Informed Mutation of Wind Farm Layouts to Maximise Energy Harvest

Michael Mayo, Maisa Daoud

Department of Computer Science
Faculty of Computing, Mathematics and Statistics
University of Waikato
Hamilton, New Zealand

Abstract

Correct placement of turbines in a wind farm is a critical issue in wind farm design optimisation. While traditional "trial and error"-based approaches suffice for small layouts, automated approaches are required for larger wind farms with turbines numbering in the hundreds. In this paper we propose an evolutionary strategy with a novel mutation operator for identifying wind farm layouts that minimise expected velocity deficit due to wake effects. The mutation operator is based on constructing a predictive model of velocity deficits across a layout so that mutations are inherently biased towards better layouts. This makes the operator informed rather than randomised. We perform a comprehensive evaluation of our approach on five challenging simulated scenarios using a simulation approach acceptable to industry [1]. We then compare our algorithm against two baseline approaches including the Turbine Displacement Algorithm [2]. Our results indicate that our informed mutation approach works effectively, with our approach identifying layouts with the lowest aggregate velocity deficits on all five test scenarios.

Keywords: wind farm, layout optimisation, velocity deficit, wake effect, evolutionary strategy, informed mutation operator, turbine displacement algorithm

1. Introduction

Effective optimisation of large wind farms layouts is a significant open research problem for two primary reasons.

The first reason is that solving this problem well is relevant to the global economy. Worldwide, the wind power industry is rapidly expanding, and the Global Wind Energy Council predicts that wind energy production could reach as much as 2,000 gigawatts (GW) globally by 2030 [3]. This would account for approximately 18% of the world's energy production [3], and cost reductions in the production of this renewable energy are therefore critical.

As the demand for wind energy increases, so too must the size of the wind farms. For example, the London Array [4], commissioned in 2013, generates 630 megawatts (MW) of power and comprises 175 offshore turbines. This generates enough power to service 490,000 households. In the US, the Alta Wind Energy Plant [5] consists of 600 turbines generating power equivalent to the usage of 257,000 households. Both of these are dwarfed by the Gansu project in China [6], which is planned to generate 20GW by 2020, and is being constructed from smaller 100-200MW farms with an estimated 36 turbines being added to the farm per day. Clearly, even small efficiencies at any of the stages in wind farm design have the potential to translate into significant gains.

The particular cost saving avenue we focus on in this paper is that of arranging the turbines in a farm to minimise *wake effects* [7, 8]. Wake effects occur when one wind turbine is placed downstream of either another turbine or an obstacle such as a building. Wakes are characterised by decreased air stream velocity along with higher turbulence and vorticity compared to the surrounding unaffected air stream. Wake effects typically are a cause of power losses due to the reduced velocity of the wind [8]. They also lead to increased maintenance costs due to the increased turbulence, especially so when a turbine is partially inside a wake and partially outside [8]. Increased noise is also a consequence of the wake effect [8].

Proper turbine placement inside a wind farm to minimise wake effects, therefore, is a pressing problem.

The second primary reason why the wind farm layout optimisation problem is interesting for research is from the perspective of computational intelligence. The problem itself is challenging because there is usually no means of solving layout problems analytically, and the various objective functions that are used are highly non-linear, discontinuous due to layout constraints, and multimodal. Therefore, the most frequent way of solving this problem is to approximate a solution using a metaheuristic search algorithm such as a genetic algorithm (e.g. [9]) or local search (e.g. [2]).

Characteristics of the problem further add to the computational challenge, and those are the high dimensionality of layouts (for example, a 500-turbine layout in which the turbines are homogenous and specified completely by a two-dimensional position amounts to a thousand dimensional optimisation problem), and the time complexity of the evaluation function, which is at least quadratic in the number of turbines depending on the particular method used. For large layouts, this means that effectively, only a small fraction of the search space can be explored in a reasonable amount of time.

In this paper, we propose and evaluate a new algorithm for solving the wind farm layout optimisation problem. The algorithm is inspired by the idea of searching using an evolutionary algorithm (EA) that has an *informed mutation operator* [10], in comparison to a typical evolutionary approach that uses an uninformed or randomised operator. In theory, informed operators have a higher probability of making improvements whereas uninformed operators have no such bias. The former should therefore help an EA reach a better quality solution more readily than the latter.

The cost of using an informed operator, however, is that it is more complex than an uninformed operator, and this typically makes the operator problem-specific. In other words, the informed operator can only be used for solving the wind farm layout optimisation problem. In this research, we use machine learning as a basis for making our mutation operator informed.

Previously, we have already conducted a preliminary investigation of this approach vs. an identical approach that uses an uninformed mutation operator [11]. The results were positive when evaluated on a set of benchmark problems, and therefore in the current paper we continue our investigation by providing (i) a modified version of our algorithm that has been further enhanced and improved, and (ii) a more extensive evaluation of our approach, this time comparing to the current state-of-the-art algorithm, namely the turbine displacement algorithm (TDA) [2].

2. Background

In this section we describe the wind farm layout optimisation problem itself. We then discuss the wind farm layout evaluation method used in this research, and then the current state-of-the-art layout optimisation algorithm from the literature, TDA [2], is described.

2.1. The Wind Farm Layout Optimisation Problem

A *wind farm* is defined as a collection of possibly heterogenous wind turbines that are located in the same approximate area and are used to harvest kinetic energy from the wind. Wind farms may be on-shore or off-shore. If on-shore, then they may be located on terrain that is either flat or rugged. In the latter case, modelling the wind farm is more difficult, and therefore many current approaches make the assumption of near-smooth terrain so that turbine positions can be specified solely by two dimensional coordinates.

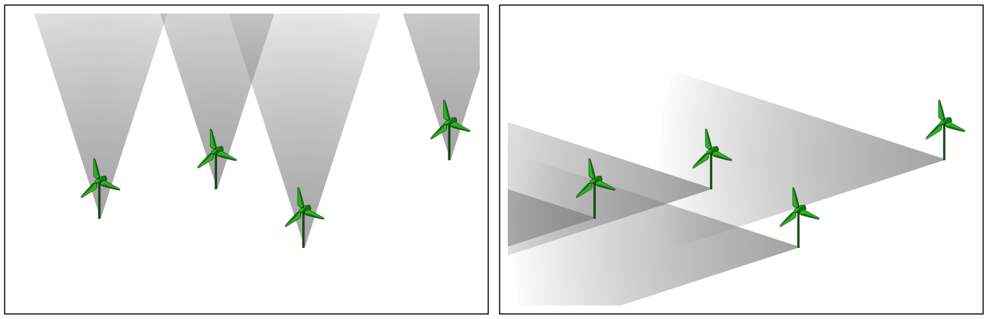
A wind farm typically constrains the positions of its turbines within its layout regions. There are various reasons for this. The two main ones are firstly the presence of obstacles (e.g. roads and buildings) on the layout where turbines cannot be placed, and secondly the fact that two turbines cannot be positioned too closely together due to safety concerns. This minimum distance constraint arises because the immediate wake of a wind turbine is extremely turbulent, and therefore turbines placed too closely together may damage each other. A separation between turbines of eight times the turbine's rotor radius is therefore recommended [1].

Despite minimum distance constraints, turbines still interact with each other (albeit less strongly), and it is this interaction that leads to the optimisation problem. The primary means by which two or more turbines interact is called the wake effect, which was discussed in the Introduction.

To explain the wake effect, it is easiest to envisage a single turbine placed such that its rotor blades are perpendicular to the current wind direction. Such a turbine is unhindered in its ability to harvest the kinetic energy of the wind. It should be able to harvest 100% of the potential energy that it could harvest: we therefore say that its *expected velocity deficit* is 0.0, or conversely, its *expected wake free ratio* – which amounts to 1.0 minus the expected velocity deficit – is 1.0.

Now imagine a second turbine directly behind the first turbine: the second turbine experiences the velocity deficit caused by the first turbine. This results in the second turbine being unable to harvest the same amount of kinetic energy as the first turbine – in fact, the second turbine will only be able to extract some fraction, for example 80%, of the energy that the first turbine harvests. This situation corresponds to the second turbine having a velocity deficit of 0.2.

The wake that a turbine generates is a spreading cone of gradually decreasing velocity deficit. The cone's apex corresponds to the turbine's po-



(a) Wind blowing north (0°) (b) Wind blowing west (270°)

Figure 1: The same four-turbine layout showing turbine positions and turbine wake interferences for two different wind directions. Wakes are depicted as cones. Darker areas of the layout indicate regions of increasing velocity deficit; white areas indicate areas of no velocity deficit.

sition, and the rate of velocity deficit decreases with distance depending on several factors including the angle made between the turbine's rotor blades and the wind direction, the diameter of the rotor blades, the wind speed, and the terrain roughness [1].

If a turbine lies in the wake of more than one other turbines, then the velocity deficits aggregate [1]. This may result in some turbines having a very high velocity deficit compared to others.

The calculation is also complicated by the fact that turbines will experience different expected velocity deficits for each different predominant wind direction. Figure 1 illustrates this. In the figure, the same small layout is depicted twice, the versions differing only in wind direction. Clearly, when wind is blowing north (Figure 1(a)), there are no velocity deficits between turbines; but when the wind direction changes (Figure 1(b)), two of the turbines experience velocity deficits, and one of the turbines lies in the wake of not one but two other turbines.

It is evident, then, that the total power output of a wind farm depends heavily on the expected velocity deficits of the individual turbines that make up the farm. These in turn are functions of the turbines' relative and absolute positions on the farm along with the predominant wind speeds and directions. Therefore different positions for the turbines will lead to different power outputs, and the optimisation problem is one of finding the configuration that maximises total power output.

2.2. Simulation of Wind Farms

The wind farm model we utilise in this research is the approach presented by Kusiak and Song [1], which was re-used in both Wilson et al. [12, 13] and the 2014 and 2015 Wind Farm Layout Optimisation competitions [14].

The model makes several simplifying assumptions about terrain roughness (i.e. terrain is assumed relatively smooth), turbine homogeneity (all the turbines are identical), wind speed distributions (wind speeds follow a Weibull distribution), and the variation of wind speed with height. Despite these simplifications, the model has the advantage of being "acceptable for industrial application" [1] and is therefore ideal for research purposes as well.

The time complexity of this model is $\mathcal{O}(n^2d)$ where n is the number of turbines in the layout and d is the number of wind directions considered. The n^2 term arises because wake effects between every single pair of turbines must be calculated individually, which is quadratic in the number of turbines. The constant factor d specifies the fidelity of the simulation. For example, if wind data is discretised into 15° segments, then $d = \frac{360^\circ}{15^\circ} = 24$. If a finer grained simulation is required (and the corresponding finer grained wind data is available) then d may be much higher.

Once the expected velocity deficits of the individual turbines are calculated, the overall sum or average expected velocity deficits across the entire farm can be easily computed and used as a measure of the value or fitness of the layout. We adopt this recommended approach in this paper.

2.3. The Turbine Displacement Algorithm

The current state-of-the-art algorithm in the literature for optimising a wind farm layout is the turbine displacement algorithm (TDA) proposed by Wagner [2]. In essence, TDA is a very simple local search algorithm that moves one random turbine at a time before evaluating the modified layout. If the modified layout is at least as good as the original layout, then the algorithm keeps the modified layout and discards the original. In this way, beneficial modifications accumulate and the layout is gradually optimised.

The choice of moving one turbine at a time was made chiefly because of the $\mathcal{O}(n^2)$ time complexity of the Kusiak & Song evaluation function [1] used in the original TDA publication [2]. The quadratic time complexity can be substantially mitigated using a neat algorithmic "speedup" strategy if only one turbine moves between evaluations. However, it turns out that this constraint is also useful for more than mitigating time complexity: TDA

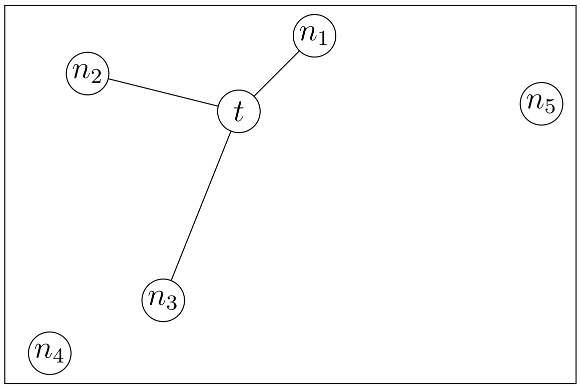


Figure 2: Example of turbine t's neighbourhood where $K = 3$. The three closest turbines to t are n_1, n_2 and n_3 and are included in the neighbourhood. Other turbines that are further away are excluded.

is also a highly effective algorithm when compared to other approaches, even if the algorithmic speedup is not employed.

In fact, in two recent extensive evaluations, both Wagner [2] and Wilson et al. [13] found that TDA outperformed all other approaches including genetic algorithms, particle swarm optimisation, and developmental models in terms of finding layouts with the highest expected velocity deficit per turbine, across several different wind and obstacle scenarios.

The interestingness of TDA lies in its heuristic for shifting each turbine. Because wake effects are reduced with distance, the algorithm makes the simplifying assumption that only the K nearest neighbouring turbines to the given turbine are important. The K nearest neighbourhood is illustrated in Figure 2. A *displacement vector* is then calculated, which is a vector pointing in a direction *away* from the K nearest neighbouring turbines. The rationale for this is that moving the turbine away from its neighbours is more likely to decrease its velocity deficit. The displacement vector is then perturbed with angular noise (the magnitude of the noise being determined by a parameter σ_{dir}), optionally flipped in direction with probability p, and then added to the current turbine's position to get its new position. If the turbine's new position is invalid (e.g. outside of the layout, or colliding with an obstacle), then the displacement vector is gradually reduced in magnitude until the new position becomes valid.

We note that inverting the displacement vector with probability p actually brings the turbine closer to its K nearest neighbours. Wagner's justification for this is that sometimes closer groups of turbines actually increase a farm's overall power output [2]. The value of p should typically be set to a small value.

We also note that the TDA algorithm as originally published has the potential for a divide-by-zero exception in the displacement vector calculation (see lines 7-8 of Algorithm 2 in [2]). In our implementation of TDA we therefore assign the displacement vector to a random unit vector if a divide-by-zero error occurs.

Finally, we note that TDA algorithm is not a "random walk" through the space of possible layouts. This is because the probability of mutating layout A to get layout B is *not* the same as the probability of mutating B to get A. This is because each turbine in the layout, besides its position, also has an associated magnitude that determines the initial size of the displacement vector. If shifting a turbine results in a global improvement to the layout, then the turbine's magnitude increases by a small amount; conversely, it decreases. Thus it is the sequence of accepted previous mutations that determine the probability distributions over next states when TDA's mutation operator is applied. The initial value of each turbine's magnitude is determined by the parameter σ_{dist_init}.

3. An Evolutionary Strategy with Informed Mutation

We now describe our new approach to optimising wind farm layouts.

3.1. Local Neighbourhood Definition

The approach presented in this research builds on the notion of a turbine's K nearest neighbours being important. We also follow the same basic pattern of the TDA approach in that one turbine is moved at a time, and the layout is evaluated after every move.

However, rather than using TDA's heuristic approach of computing a displacement vector and adding it to the turbine's current position, we instead use machine learning to construct a predictive model of velocity deficits for all the turbine neighbourhoods in a layout. We then attempt to shift the current turbine to the best possible location on the layout (i.e. the location with lowest predicted velocity deficit), as predicted by our model.

To explain in more detail, we must further refine the notion of what constitutes a neighbourhood of size K. Let us consider as an example a single turbine t and, supposing $K = 3$ as in Figure 2, its neighbouring turbines n_1, n_2 and n_3. Our definition of t's neighbourhood is the following:

- the absolute (x, y) position of t on the layout, and
- the relative locations of t's neighbours n_1, n_2 and n_3 with respect to t, sorted in ascending order of distance from t.

Differentiating between absolute and relative location information is important in our view because absolute information (namely t's position globally on the layout) has an impact on velocity deficit. For example, if t is positioned on the edge of the layout that is facing the predominant wind direction, then it is likely to have a lower velocity deficit than if it were on the opposite side of the layout. Similarly, relative information is important because it is the relative configuration of neighbouring turbines that produces the majority of the velocity deficit that a turbine experiences. We therefore include both types of location information in our neighbourhood definition.

Sorting the neighbours by distance from t is also important because this ensures that neighbourhoods can be compared sensibly for similarity or dissimilarity. If the sorting step were excluded from the algorithm then it is not possible to directly compare neighbourhood configurations because the ordering of the neighbouring turbines would be arbitrary, and thus the model would be degraded.

3.2. Predictive Model Building Algorithm

Once the neighbourhood representation is determined, the next step in our proposed approach builds a predictive model of velocity deficits across the layout. In essence, this is achieved by first of all evaluating the layout so that the velocity deficits for each turbine are available. The velocity deficits are then converted into wake free ratios by subtracting the deficit from one, and these wake free ratios will be used as regression targets for the predictive model. The conversion from velocity deficit to wake free ratio is a convenience that converts the optimisation problem from one of minimisation to one of maximisation.

Next we calculate the neighbourhood configurations (i.e. the absolute and relative locations discussed above) for each and every turbine in the layout, and label each configuration with the central turbine's wake free ratio. Once this is achieved, we can build the predictive model.

```
Input: turbine positions $T = \{(x_1, y_1), (x_2, y_2), \dots\}$, wake free ratios
       $W = \{w_1, w_2, \dots\}$, neighbourhood size $K$
begin
    /* start with an empty dataset with dimensionality that
       is a function of K                                    */
    $D \leftarrow create\_empty\_dataset(K)$;
    /* iterate over every turbine in the layout              */
    foreach turbine position $(x_i, y_i) \in T$ do
        /* get the K nearest neighbours of the current
           turbine                                           */
        $knn \leftarrow k\_nearest\_neighbours(T, K, (x_i, y_i))$;
        /* calculate the angle and distance of each
           neighbour from the current turbine                */
        foreach neighbour $(x_j, y_j) \in knn$ do
            $d_j \leftarrow distance((x_i, y_i), (x_j, y_j))$;
            $\theta_j \leftarrow angle((x_i, y_i), (x_j, y_j))$;
        end
        /* sort the neighbours into ascending order of
           distance and then add the neighbourhood
           configuration to D                                */
        $sort\_by\_distance(knn)$;
        $ex \leftarrow create\_example(x_i, y_i, d_1, \theta_1, \dots, d_K, \theta_K, w_i)$;
        $add\_example(D, ex)$;
    end
    /* learn the model given the labelled dataset            */
    $P \leftarrow build\_model(D)$;
    /* done -- return the newly built model                  */
    return $P$
end
```

Algorithm 1: Model building algorithm. It is assumed that each turbine has an associated wake free ratio, i.e. $|T| = |W|$.

More formally, Algorithm 1 shows pseudocode used to construct the model. In our approach, we use polar coordinates (i.e. an angle and a distance) to encode relative location information. This makes the sorting step of the algorithm easier because the distances are explicit and do not need to be calculated.

The algorithm is also not specific about the particular predictive model used. Essentially, any predictive model capable of regression is appropriate.

```
Input: turbine positions $T = \{(x_1, y_1), (x_2, y_2), \dots\}$, wake free ratios
       $W = \{w_1, w_2, \dots\}$, number of samples $N$, predictive model $P$
begin
    /* select the worst turbine out of the entire layout */
    $i \leftarrow index\_of\_turbine\_with\_lowest\_wfr(W)$;
    /* randomly select the first sample */
    $(x_{best}, y_{best}) \leftarrow random\_valid\_location()$;
    $w_{best} \leftarrow predict\_wfr(T, P, (x_{best}, y_{best}))$;
    /* select N-1 more samples (note that this loop will
       not execute if N=1) */
    for $j = 2 \dots N$ do
        $(x, y) \leftarrow random\_valid\_location()$;
        $w \leftarrow predict\_wfr(T, P, (x, y))$;
        /* always keep the best sample */
        if $w > w_{best}$ then
            $(x_{best}, y_{best}) \leftarrow (x, y)$;
            $w_{best} \leftarrow w$;
        end
    end
    /* shift the worst turbine to the best predicted point
       from amongst the samples */
    $T \leftarrow move\_turbine(T, i, (x_{best}, y_{best}))$;
    /* return the updated list of turbine positions */
    return $T$
end
```

Algorithm 2: Informed Mutation Operator algorithm.

3.3. Informed Mutation Operator

We now turn to a description of our informed mutation operator. The mutation operator proposed here takes two of the same inputs as used by the previous algorithm, namely the (x, y) positions of the turbines on the layout as well as the wake free ratios of those individual turbines. It also takes two additional parameters: the model P which was built by applying Algorithm 1, as well as a new parameter N that specifies how many randomly selected locations will be evaluated by the model.

Essentially, the mutation operator first of all selects the turbine with the lowest individual wake free ratio which it decides to shift. It then samples N random valid locations on the layout. The wake free ratio at each of the N locations is predicted by first of all "pretending" that the turbine under consideration is to be shifted to the sampled position, and then using the model to predict what the turbine's wake free ratio would be at that point. The sampled location with the highest prediction is the one that the turbine is actually shifted to.

This process is depicted in Algorithm 2. Clearly, when $N = 1$, there is no influence of the model on position selection and therefore the mutation operator amounts to a randomised operator. However, when $N > 1$, the model does have some influence, and with higher values of N, the influence is greater. On the surface, it may seem that extremely high values of N would be beneficial because there is a much greater chance that locations with high predicted wake free ratio can be discovered. However, in practice (as our evaluations later show), higher values of N may also mislead the search if the model is inaccurate. We therefore prefer modest values for N such as 10, 100, or at most, 1000.

3.4. Final Evolutionary Strategy

The final algorithm (depicted as Algorithm 3) that we are presenting in this paper is now described. Basically, we propose a 1+1 Evolutionary Strategy (ES) [15] with a stopping criteria determined by a maximum number of evaluations MAX_EVALS. Inside the main loop of the algorithm, there is first of all a check to determine if the predictive model should be either built for the first time or rebuilt. We included the periodic model rebuilding because if the model is built only once, it may quickly go "out of date," as the algorithm proceeds to better layouts well beyond its initial one in terms of quality.

```
Input: neighbourhood size $K$, number of samples $N$, maximum
       number of evaluations $MAX\_EVALS$, model rebuild interval
       $MRI$
begin
    /* initialise the evolutionary strategy by creating a
       random initial layout                                    */
    $best \leftarrow create\_initial\_layout()$;
    $best\_val \leftarrow evaluate(best)$;
    $num\_evals \leftarrow 1$;
    /* begin the evolutionary strategy's main iteration         */
    repeat
        /* check to see if the model P needs to be built
           using Algorithm 1                                    */
        if $(num\_evals - 1)\%MRI == 0$ then
            $T \leftarrow get\_turbine\_positions(best)$
            $W \leftarrow get\_wake\_free\_ratios(best)$
            $P \leftarrow invoke\_algorithm1(T, W, K)$;
        end
        /* copy the best solution and then mutate it using
           Algorithm 2                                          */
        $candidate \leftarrow copy(best)$;
        $T \leftarrow get\_turbine\_positions(candidate)$
        $W \leftarrow get\_wake\_free\_ratios(best)$
        $T \leftarrow invoke\_algorithm2(T, W, N, P)$;
        $set\_turbine\_positions(candidate, T)$;
        /* evaluate the candidate solution and keep it if it
           is better                                            */
        $candidate\_val \leftarrow evaluate(candidate)$;
        $num\_evals \leftarrow num\_evals + 1$;
        if $candidate\_val \geq best\_val$ then
            $best \leftarrow candidate$;
            $best\_val \leftarrow candidate\_val$;
        end
    until $num\_evals \geq MAX\_EVALS$;
    /* done -- return best layout found                         */
    return $best$
end
```

Algorithm 3: Final Evolutionary Strategy.

A parameter MRI is used to govern the frequency with which the model should be rebuilt. As the algorithm indicates, if $MRI = 1$ then the model will be built every iteration, but if $MRI \geq MAX\_EVALS$, it will be built only once. Any value of MRI between these extremes is a compromise and represents a trade off between recency of the model and model building overhead.

The next step in the main loop of the algorithm is to mutate the copy of the current best layout. This is performed using Algorithm 2 which has already been described.

Finally, the candidate copy is evaluated and compared to the best layout. If its overall expected wake free ratio is higher or the same, then the candidate is retained as the new best layout.

3.5. Other Considerations

Although we have presented our approach as a 1+1 ES, it is by no means limited to this. It is straightforwardly possible to generalise Algorithm 3 to a $\lambda + \mu$ ES, or even a genetic algorithm, but we leave this to future work.

Finally we will point out that the algorithm used in this study differs from the one previously published [11] during our preliminary investigation of this approach. The main changes are primarily that the current version of the algorithm selects the worst turbine to mutate on each iteration; previously it was a random turbine, which was less effective. Furthermore, in order to make the algorithm more comparable to TDA, this version of the algorithm shifts only one turbine at a time. Previously, a percentage of turbines were moved per iteration – which in turn meant that the effects of mutation were partially dependent on the layout size (i.e. larger layouts resulted in more turbines moving per iteration and vice versa). By changing the algorithm to shift only one turbine per iteration, this drawback is ameliorated.

4. Evaluation

In this section, we describe the scenarios and implementation-specific settings used to evaluate our informed mutation operator-based ES, and then compare our approach with the current state-of-the-art algorithm TDA.

4.1. Scenarios

The test scenarios utilised are those used in the 2014 Wind Farm Layout Optimisation competition [14]. There are five diverse and challenging layout

Table 1: The layout dimensions and number of turbines for each scenario.

Scenario	Width (m)	Height (m)	# Turbines
1	3,500	16,100	220
2	4,000	9,900	150
3	15,800	11,300	710
4	10,500	7,400	300
5	15,900	14,500	910

problems in the evaluation set, and while they are all layouts with a rectangular boundary, they also all contain obstacles of different shapes and sizes. The number of turbines that must be optimised is fixed for each scenario, but varies considerably between 150 and 910 turbines. The exact dimensions and number of turbines per scenario is given in Table 1.

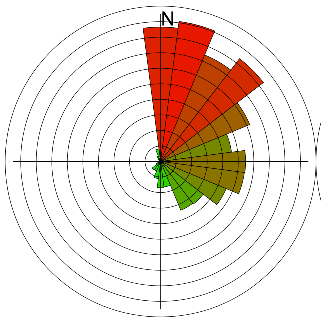
Each scenario also has its own unique wind speed/direction profiles. The wind speed data is discretised into 15° bins, and is depicted in Figure 3 using wind roses. Note that in our wind roses, each reading on the wind roses gives an expected wind speed along one discretised wind direction. Expected wind speed is defined as the average speed observed when wind blows in one particular direction, multiplied by the probability of the wind blowing in that direction. The scale of the wind rose is then adjusted to fit the expected wind speeds. In contrast, typical wind roses show show the probability of wind blowing in a direction and its speed separately – which potentially results in a more difficult plot to read.

Finally, each scenario also has its own unique obstacles, and these vary from a single large rectangular obstacle to multiple smaller obstacles, or a mixture of larger and smaller obstacles. The exact obstacles are best described visually, and are depicted in Figure 4.

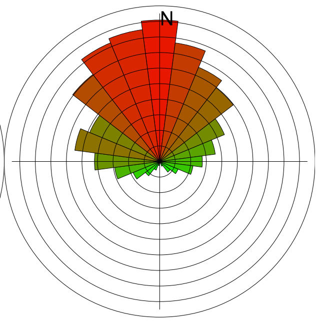
4.2. Experimental Set-up

The evaluation we performed consisted of comparing TDA to our proposed new approach. To make the comparison, we implemented TDA and set its parameters to the same values as used in the original paper describing TDA [2] where possible. We also fixed the maximum number of evaluations to the same for both algorithms, and set the model rebuild interval for the ES to a constant. These fixed parameters are shown in Table 2.

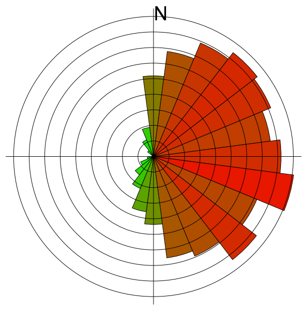
We note that the number of evaluations performed is fixed for all algorithms to a constant 1,000. This makes the comparison fair, but it does mean



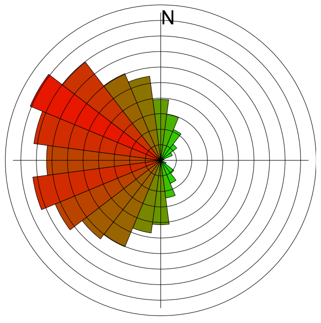
(a) Scenario 1



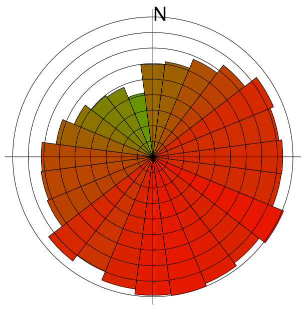
(b) Scenario 2



(c) Scenario 3



(d) Scenario 4

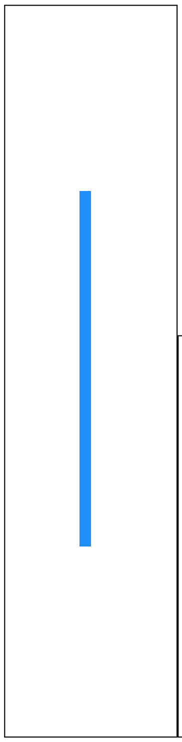


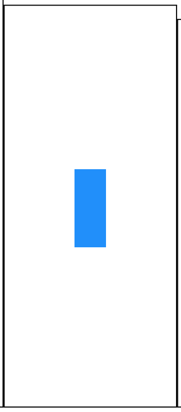
(e) Scenario 5

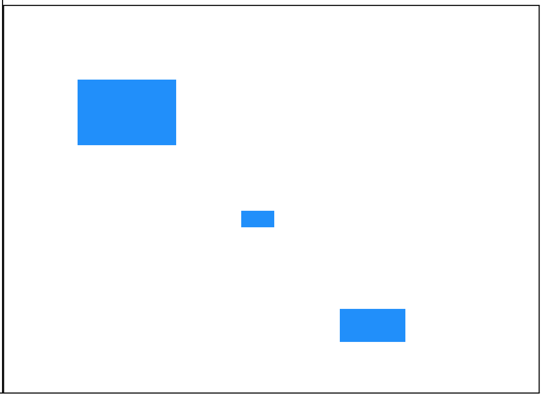
Figure 3: Wind profiles used in each scenario. Depicted are the wind roses, which give the expected wind speed in each direction. Directions are discretised into 15° bins. Each concentric circle in a rose represents an expected wind speed increase of 0.2 m/s.

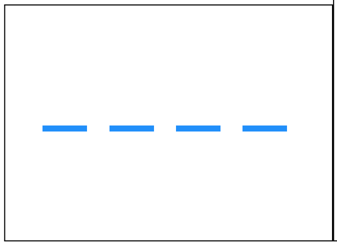
Table 2: The fixed parameters used in the evaluation.

Algorithm	Parameter	Value
TDA/ES	MAX_EVALS	1000
TDA	p	0.2
TDA	σ_{dir}	$\frac{\pi}{6}$
TDA	σ_{dist_init}	$1.05 \times$ min. turbine dist.
ES	MRI	50


(a) Scenario 1


(b) Scenario 2


(c) Scenario 3


(d) Scenario 4

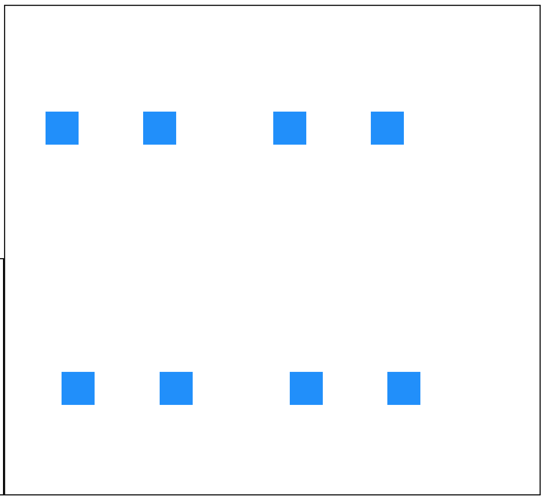

(e) Scenario 5

Figure 4: Layouts with obstacles used in each scenario. Layouts are not shown to scale.

Table 3: The algorithms used in the evaluation.

Algorithm	K	N
TDA	4	–
TDA	8	–
ES	–	1
ES	4	10
ES	4	100
ES	4	1000
ES	8	10
ES	8	100
ES	8	1000

that the experimental algorithms will have a computational overhead due to the model building and predictions made. Fortunately, the model building overhead requires constant time (since it is a function of the layout size and the number of evaluations, both of which are fixed per run) while the number of predictions made is linearly proportional to N. In comparison to the evaluation function, therefore, the model's overhead is low. We also note that the constant-time model building overhead depends on the specific regression model learning algorithm used, and will therefore vary considerably depending on choice made.

In terms of the parameters that were varied, we were interested in assessing the effects of different neighbourhood sizes (by varying K for both algorithms), and also the effect of the model in the ES (which can be varied by changing N). We therefore considered nine different algorithms in total, each differing in the value of K and N used. The specific values of K and N are given in Table 3. There are three baselines in this experiment: two variants of TDA, and one ES version with $N = 1$ (which effectively ignores the model, so K is not relevant). This third baseline amounts to an ES with a randomised mutation operator.

In terms of the exact predictive model used by the ES, we have chosen the widely-used machine learning algorithm Random Forest [16] which is easily adapted for regression.

One final issue in our set-up is the way that initial layouts for both algorithms are constructed. One choice is to create the initial layouts randomly, i.e. by placing turbines at random valid locations where they intersect neither with each other nor an obstacle. However such an approach tends to produce inferior and more variable starting conditions which may unduly influence the performance of an algorithm. Therefore Wagner uses a grid-based initialisation for TDA [2]. In this initialisation approach, turbines are placed in a grid formation with the spacing between turbines fine-tuned so that the correct number of turbines can occupy the maximum amount of space.

The difficulty with initialising layouts in a grid formation in our scenarios is the presence of obstacles: if the grid is sized to optimally fit the correct number of turbines, then some of the turbines will collide with obstacles, and the initial layout will therefore not be able to fit the requisite number of turbines. In the original paper on TDA, this was not an issue because no obstacles were present.

We therefore propose an alternative, "obstacle-friendly" means of creating an initial layout in an approximate grid formation. We use this initialisation approach for all of our algorithms. The basic idea is, first of all, tune the spacing between turbines in the grid so that the layout can fit *slightly more* turbines than are required, even if turbines are not placed on locations containing obstacles. A consequence of this is that the spacing between turbines shrinks as the area of the obstacles increases.

Once the turbines have been positioned, then some of them are randomly culled from the layout until the number of turbines is reduced to the required fixed quantity. This approach means that turbines will be mostly initialised in a grid formation around the obstacles, but some of the grid positions will be vacant. For exactness, the layout initialisation algorithm is shown as Algorithm 4.

To conclude this overview of the experiment, we report the total number of runs and repetitions we performed. For each algorithm and scenario, we conducted thirty independent trials. This meant that in total, we conducted $30 \times 5 \times 9 = 1,350$ runs from which the results in the next section are discussed.

4.3. Results

The results of our evaluation are depicted in Figure 5 using box-and-whisker plots. To understand the results, we have arranged the algorithm result sets on each plot from left to right in the same order as they appear in Table 3. The first three box-and-whisker plots depict performances of our three baseline algorithms, and the following six plots depict the results of our experimental algorithms. Each plot clearly shows the median, upper

```
Input: desired initial layout size $S$, minimum turbine distance $MTD$,
       scenario width and height $W$ and $H$, scenario obstacles $O$
begin
    /* calculate the optimal grid spacing between turbines
       */
    $spacing \leftarrow \frac{W}{2}$;
    $count \leftarrow num\_turbines\_in\_rectangle(spacing, W, H, O)$;
    while $count < S$ and $spacing > MTD$ do
        $spacing \leftarrow spacing \times 0.999$;
        $counts \leftarrow num\_turbines\_in\_rectangle(spacing, W, H, O)$;
    end
    /* add the turbines to the grid so long as they do not
       collide with any obstacles or each other            */
    $layout \leftarrow create\_empty\_layout(W, H)$;
    $x \leftarrow 0$;
    $y \leftarrow 0$;
    while $x < W$ do
        while $y < H$ do
            $place\_turbine\_if\_possible(layout, x, y, O)$;
            $y = y + spacing$;
        end
        $x = x + spacing$;
    end
    /* randomly remove turbines if too many were added    */
    while $size(layout) > S$ do
        $delete\_random\_turbine(layout)$;
    end
    /* done                                                */
    return $layout$
end
```

Algorithm 4: Algorithm used to construct the initial layout.

Table 4: Best layouts found by scenario.

Scenario	Best wake free ratio	Algorithm
1	0.9282	ES, $K = 8, N = 1000$
2	0.9264	ES, $K = 8, N = 1000$
3	0.8715	ES, $K = 8, N = 1000$
4	0.8946	ES, $K = 4, N = 10$
5	0.8599	ES $K = 8, N = 1000$

and lower quartiles, along with the minimum and maximum wake free ratios achieved by each algorithm on each scenario over thirty runs. All data points (including outliers) are included within the whiskers of the plots for conciseness.

We can make the following general observations from Figure 5.

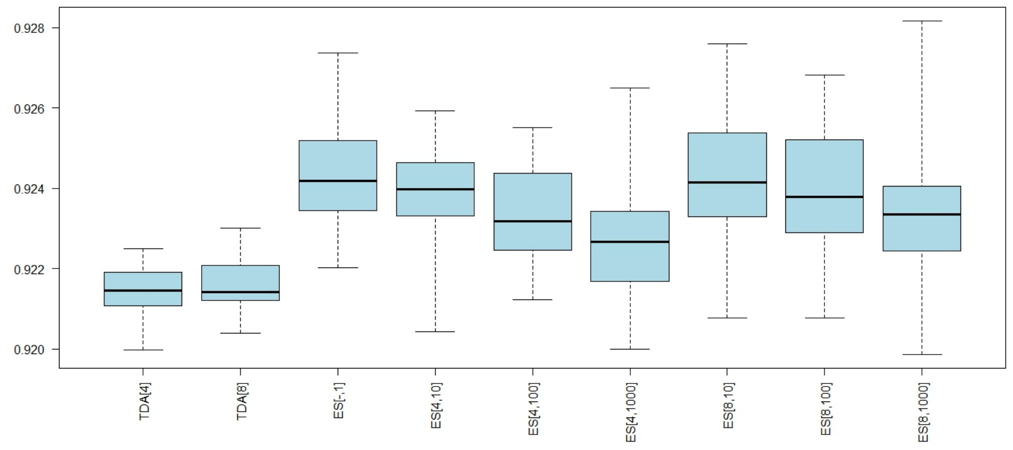
Firstly, for each scenario, the algorithm producing the overall best final layout (i.e. the algorithm with the highest "whisker") is uniformly one of the experimental algorithms. In four cases out of five it is the ES with $K = 8$ and $N = 1000$. Specific details about the best layout found for each scenario and which algorithm found it are given in Table 4. The fact that our proposed approach consistently finds the best layouts overall is encouraging.

An examination of the median performances that the various algorithms tells a different story, however. For Scenario 1, the baseline ES with $N = 1$ is at least equal to the median performance of the best experimental algorithm. Similarly, for Scenario 2, the baseline method's median is only slightly less than the best experimental algorithm's median. It is only for the latter three scenarios that there is a clearer distinction between the median of the best baseline algorithm and the median of the best experimental algorithm. The difference in distributions is clearest in the case of Scenario 5, in which the interquartile ranges of the algorithms with $K = 8$ do not overlap the baselines at all.

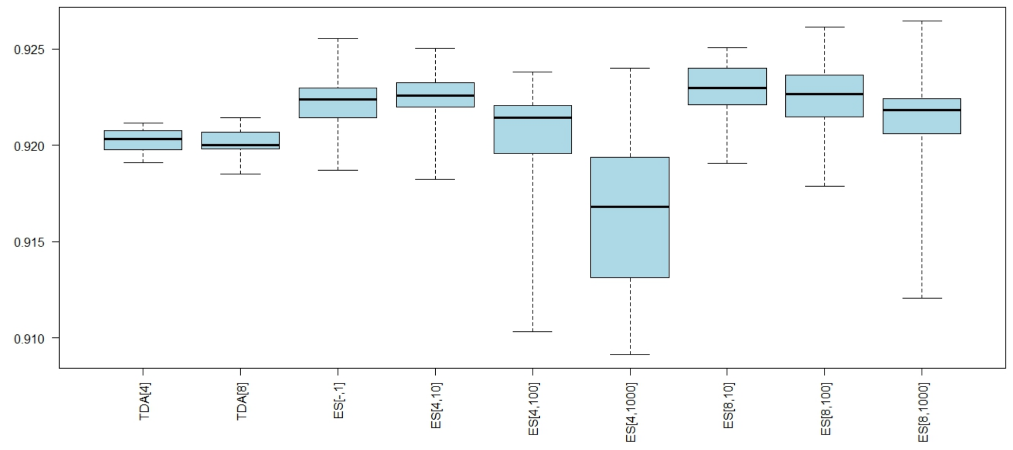
With respect to neighbourhood sizes, the results show that smaller neighbourhood sizes in general (i.e. where $K = 4$) lead to poorer results. This may be due to underfitting because the number of features used by the predictive model is smaller.

Poorer median performance also appears to be correlated with higher values of N. If only median values are considered, then a modest value of $N = 10$ is optimal in most cases.

Ironically however, it is the cases with $N = 1000$ that mostly produce the

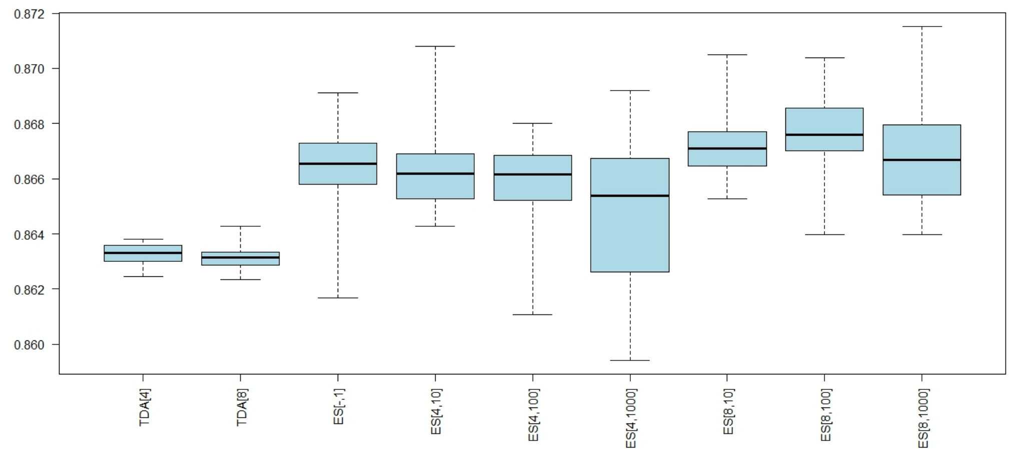


(a) Scenario 1

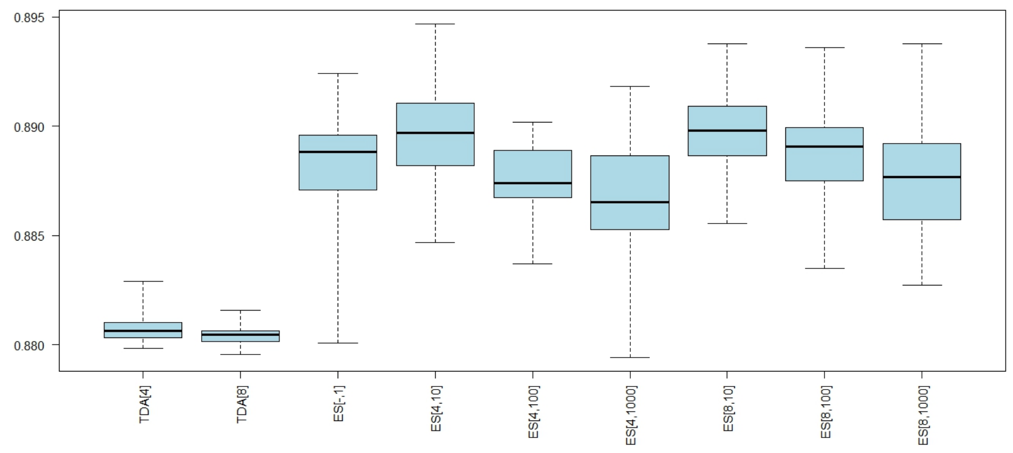


(b) Scenario 2

Figure 5: Results depicted as box-and-whisker plots.

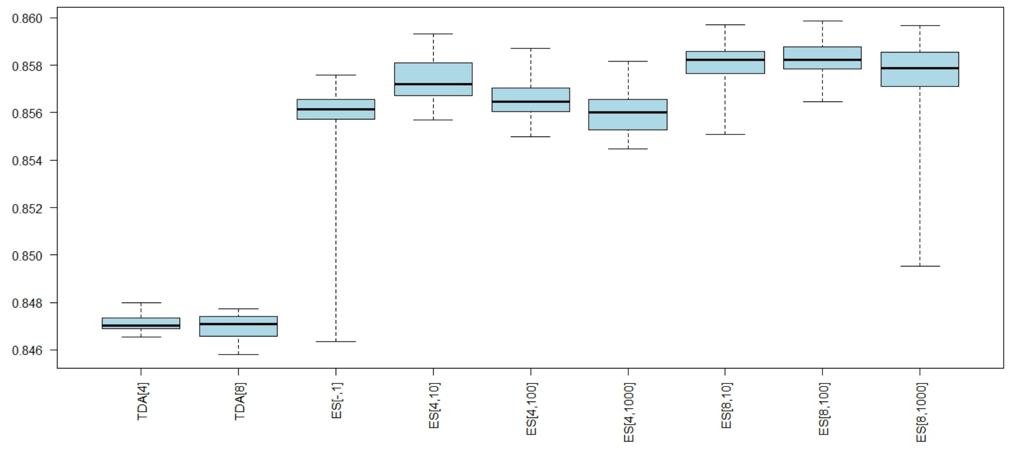


(c) Scenario 3



(d) Scenario 4

Figure 5: Results depicted as box-and-whisker plots.



(e) Scenario 5

Figure 5: Results depicted as box-and-whisker plots.

Table 5: Reported performance of TDA in the 2014 Wind Farm Layout Optimisation competition.

Scenario	Reported best wake free ratio
1	0.9151
2	0.9112
3	0.8535
4	0.8777
5	0.8373

best layouts on all scenarios except for Scenario 4. A possible explanation for this is that higher values of N lead to a greater variance (i.e. a higher chance of discovering better or worse layouts) across individual runs. The longer "whiskers" for these algorithms on the plots are evidence for this.

It is useful to compare the results of our evolutionary strategies to TDA, the current state-of-the-art approach in the literature. In most cases, the ES variants outperforms TDA by a wide margin. Furthermore, the distribution of results is much narrower for TDA than it is for the ES variants.

We were curious as to whether TDA's lower performance was a consequence of our implementation of it, or a genuine reflection of TDA's true likely performance. To that end, we examined the result of the 2014 Wind

Farm Layout Optimisation competition in which TDA was an entrant. The results of TDA on the five scenarios, as reported in the competition results, are given in Table 5. They show that, if any conclusion is to be drawn, it is that our implementation of TDA actually slightly outperforms the version used in the competition. Specifically, the median results in the plots are actually slightly higher than the wake free ratios shown in Table 5.

It is interesting to speculate as to the reason why TDA's performance is below that of the other algorithms. In our opinion, TDA's performance is related to the size of the layouts. Scenario 2 in the evaluation set is the layout with the smallest number of turbines (only 150) and Figure 5(b), which concerns this scenario, shows that TDA is comparable to the other algorithms. Since 1,000 evaluations are performed per run, it can be expected that TDA will mutate each individual turbine in Scenario 2 $\frac{1000}{150} = 6.66$ times on average per run. However for the largest layout (Scenario 5 with 910 turbines), the number of expected mutations per turbine drops to $\frac{1000}{910} = 1.10$. TDA therefore may be the optimal choice for smaller layouts, but for larger layouts it suffers because it requires more evaluations to achieve the same degree of position tuning. A future modification to the TDA algorithm could alleviate this problem.

5. Impact of Model Error on Algorithm Performance

In the final section of the evaluation portion of this paper, we examine the quality of the predictive models that our proposed algorithms are learning. In machine learning, a critical factor in model performance is the amount of training data supplied to the model. A smaller amount of training data may result in an underfit model, which consequently is less accurate than it could be if more training data were supplied.

Unfortunately, data quantity may be an issue for our proposed ES approach because the data used to construct the models is dependent on the number of turbines in the layout. Specifically, as Algorithm 1 shows, the number of examples in each dataset is equal to the number of turbines in the layout. This means that for some layouts (e.g. Scenario 2) the number of training examples is low, whereas for other layouts (e.g. Scenario 5) the number of examples is much higher.

To explore this issue, we ran another experiment in which a single algorithm (ES with $K = 8$ and $N = 1000$) was executed on each of the five

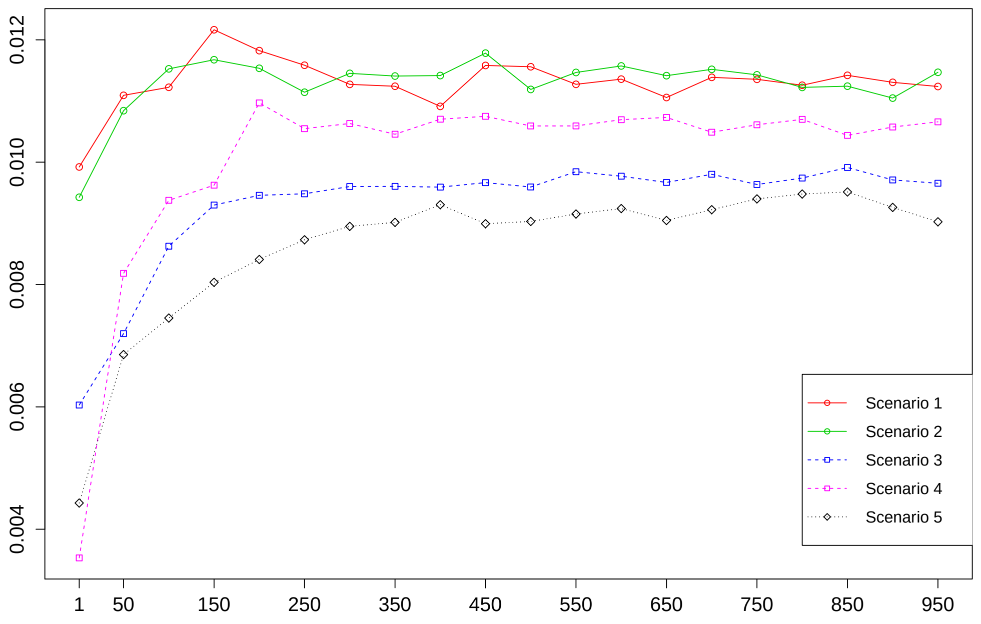


Figure 6: Cross-validated model RMSE error (y) by evaluation number (x) as measured during one run of the ES with $K = 8$ and $N = 1000$. The Model Rebuild Interval (MRI) in all cases was 50, and the error was estimated each time the model was built or rebuilt.

scenarios. All other parameters were set to the same values as in the previous experiments. However, we did make one change to the implementation of our algorithm: whenever the model was rebuilt, its generalisation error was estimated by performing a ten-fold cross-validation experiment on the training data. The results are depicted in Figure 6.

Figure 6 is interesting for two main reasons. Firstly, it shows model error increasing in the early stages of the ES' run across all scenarios. Although seemingly counter-intuitive, this makes sense because layouts are initialised with a grid formation (see Algorithm 4) and therefore the wake free ratios are likely more predictable in the early stages of the run. However, as the layouts become more randomised over time, the prediction problem becomes more difficult and model error increases.

The second interesting aspect of Figure 6 is the ranking in terms of error. A simple comparison with Table 1 shows that the model error decreases as the number of turbines in the scenario increases. Specifically, Scenarios 1 and 2, with the smallest number of turbines, experiences the highest model error; conversely Scenario 5, with the most turbines, experiences the least error.

This indicates that to some extent, the models may indeed be underfitting the problem for scenarios with a smaller number of turbines, and therefore greater performance gains may be possible if this issue is addressed algorithmically by finding a better way to obtain data for building the models.

6. Conclusion

To summarise, we have investigated a novel approach to optimising wind farm layouts in which an ES is combined with an informed mutation operator (based on machine learning) to bias the search for wind farm layouts with low expected velocity deficits/high expected wake free ratios. We have evaluated our proposed algorithm on five challenging wind farm simulation scenarios, and have shown that our approach finds the best layout compared to two baseline algorithms, the Turbine Displacement Algorithm [2] and a more standard evolutionary strategy.

One issue for further investigation is whether the reductions in overall velocity deficit that we have observed would correspond to actual increases in power output for a real farm. Admittedly, the gains "in silico" are small but two caveats are worth mentioning.

Firstly, all of our experimental runs had a limited budget of 1,000 evaluations. This enabled us to perform multiple repeats and therefore obtain statistics about average algorithm performance. In practice, the evaluation budget is only limited by compute power and therefore long runs of much more than 1,000 evaluations (with correspondingly fewer repetitions) would be feasible in an industrial setting with only a single scenario of concern. Differences between algorithms may become more pronounced under such conditions.

Secondly, we should also point out that the evaluation function we used was chosen primarily to enable comparison of results reported in the literature, and its limitations are well-known (see, for example, the discussion of models in the survey by Herbert-Acera et al. [8]). Beyond the Kusiak & Song approach, development of new wake models is a current area of research. For example, a three-dimensional (as opposed to a two dimensional) decision model is proposed by Song et al. [17], and an improved variant of Kusiak & Song's method is proposed by Lückehe at al. [18]. Computational fluid dynamics (CFD) is also commonly used, for example in commercial software such as WaSP [19].

However, depending on the method used, the time complexity of some of the more advanced methods may be exponential or even hyper-exponential [8]. Such approaches are clearly not suitable for repeated simulation of large layouts, but they could be used occasionally to validate the approximate performance of simpler models, in a style related to surrogate modelling via problem approximation [20, 21] – this is an intriguing area of future research.

We would expect however that any final assessment of an algorithm's performance when applied to a realistic wind engineering situation will depend on some or all of the layouts being evaluated by whichever more advanced and accurate methods are available at the time.

To conclude, the results presented in this paper are encouraging and should be useful for researchers working in wind farm design automation. Extending the algorithm to cope with a variable number of turbines (as opposed to a fixed number), addressing the underfitting issue identified in Section 5, and exploring the same approach but with different wake modelling techniques are our future areas of investigation.

[1] A. Kusiak, Z. Song, Design of wind farm layout for maximum wind energy capture, Renewable Energy 35 (2010) 685–694.

[2] M. Wagner, J. Day, F. Neumann, A fast and effective local search algorithm for optimizing the placement of wind turbines, Renewable Energy 51 (2013) 64–70.

[3] Global Wind Energy Council, Global Wind Energy Outlook 2014, 2014.

[4] London Array brochure, Online PDF brochure, retrieved 9 Nov 2015, `http://www.londonarray.com/wp-content/uploads/London-Array-Brochure.pdf`.

[5] Alta Wind Energy Center, WWW, retrieved 9 Nov 2015, `http://www.power-technology.com/projects/alta-wind-energy-center-awec-california/`.

[6] J. Watts, Winds of change blow through china as spending on renewable energy soars, `http://www.theguardian.com/world/2012/mar/19/china-windfarms-renewable-energy`, The Guardian.

[7] M. Samorani, The wind farm layout optimization problem, in: P. Pardolas (Ed.), Handbook of Wind Power Systems, Springer-Verlag, 2013, pp. 21–38.

[8] J. F. Herbert-Acero, O. Probst, P.-E. Réthoré, G. C. Larsen, K. K. Castillo-Villar, A review of methodological approaches for the design and optimization of wind farms, Energies 7 (11) (2014) 6930. doi:10.3390/en7116930.
URL `http://www.mdpi.com/1996-1073/7/11/6930`

[9] G. Mosetti, C. Poloni, B. Diviacco, Optimization of wind turbine positioning in large wind farms by means of a genetic algorithm, Journal of Wind Engineering and Industrial Aerodynamics 51 (1) (1994) 105–116.

[10] K. Rasheed, H. Hirsh, Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan Kaufmann, 2000, pp. 628–635.

[11] M. Mayo, M. Daoud, An adaptive model-based mutation operator for the wind farm layout optimisation problem, in: Proc. IEEE Conference on Systems, Man and Cybernetics, 2015.

[12] D. Wilson, E. Awa, S. Cussat-Blanc, K. Veeramachaneni, U.-M. O'Reilly, On learning to generate wind farm layouts, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13, ACM, 2013, pp. 767–774.

[13] D. Wilson, S. Cussat-Blanc, K. Veeramachaneni, U. O'Reilly, H. Luga, A continuous development model for wind farm layout optimization, in: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, GECCO '14, ACM, New York, NY, USA, 2014, pp. 745–752.

[14] D. Wilson, `http://www.irit.fr/wind-competition/`, URL (2015).

[15] H. Beyer, H. Schwefel, Evolution strategies: A comprehensive introduction, Journal Natural Computing 1 (1) (2002) 3–52.

[16] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.

[17] Z. Song, Z. Zhang, X. Chen, The decision model of 3-dimensional wind farm layout design, Renewable Energy 85 (2016) 248 – 258. doi:http://dx.doi.org/10.1016/j.renene.2015.06.036.
URL `http://www.sciencedirect.com/science/article/pii/S0960148115300586`

[18] D. Lückehe, M. Wagner, O. Kramer, On evolutionary approaches to wind turbine placement with geo-constraints, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, GECCO '15, 2015.

[19] Wind Atlas Analysis and Application Program (WAsP), available online `http://www.wasp.dk/` (date accessed: 10 nov 2015).

[20] Y. Jin, Surrogate-assisted evolutionary computation: recent advances and future challenges, Swarm and Evolutionary Computation 1 (2011) 61–70.

[21] M. Bhattacharya, Evolutionary approaches to expensive optimisation, International Journal of Advanced Research in Artificial Intelligence 2 (3) (2013) 53–59.