a result as expected in that sense. Note, again, the last code line to remove the superimposed yield function.
The `value`s returned by optimizers operating on the standard cost and yield functions through `Profit()` can be interpreted as negative profits in currency (Euros) immediately. As pso, CRS, and genoud each return a good result of `-2445920` (pso setup shown in Figure 7), the setups found by these optimizers generate 2.4 mio. Euros of profit per year. The setup found via SANN generates 2.3 mio. Euros, a simple L-BFGS-B approach only 2.1 mio. Euros. Optimizers should compete in terms of the returned `value`s, higher (absolute) profits are better.
Using function `ProfitContributors()`, the influence of each turbine on the total profit can be analyzed using the following code chunk, while using the second column values as labels for `PlotResult()` results in Figure 8:

```
R> NumTurbines <- 4
R> set.seed(1235)
R> Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
+    method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
+    upper = rep(1, NumTurbines * 2))
R> Contribs <- ProfitContributors(Result)
R> Contribs

     Turbine   Profit
[1,]       1 473491.1
[2,]       2 499280.9
[3,]       3 554724.1
[4,]       4 324832.1
```

```
R> PlotResult(Result, DoLabels = TRUE, Labels = Contribs[, 2])
```

We emphasize that increasing the number of turbines to a more reasonable number increases feature space dimension, and with it, the necessity for increased iterations, population sizes,

generations, or whatever tuning parameter necessary for the optimizer to more profoundly explore the problem space. Gradient-based optimizers might come in handy, then. So far, for a 20 turbines setting, `e$FarmVars$BenchmarkSolution` provides an orientation (the plots are shows in Figures 9 and 10, respectively):

```
R> Result <- list(par = e$FarmVars$BenchmarkSolution)
R> Result$value <- Profit(Result$par)
R> Result

$par
 [1] 0.66681197 0.51920366 0.40001173 0.73883743 0.39199855 0.29543266
 [7] 0.83333332 0.36453646 0.84200820 0.17836044 0.58333332 0.20833333
[13] 0.77751433 0.57246404 0.13369234 0.08333335 0.00000000 1.00000000
[19] 0.04166667 0.12246495 0.53759769 0.03688077 0.57690332 0.41578636
[25] 0.75026990 0.42526027 0.63112939 0.07676504 0.98465420 0.02097486
[31] 0.44808668 0.64066857 0.43917715 0.88938042 0.37499991 0.19688800
[37] 0.40500598 0.40134872 0.91059693 0.08887331

$value
[1] -12053607

R> PlotResult(Result)
R> ShowWakePenalizers(Result)
```

4. Conclusion

wflo is an approach to provide a level playing field for researchers in WFLO: It makes the necessary data available at high quality and contains a useful set of functions that takes the necessity to implement a wake model and a specific problem function away from the researcher, enabling her/him to entirely focus on the actual job, which is the optimizer itself. Furthermore, a unified benchmark is provided.

A number of turbines to place of around four is a good start to obtain first experience with **wflo**. Things become however really challenging if, say, 20 turbines are to be placed. Every WFLO researcher is invited to try her/his approach with **wflo** to get to see how competitive the contributed approach really is.

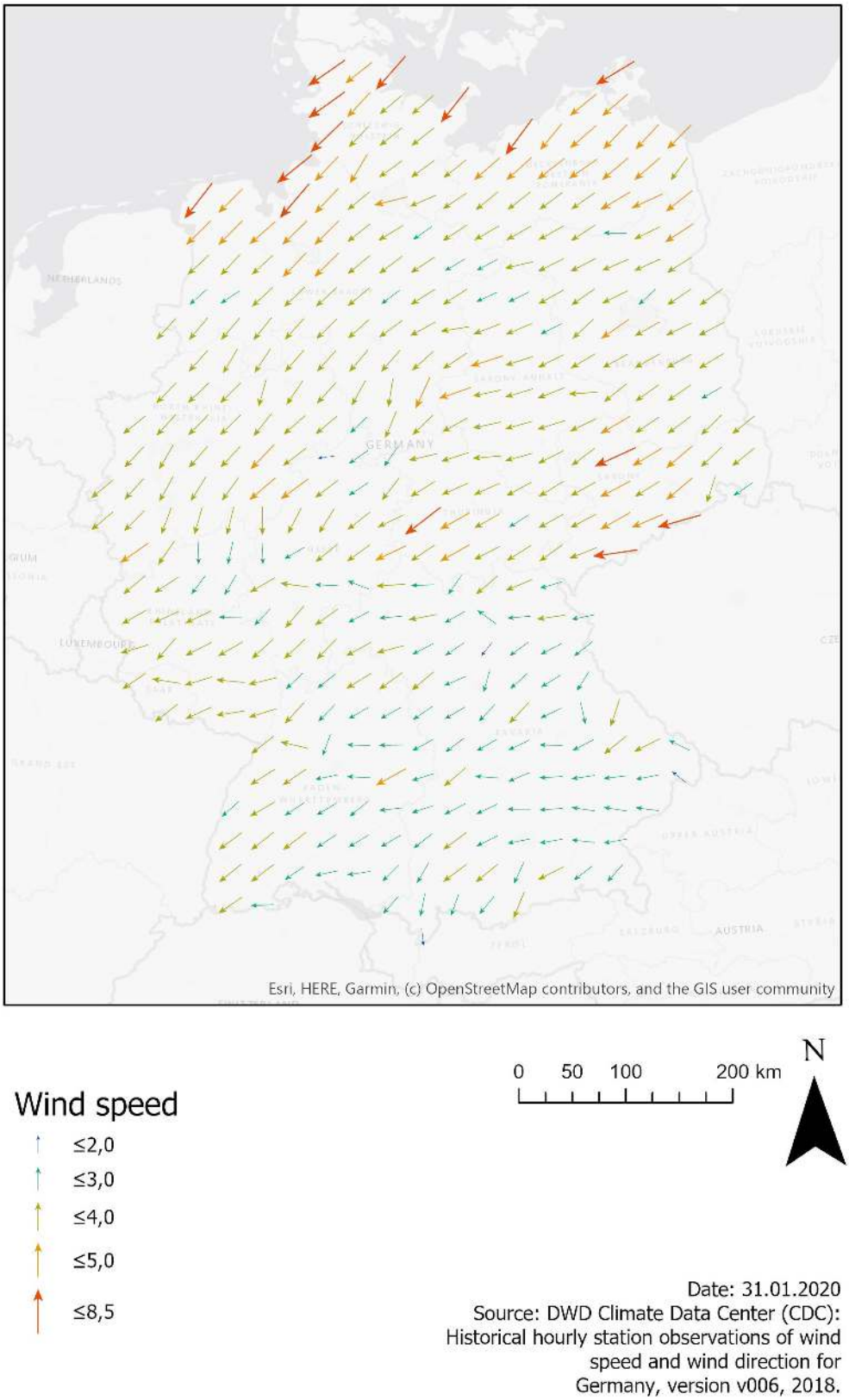

Figure 2: Wind speed and wind direction in Germany.

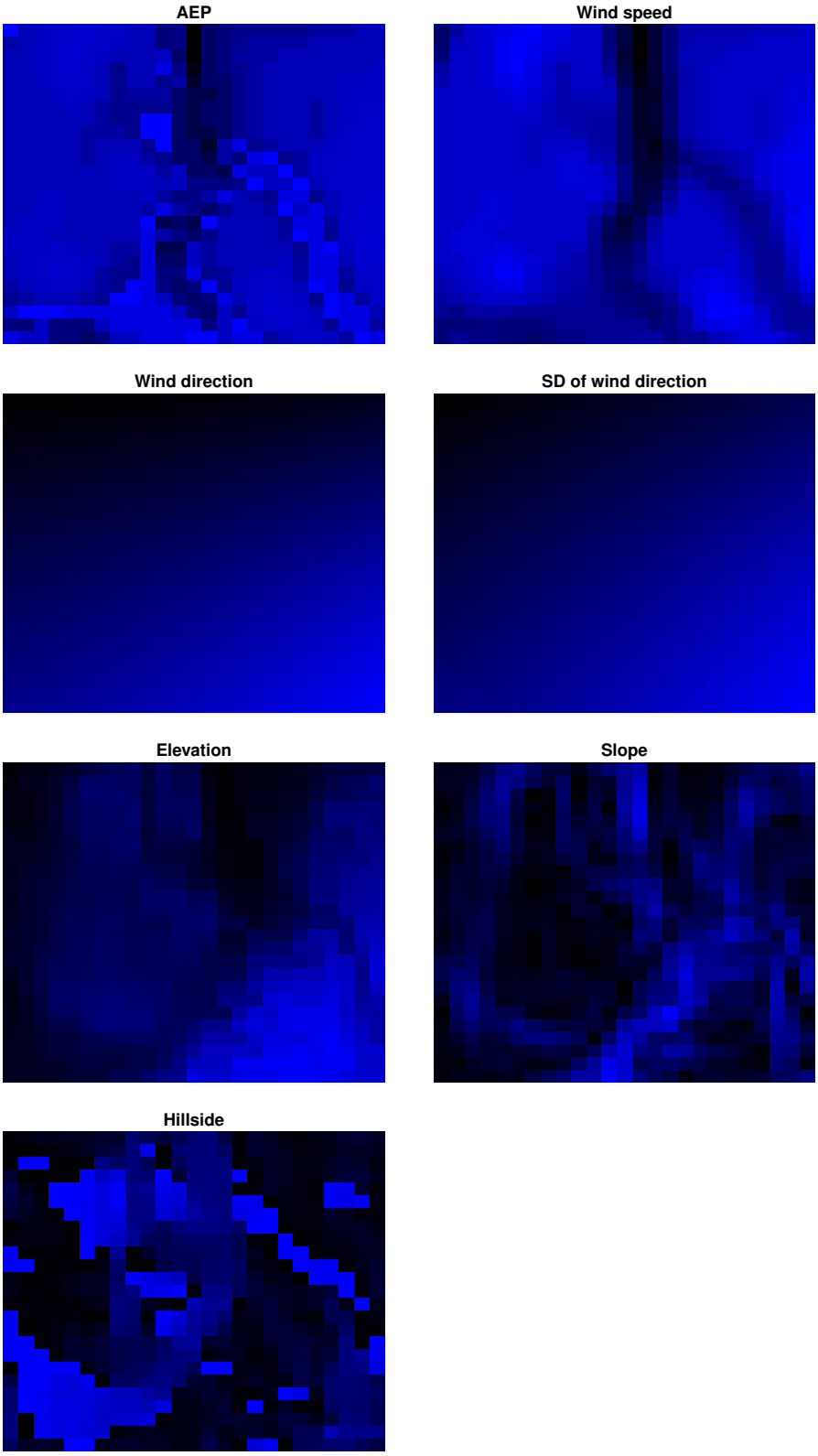

Figure 3: Image plots for the seven slots in FarmData. Lighter shades of blue indicate larger values, darker ones indicate low values.

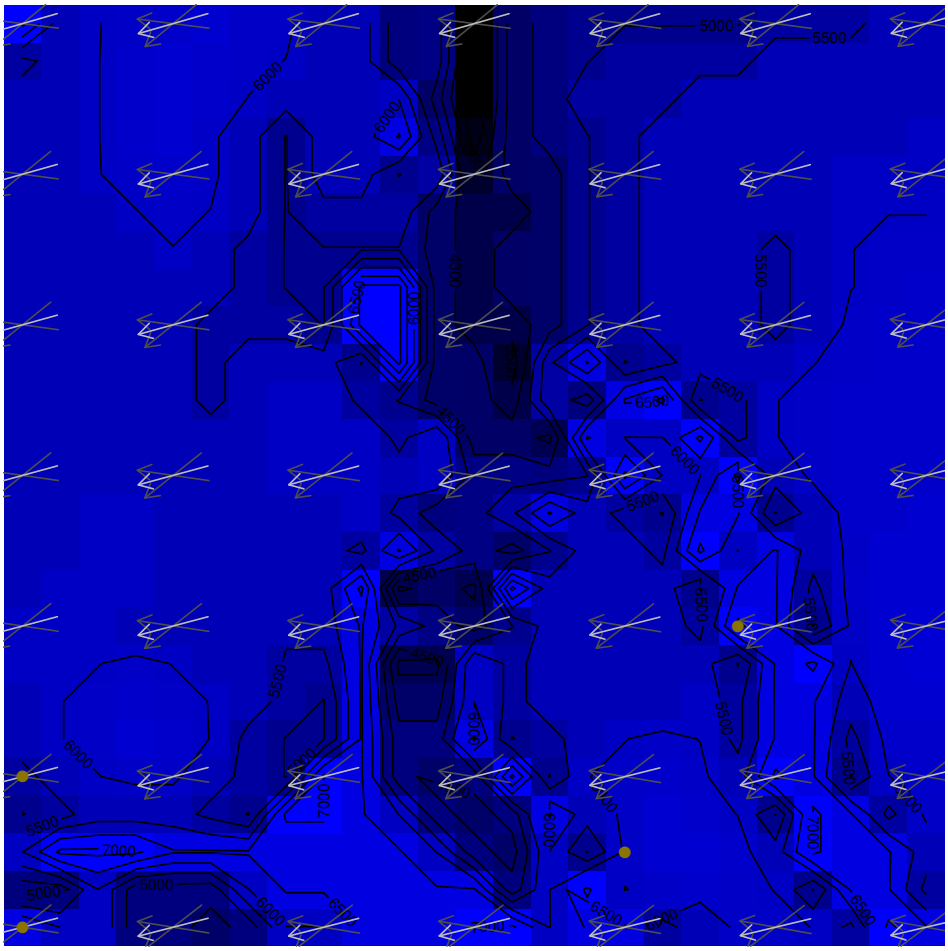

Figure 4: PlotResult() showing the benchmark area, four turbines, L-BFGS-B optimizer.

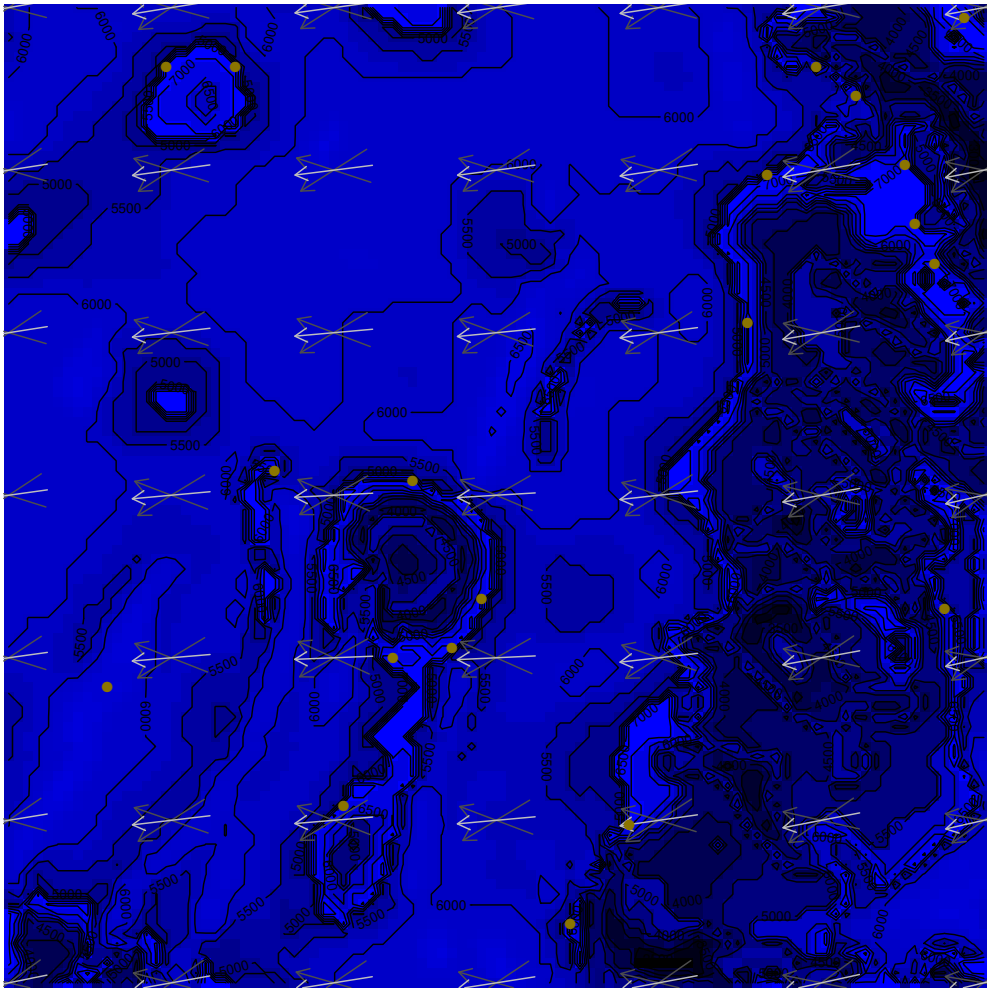

Figure 5: Alternative testing ground, together with a first solution.

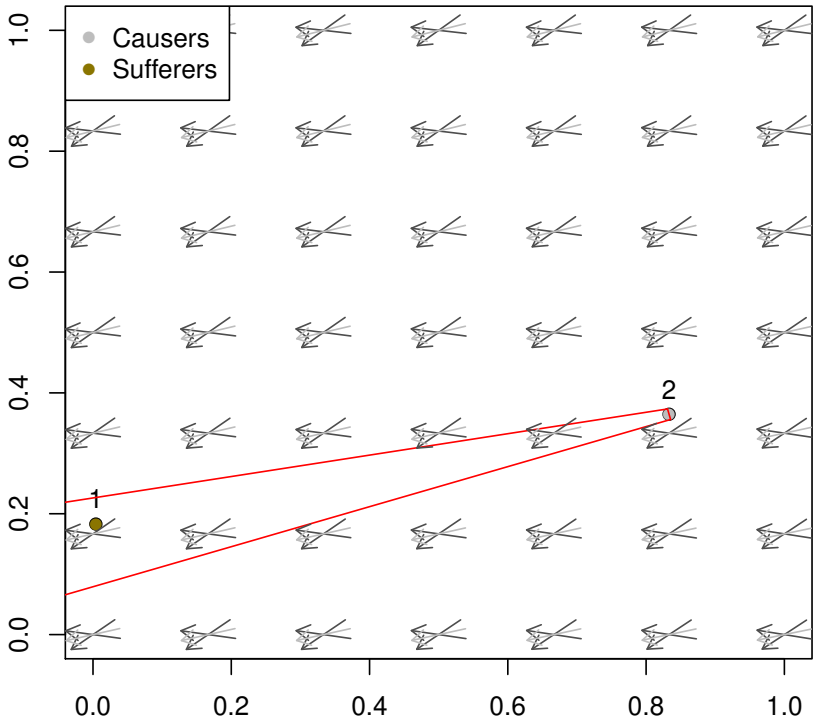

Figure 6: ShowWakePenalizers() showing the wake structure in the benchmark area, four turbines, L-BFGS-B optimizer.

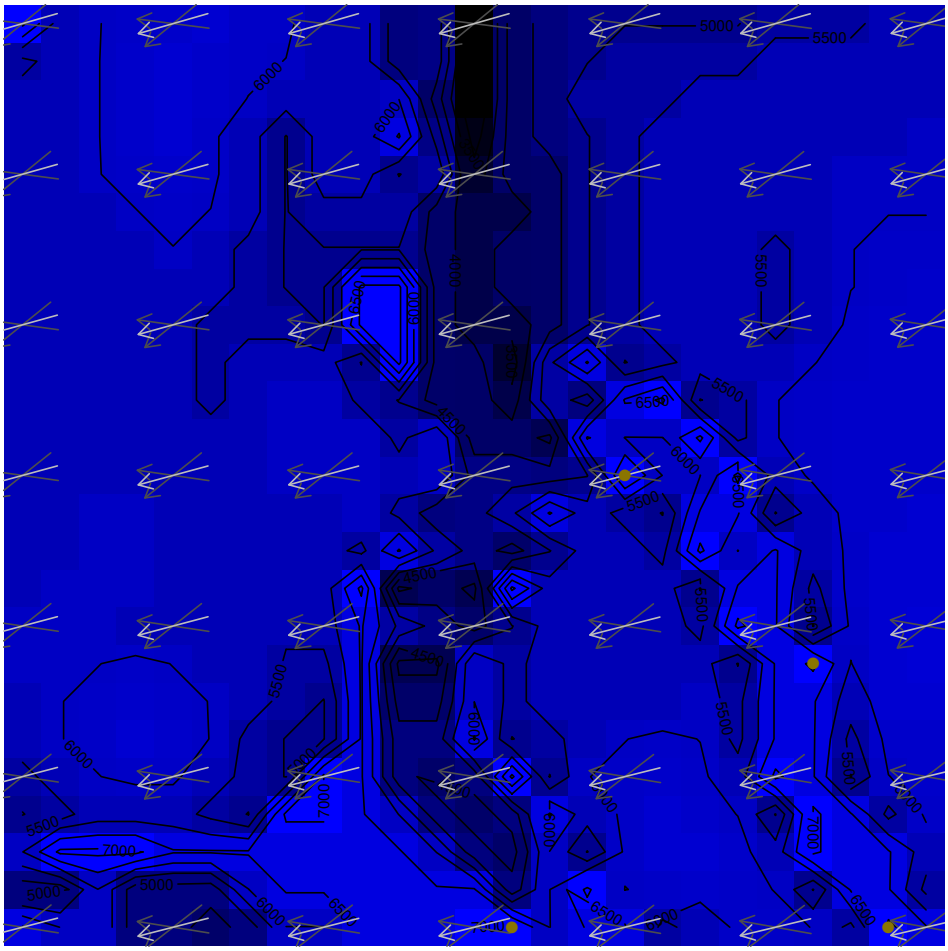

Figure 7: PlotResult() showing the pso optimizer result, four turbines.

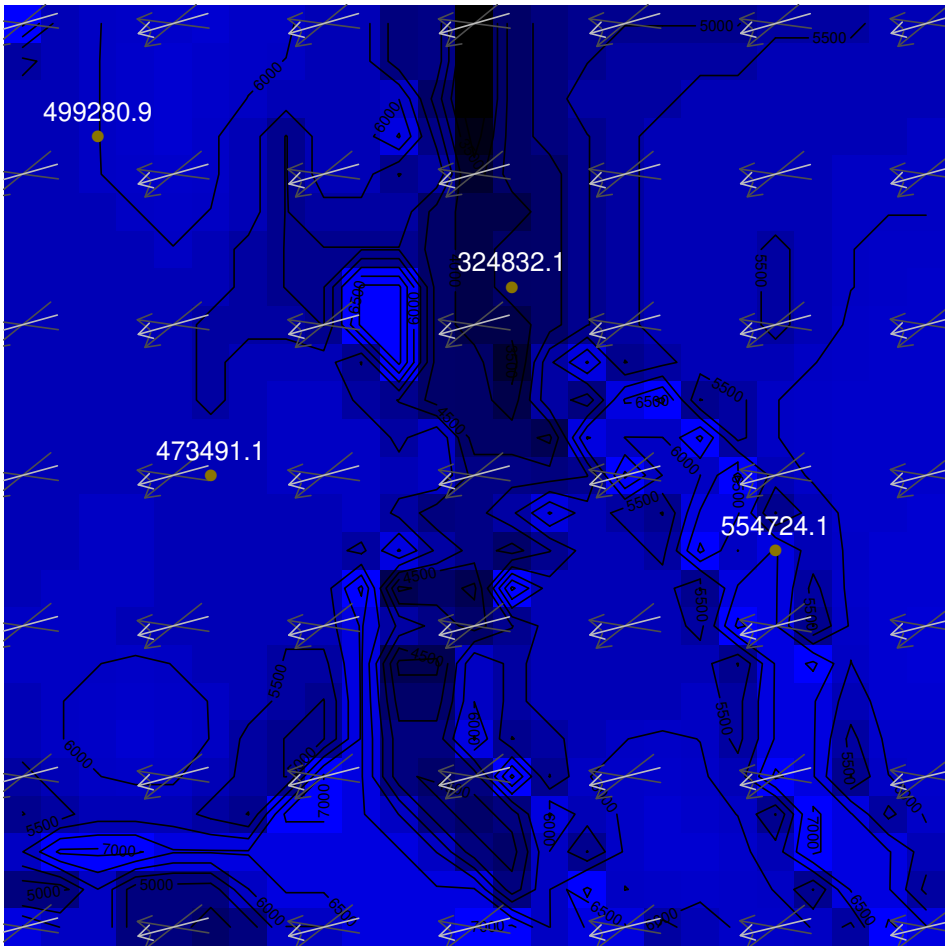

Figure 8: PlotResult() showing an L-BFGS-B result (four turbines), imposing the values returned by ProfitContributors() as labels.

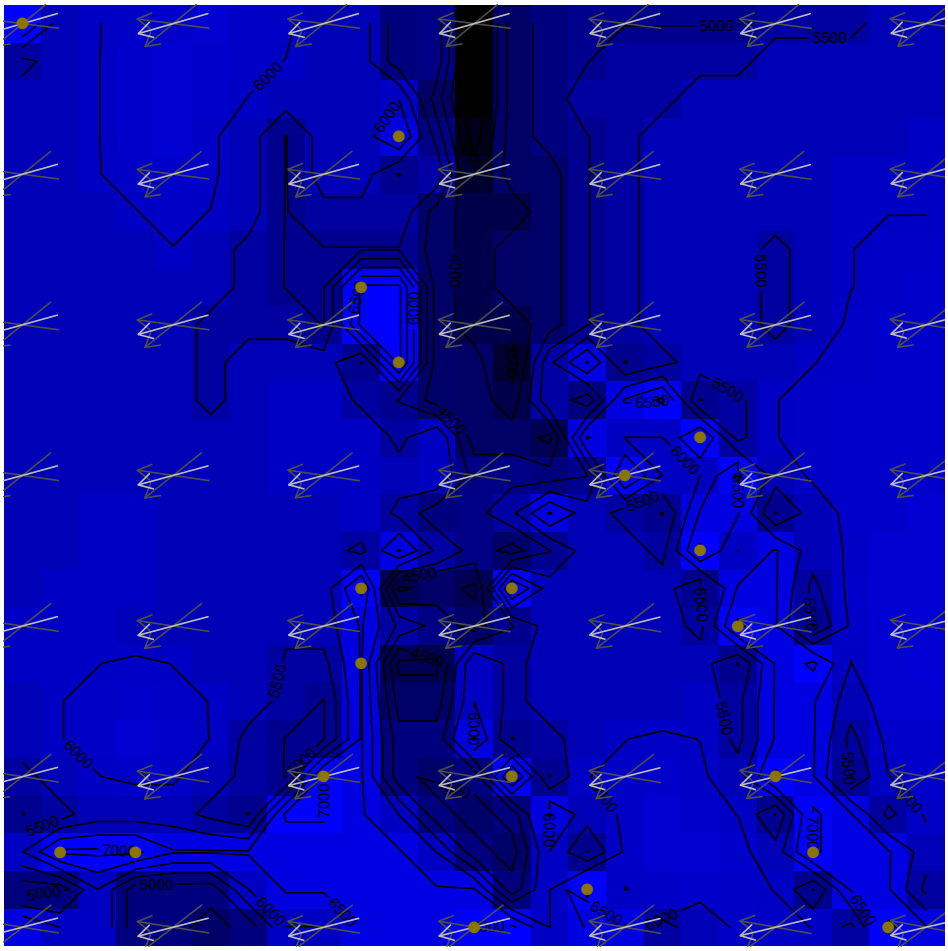

Figure 9: PlotResult() showing the 20 turbines benchmark setup result.

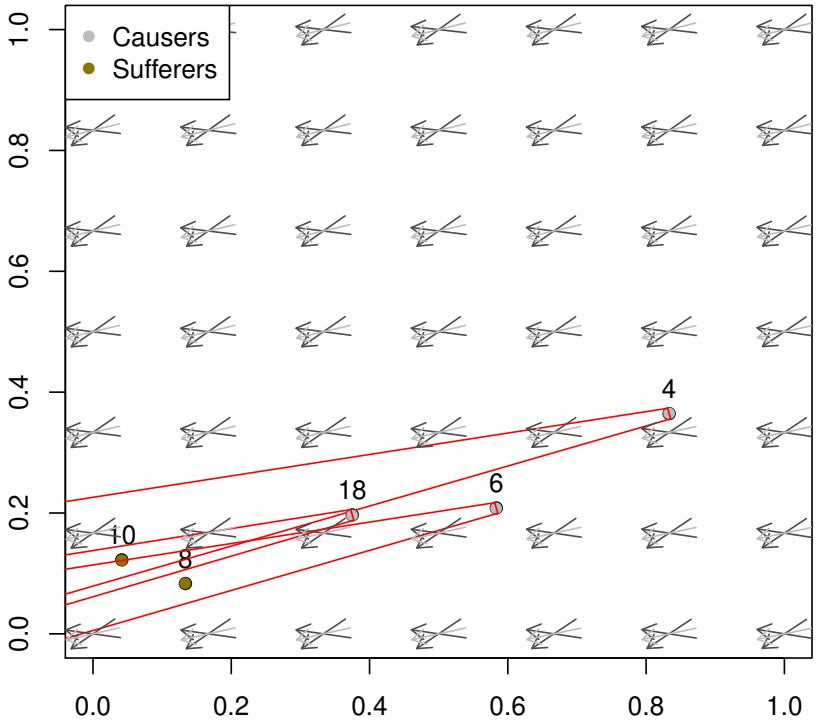

Figure 10: ShowWakePenalizers() showing the 20 turbines benchmark wake structure result.

References

Antonini EG, Romero DA, Amon CH (2020). "Optimal design of wind farms in complex terrains using computational fluid dynamics and adjoint methods." *Applied Energy*, **261**, 1–12.

Baker NF, Stanley AP, Thomas JJ, Ning A, Dykes K (2019). "Best Practices for Wake Model and Optimization Algorithm Selection in Wind Farm Layout Optimization." In *Golden, CO: National Renewable Energy Laboratory*.

Bendtsen C (n.d.). URL https://cran.r-project.org/package=pso.

Chen Y, Li H, Jin K, Song Q (2013). "Wind farm layout optimization using genetic algorithm with different hub height wind turbines." *Energy Conversion and Management*, **70**, 56–65.

EEG (2017). "Erneuerbare-Energien-Gesetz - EEG 2017." URL https://www.gesetze-im-internet.de/eeg_2014/BJNR106610014.html.

Garey MR, Johnson DS (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Giovanni G (2019). "A novel method for wind farm layout optimization based on wind turbine selection." *Energy Conversion and Management*, **193**, 106–123.

Jensen N (1983). "A note on wind generator interaction. Technical Report Riso-M-2411." *Technical report*, Riso National Laboratory.

Johnson SG, Ypma J (n.d.). URL https://nlopt.readthedocs.io/en/latest/.

Kirchner-Bossi N, Porte-Agel F (2018). "Realistic Wind Farm Layout Optimization through Genetic Algorithms Using a Gaussian Wake Model." *energies*, **11**(3268), 3268–.

Mebane W, Sekhon JS (2011). "Genetic Optimization Using Derivatives: The rgenoud package for R." *Journal of Statistical Software*, **42**(11), 1–26.

Park JW, An BS, Lee YS, Jung H, Lee I (2019). "Wind farm layout optimization using genetic algorithm and its application to Daegwallyeong wind farm." *JMST Advances*, **1**, 249–257.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Shakoor R, Hassan MY, Raheem A, Wu YK (2016). "Wake effect modeling: A review of wind farm layout optimization using Jensen's model." *Renewable and Sustainable Energy Reviews*, **58**, 1048–1059.

Thomas JJ, Ning A (2018). "A method for reducing multi-modality in the wind farm layout optimization problem." *IOP Conf. Series: Journal of Physics*, **1037**, 1–11.

Wu Y, Zhang S, Wang R, Wang Y, Feng X (2020). "A design methodology for wind farm layout considering cable routing and economic benefit based on genetic algorithm and GeoSteiner." *Renewable Energy*, **146**, 687–698.

Yang K, Kwak G, Cho K, Huh J (2019). "Wind farm layout optimization for wake effect uniformity." *Energy*, **183**, 983–995.

Affiliation:

Carsten Croonenbroeck
Rostock University
Environmental Science
Justus-von-Liebig-Weg 7
18059 Rostock, Germany
Tel.: +49 381 498 3267
Fax: +49 381 498 3262
E-mail: carsten.croonenbroeck@uni-rostock.de

David Hennecke
Rostock University
Geodesy and Geoinformatics
Justus-von-Liebig-Weg 6
18059 Rostock, Germany
Tel.: +49 381 498 3204
E-mail: david.hennecke@uni-rostock.de