Optimization Models for Turbine Location in Wind Farms

Jan Kristian Haugland
University of Bergen
admin@neutreeko.net

May 24, 2012

Acknowledgements

I would like to thank my advisor, Professor Dag Haugland, for his support, guidance and tireless proofreading throughout the project.

I would also like to thank the Department of Informatics for providing me with very good working conditions.

Contents

Chapter 1

Introduction

In a world with increasing demand for electrical energy, maximal production of coal and other fossil fuels possibly being reached within decades, and increased amount of extreme weather due to global warming, the use of wind turbines as a source of electrical energy is more topical than ever before.

A number of wind farms are already in existence (confer [1]). Many of the world's largest onshore wind farms can be found in the United States, but also in other countries like China and Australia. Smaller onshore wind farms can be found in a number of different countries all over the globe. The capacity of each farm can be anything up 720 MW (attained by the Alta Wind Energy Center in the U.S.).

The largest offshore wind farms are found in Europe, particularly in Denmark and Great Britain. The largest one is Walney off the coast of Cumbria in England, with a capacity of 367 MW. In Norway, Vestavind Offshore [2] is planning to build Havsul, a wind farm with a capacity of up to 350 MW, which will cover the energy consumption of 50,000 households.

In this thesis we investigate how to design wind farms, onshore or offshore, so as to maximize the power output. The cost that is in the focus of our attention is the wake effect, i.e., the fact that the wind velocity is reduced downstream behind a turbine and causes a reduced power output from the turbines thus located.

More specifically, we consider the following problems.

Problem 1: Suppose we are given an offshore region with some data on the wind conditions, and a number of turbines that are to be placed within the region. We would like to find the placement that returns the maximal amount of power.

Problem 2: Similar to Problem 1, but now the number of turbines is not fixed, and there are only a finite number of possible locations. We take the cost of installing and running a wind mill into consideration, and maximize

the net present value of the wind farm.

These problems can be modelled along these lines:

Problem 1
Objective function: Total power output of the turbines, averaged over a given distribution of wind directions and mean wind velocities.
Variables: Coordinates of the turbines.
Constraints: The region in which the turbines can be placed is bounded. The distance between any two turbines is bounded from below.

Problem 2
Objective function: Net profit, i.e., power output (possibly converted) minus costs.
Variables: Decision variables for whether a location should contain a turbine.
Constraints: Constraints on the distances between turbines may apply, depending on the procedure for generating the set of possible locations.

In order to handle these problems, we need a model for the wake effect, and an optimization method.

1.1 Wake models

A method for estimating the wake effect which has become increasingly popular in recent years is to solve some simplified version of the Navier-Stokes equations. See, for instance, Heggelund and Skaar [3]. Moreover, a wake effect tool called Fuga (confer [4]) developed at Risø solves linearized Reynolds-averaged Navier-Stokes equations for farms with 200 turbines in a matter of seconds. It is not publicly available currently.

Over the years, several wake models that are considerably simpler have been proposed. Three well known examples are the Jensen model, the Frandsen model and the Larsen model. A brief description of each of these will be given in Chapter 2.

In this thesis, we are going to consider a wake model, based on the Jensen model, described by Katić, Højstrup and Jensen [5]. This model has been used to this day, or at least until recently, by the Wind Atlas Analysis and Application Program [6] ("a PC program for predicting wind climates, wind resources and power productions from wind turbines and wind farms").

Based on the contents of Katić *et al.*'s paper, we are proposing an improved version which is still fairly simple, but appears to fit the measurements better. This is the topic of Chapter 3.

1.2 Wind farm optimization

In the field of nonlinear programming, a number of computational optimizations techniques are in existence, such as Newton's method, conjugate gradient methods, and variants of these. Nowadays, a natural approach to an optimization problem is to apply commercial solver software.

However, exact optimization sometimes requires prohibitively long running time. When the task at hand is wind farm optimization, it is not unreasonable to expect that this is the case. Assuming that local optimization packages, which are often included in commercial solvers, are still not good enough, we then have to develop our own heuristic methods.

Let us take a look at one of today's perhaps most promising wind farm optimization platforms.

There is an EU-project led by Risø called TopFarm that was reported to be completed in the summer of 2011. The result is a simulation platform that can optimize the total economic benefits.

The following is from Risø's own web pages [7]:

"The core of the optimization tool consists of 5 calculation modules. The first module consists of models (of varying complexity) that describe the wind inside a wind farm. The second module is a detailed model of how the wind affects each wind turbine in a wind farm. The input to this module is the wind field from the first module and the result is production data and load data from the individual turbines.

The third module comprises models of the control system at both wind farm level and wind turbine level, while the fourth module contains cost models, which make it possible to formulate the optimization problem in economic terms.

The fifth and the last module is a package of optimization algorithms that, with the input from the other four modules, generates the optimum layout of a given wind farm. This is the module that gave rise in the case story to the new location of turbines at Middelgrunden wind farm, as seen on the picture."

The paper by Réthoré *et al.* [8] gives more details.

TopFarm appears to be a powerful tool in wind farm optimization, and its optimization method is in part a genetic algorithm, which belongs to the class of metaheuristic methods. Although genetic algorithms are considered to be the most popular approach in wind park optimization (confer [9]), they can be rather slow (confer [10]). This opens up for the possibility that it is worthwhile to consider simpler methods.

There are a number of heuristic methods in the literature (confer [11]). Trying out all of them is beyond the scope of this thesis, and we will instead compare three particularly simple ones. This is the topic of Chapter 4, where we apply some rather simplified test cases. Chapter 5 deals with experiments on a bigger variety of test cases, with the same methods (more

or less) as in Chapter 4.

Chapter 2

Background

The descriptions of the models given in Sections 2.1 and 2.2 are based in part on [3].

2.1 The Jensen model

When wind with initial velocity v flows through the rotor of a wind turbine, it will normally lose some kinetic energy to the rotor blades, and the wind velocity immediately behind the rotor will be αv for some α, $0 < \alpha < 1$. The exact value of α depends on the pitching of the blades. The power coefficient, which denotes the ratio of the power that is extracted from the wind by the turbine to the available power in the wind (assuming an ideal massless rotor and no heat transfer), is given by

$$C_p = \frac{1}{2}(1 + \alpha - \alpha^2 - \alpha^3)$$

which leads to Betz' law [12]: The maximal value of C_p is $\frac{16}{27} \approx 0.593$.

In the Jensen model [13], it assumed that this maximum is attained. Since

$$\frac{\mathrm{d}C_p}{\mathrm{d}\alpha} = \frac{1}{2}(1 - 2\alpha - 3\alpha^2) = \frac{1}{2}(1 + \alpha)(1 - 3\alpha)$$

it follows that $\frac{\mathrm{d}C_p}{\mathrm{d}\alpha} > 0$ for $0 < \alpha < \frac{1}{3}$ and $\frac{\mathrm{d}C_p}{\mathrm{d}\alpha} < 0$ for $\frac{1}{3} < \alpha < 1$, so that the optimal value of α must be $\frac{1}{3}$.

This is not necessarily optimal if there is more than one turbine involved, due to the wake effect. For example, if two turbines (with the same blade pitch) are located rather close to each other, and the wind direction is always from the first turbine to the second one or vice versa, the total power output is given by

$$P = \frac{C_p \rho A}{2}(v^3 + (\alpha v)^3) = \frac{\rho A}{4}(1 + \alpha - \alpha^2 + \alpha^4 - \alpha^5 - \alpha^6)v^3$$

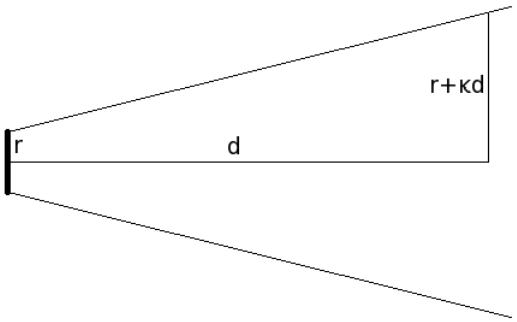

Figure 2.1: Linear expansion of the wake behind a single turbine

where ρ is the density of air, and A is the rotor area. Dividing off constant terms and setting the derivative with respect to α to zero gives

$$1 - 2\alpha + 4\alpha^3 - 5\alpha^4 - 6\alpha^5 = 0 \Rightarrow (1+\alpha)^2(1-2\alpha)(1-2\alpha+3\alpha^2) = 0$$

and it follows that the optimal value of α in this case is $\frac{1}{2}$.

However, a much more realistic scenario is that turbines are placed some distance apart, and that with wind directions distributed over a full rotation, the wake effect will be small for most turbines at any given time. It therefore makes sense to retain $\alpha = \frac{1}{3}$, while the optimal value in any given case is presumably at least slightly bigger.

With a fixed value of the power coefficient, it follows that the power output of a single turbine is a fixed constant times v^3. Thus for simplicity, we will define the *profit* of a single turbine to be v^3.

Furthermore, the Jensen model is based on a linear expansion of the wake region (confer Fig. 2.1), and on mass conservation. This means that the wind velocity in the wake a given distance d behind a turbine can be expressed as $(1-\delta)v$ where δ, called the velocity deficit coefficient, must be $\frac{2}{3}$ for $d = 0$, and is inversely proportional to the area of the corresponding cross section of the wake region. It follows that

$$\delta = \frac{2}{3}\left(\frac{r}{r+\kappa d}\right)^2$$

up to a distance $r + \kappa d$ from the centre line, where κ is the decay constant (also called entrainment constant, e.g., in [14]).

2.2 Other wake models

Besides the Jensen model, two relatively simple wake models that occur in the literature are the Frandsen model [15] and the Larsen model [16]. Like the Jensen model, they have mass conservation and a top hat-shaped

distribution across the wake, and the main difference between the three models appears to be the expansion of the wake (which of course affects the velocity deficit). The Frandsen model and the Larsen model both predict a concave expansion.

The Frandsen model, which assumes a regular array-geometry of the layout, says that the wake expansion has the form

$$r = r_0 \max\left(\beta, \alpha \frac{x}{2r_0}\right)^{\frac{1}{2}}$$

in which α and β are constants.

The Larsen model yields

$$r = \left(\frac{35}{2\pi}\right)^{\frac{1}{5}} \left(3c_1^2\right)^{\frac{1}{5}} \left(C_T A x\right)^{\frac{1}{3}}$$

in which C_T is the thrust coefficient, A is the rotor area and c_1 is a constant related to the mixing length.

With the development of simulation software (like Fuga), it may be possible in the near future to find a simple model that outperforms all of the above models.

2.3 Katić *et al.*'s wake combination model

For multiple wakes, the combined deficit would logically appear to be the sum of the individual deficits, under the assumptions. (We will refer to this as *Jensen's model in its simplest form.*) However, this estimate is known to compare poorly with observations. In Katić *et al.*'s model, the combined deficit is instead the 2-norm of the vector of individual deficits:

$$v_j = v_0 - \sqrt{\left(\delta_{1j} v_1\right)^2 + ... + \left(\delta_{(j-1)j} v_{j-1}\right)^2}$$

where v_0 is the initial wind velocity, v_i is is the wind velocity experienced by turbine i, and δ_{ij} is the deficit coefficient at turbine j caused by turbine i's presence. The authors state that this formula is chosen so that the velocity deficit from a line of turbines will quickly reach an equilibrium level, in agreement with observations. They do not say why the 2-norm is chosen over other functions with the same qualititive property (e.g., the ν-norm for any $\nu > 1$ with $\nu \neq 2$).

2.4 Basic properties of the Jensen model

When placing turbines in a 2-dimensional region with varying wind conditions, one can expect that local optima will contain one or several instances

of three or more turbines being placed more or less on a straight line. Therefore, we will take a closer look at the 1-dimensional case in this section.

The reason why instances of several turbines being placed more or less on a straight line are to be expected has to do with wake combination. First, we consider three turbines A, B and C that are to be placed in the plane. Suppose the distance between A and B should be a fixed length d_1 and the distance between A and C should be a fixed length d_2 with $d_1 < d_2$. Suppose furthermore that the wind direction is uniformly distributed (and the wind velocity is constant or at least independent of the direction). There is a certain probability that A will be in B's wake at any given time, and a certain probability that A will be in C's wake. The expected velocity deficit at A due to B's presence depends on d_1, but not on the direction from B to A; and likewise for C and d_2. But if B is located on the line between A and C, A will experience one combined wake instead of two isolated wakes as the wind direction varies. According to Katić *et al.*'s wake combination model, the total deficit is smaller in the case of the combined wake. (In fact, since the profit is the wind velocity raised to the third power, it can be shown that this is profitable even in Jensen's model in its simplest form.)

A similar argument applies if the wind direction is not uniformly distributed, as is normally the case.

So let us assume for now that the turbines are placed on a straight line along the wind direction. More specifically, let us assume that the coordinate of each turbine is restricted to the interval [0, 1], and that the wind direction is from 0 to 1.

Placing two turbines under these assumptions is of course completely trivial. Since the velocity deficit is a decreasing function of the distance, we would like to maximize it, and so the turbines should be placed at 0 and at 1.

Placing three turbines, however, already is non-trivial. We can argue that we must still put one at 0 and one at 1, but where does the middle one go?

Let x denote the coordinate of the middle turbine. According to Jensen's model in its simplest form, the wind velocities at the three turbines are

$$v_0,\ v_1 = v_0 - \tfrac{2}{3}v_0\left(\tfrac{r}{r+\kappa x}\right)^2,\ v_2 = v_0 - \tfrac{2}{3}v_0\left(\tfrac{r}{r+\kappa}\right)^2 - \tfrac{2}{3}v_1\left(\tfrac{r}{r+\kappa(1-x)}\right)^2$$

and the total profit is equal to

$$(v_0^3 + v_1^3 + v_2^3) \tag{2.1}$$

Observe that

$$\begin{aligned}\frac{r}{rx+\kappa x}-\frac{r}{r+\kappa x} &= \frac{r\left(r-rx\right)}{\left(rx+\kappa x\right)\left(r+\kappa x\right)}\\ &= \frac{r^2\left(1-x\right)}{x\left(r+\kappa\right)\left(r+\kappa x\right)}=O\left(r^2\right)\end{aligned}$$

as $r\to 0$ so that

$$\left(\frac{r}{rx+\kappa x}\right)^2-\left(\frac{r}{r+\kappa x}\right)^2=\left(\frac{r}{rx+\kappa x}+\frac{r}{r+\kappa x}\right)O\left(r^2\right)=O\left(r^3\right) \quad (2.2)$$

Let $u=\frac{2}{3}\left(\frac{r}{r+\kappa}\right)^2$. By (2.2) we have

$$\frac{2}{3}v_0\left(\frac{r}{r+\kappa x}\right)^2=v_0\left(\frac{u}{x^2}+O\left(r^3\right)\right)=v_0\left(\frac{u}{x^2}+O\left(u^{\frac{3}{2}}\right)\right)$$

as $u\to 0$ and

$$\begin{aligned}\frac{2}{3}v_1\left(\frac{r}{r+\kappa\left(1-x\right)}\right)^2 &= v_0\left(1-\frac{u}{x^2}+O\left(u^{\frac{3}{2}}\right)\right)\left(\frac{u}{\left(1-x\right)^2}+O\left(u^{\frac{3}{2}}\right)\right)\\ &= v_0\left(\frac{u}{\left(1-x\right)^2}+O\left(u^{\frac{3}{2}}\right)\right)\end{aligned}$$

which means that (2.1) is

$$\begin{aligned}&v_0^3\left(1+\left(1-\frac{u}{x^2}+O\left(u^{\frac{3}{2}}\right)\right)^3+\left(1-u-\frac{u}{\left(1-x\right)^2}+O\left(u^{\frac{3}{2}}\right)\right)^3\right)\\ &=v_0^3\left(1+1-\frac{3u}{x^2}+O\left(u^{\frac{3}{2}}\right)+1-3u-\frac{3u}{\left(1-x\right)^2}+O\left(u^{\frac{3}{2}}\right)\right)\\ &=v_0^3\left(3-3u-\frac{3u}{x^2}-\frac{3u}{\left(1-x\right)^2}+O\left(u^{\frac{3}{2}}\right)\right) \quad (2.3)\end{aligned}$$

as $u\to 0$. If we simplify by dividing by v_0^3, subtracting the constant term and dividing by u, we are left with the task of maximizing a function of the form

$$-3-\frac{3}{x^2}-\frac{3}{\left(1-x\right)^2}+O\left(u^{\frac{1}{2}}\right) \quad (2.4)$$

In general, suppose $g(x,u)$ has the form $f(x)+O\left(u^{\frac{1}{2}}\right)$ as $u\to 0$ and that the maximum of $g(x,u)$ for a fixed $u\geq 0$ is given by $x=x_0(u)$. (We will assume that the value of $x_0(0)$ is unique, while any global maximizer will do for $u>0$.) Let $x_0=x_0(0)$ and $x_1\neq x_0$ so that $f(x_1)<f(x_0)$, and let k be a sufficiently large constant such that $g(x_0,u)>f(x_0)-ku^{\frac{1}{2}}$

and $g(x_1,u) < f(x_1) + ku^{\frac{1}{2}}$ for all u (we may assume that u is bounded from above). If $u < \left(\frac{f(x_0)-f(x_1)}{2k}\right)^2$ then $g(x_0,u) > g(x_1,u)$. It follows that $\lim_{u\to 0} x_0(u) = x_0$.

In view of this, one can verify that as $u \to 0$, the maximizer of (2.4), and therefore of (2.3), tends to 0.5. So the middle turbine goes in the centre between the other two, as we might have expected. However, suppose now that we introduce the ν-norm for the combined wake effect. Observe that for any positive constant k we have

$$\begin{aligned}\sqrt[\nu]{k + O\left(u^{\frac{1}{2}}\right)} &= k^{\frac{1}{\nu}} + \frac{1}{\nu}k^{\frac{1}{\nu}-1}O\left(u^{\frac{1}{2}}\right) + \begin{pmatrix} \frac{1}{\nu} \\ 2 \end{pmatrix} k^{\frac{1}{\nu}-2}O\left(u\right) + ... \\ &= \sqrt[\nu]{k} + O\left(u^{\frac{1}{2}}\right)\end{aligned}$$

as $u \to 0$, by Newton's generalized binomial theorem. This yields

$$\begin{aligned} v_2 &= v_0\left(1 - \sqrt[\nu]{u^\nu + \left(\frac{u}{(1-x)^2} + O\left(u^{\frac{3}{2}}\right)\right)^\nu}\right) \\ &= v_0\left(1 - u\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}} + O\left(u^{\frac{1}{2}}\right)}\right) \\ &= v_0\left(1 - u\left(\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}} + O\left(u^{\frac{1}{2}}\right)\right)\right) \\ &= v_0\left(1 - u\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}} + O\left(u^{\frac{3}{2}}\right)\right)\end{aligned}$$

so that (2.1) is

$$\begin{aligned} & v_0^3\left(1 + \left(1 - \frac{u}{x^2} + O\left(u^{\frac{3}{2}}\right)\right)^3 + \left(1 - u\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}} + O\left(u^{\frac{3}{2}}\right)\right)^3\right) \\ &= v_0^3\left(1 + 1 - \frac{3u}{x^2} + O\left(u^{\frac{3}{2}}\right) + 1 - 3u\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}} + O\left(u^{\frac{3}{2}}\right)\right) \\ &= v_0^3\left(3 - \frac{3u}{x^2} - 3u\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}} + O\left(u^{\frac{3}{2}}\right)\right)\end{aligned}$$

Simplifying as before and ignoring the $O\left(u^{\frac{1}{2}}\right)$ term yields

$$-\frac{3}{x^2} - 3\sqrt[\nu]{1 + \frac{1}{(1-x)^{2\nu}}}$$

and setting the derivative with respect to x to zero gives

$$\frac{6}{x^3} - 6\left(1 + \frac{1}{(1-x)^{2\nu}}\right)^{\frac{1-\nu}{\nu}} \frac{1}{(1-x)^{2\nu+1}} = 0$$

$$\begin{aligned}
\Rightarrow x^3 &= (1-x)^{2\nu+1}\left(1 + \frac{1}{(1-x)^{2\nu}}\right)^{\frac{\nu-1}{\nu}} \\
\Rightarrow \frac{x^3}{(1-x)^3} &= \left((1-x)^{2\nu} + 1\right)^{\frac{\nu-1}{\nu}} \\
\Rightarrow \frac{x}{1-x} &= \left(1 + (1-x)^{2\nu}\right)^{\frac{\nu-1}{3\nu}} \\
\Rightarrow \frac{1-x}{x} &= \frac{1}{\left(1 + (1-x)^{2\nu}\right)^{\frac{\nu-1}{3\nu}}} \\
\Rightarrow x &= \frac{1}{1 + \frac{1}{\left(1+(1-x)^{2\nu}\right)^{\frac{\nu-1}{3\nu}}}}
\end{aligned}$$

which can be solved numerically using recursion. That is, we can start with a reasonable guess like $x_0 = 0.5$ and apply

$$x_{i+1} = \frac{1}{1 + \frac{1}{\left(1+(1-x_i)^{2\nu}\right)^{\frac{\nu-1}{3\nu}}}}$$

for $i = 0, 1, 2, ...$ It turns out that the sequence $(x_0, x_1, x_2, ...)$ converges rapidly. For $\nu = 2$ we get $\lim_{i\to\infty} x_i = 0.502477754...$, and hence we have

$$\lim_{u\to 0} x_0(u) = 0.502477754...$$

For higher values of ν, the value of $\lim_{i\to\infty} x_i$ converges back to 0.5. Already for $\nu = 3$, we get the value $0.5008526344....$

For a higher number of turbines, it is reasonable to expect, based on these numbers, that the optimal placement is almost, but not quite evenly spaced for $\nu > 1$. We shall return to this matter in Section 4.2.4.

Table 2.1: Wind conditions at Oseberg A Platform

Direction	Percentage
N	7%
NNE	6%
NE	2%
ENE	1%
E	2%
ESE	3%
SE	6%
SSE	13%
S	10%
SSW	7%
SW	7%
WSW	7%
W	5%
WNW	4%
NW	5%
NNW	7%

2.5 Wind conditions

Wind statistics is available on the "Windfinder" web site [17]. For the test cases of Chapter 4, we used the conditions at Oseberg A Platform outside the coast of Norway, given in Table 2.1. Apparently, statistics on velocity and direction simultaneously is unavailable.

Chapter 3

Wake model

In this chapter we propose a new wake model that we will use in the computations.

3.1 The wake behind a single turbine

Our starting point is Katić *et al.*'s model. It is in turn based on the Jensen model which was described in the previous chapter. Recall that the velocity deficit coefficient is given by

$$\delta = \frac{2}{3}\left(\frac{r}{r+\kappa d}\right)^2 \tag{3.1}$$

up to a distance $r + \kappa d$ from the centre line. In [5], κ is set to 0.075. (According to the Wind Atlas Analysis and Application Program [6], it may be better to use the smaller value of 0.04 offshore, whereas 0.05 is suggested in [18]. We have assumed $\kappa = 0.075$ for consistency.) If only a fraction ρ of the rotor circle is in the wake, the deficit is multiplied by ρ (according to [6]).

In their article, Katić *et al.* state that the aim of this simple model is not to describe the velocity field accurately; a Gaussian distribution of the deficit across the wake is more commonly assumed. A bell-shaped distribution is also evident in Fig. 4 of the article. On this basis, and since a certain level of accuracy in the velocity field description is required for our purpose, we propose a wake model with a Gaussian distribution across the wake, and mass conservation like in the original model. Immeditately behind a turbine, the velocity deficit coefficient is $\frac{2}{3}$ in a region with area πr^2 according to the Jensen model. Therefore, if the velocity deficit coefficient at a distance s from the centre line is given by

$$\delta = \xi e^{-\frac{s^2}{2\sigma^2}}$$

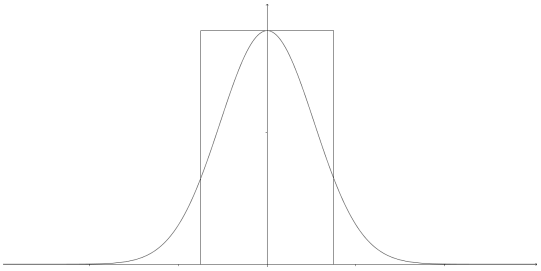

Figure 3.1: Top hat-shaped wake model vs. Gaussian wake model with a small d

where σ is some parameter corresponding to the standard deviation of a normal distribution, and ξ is some quantity independent of s, then the mass conservation equation becomes

$$\int_0^\infty 2\pi s\xi e^{-\frac{s^2}{2\sigma^2}}\,\mathrm{d}s = \frac{2}{3}\pi r^2 \Rightarrow 2\pi\sigma^2\xi = \frac{2}{3}\pi r^2$$

On solving for ξ, we get

$$\xi = \frac{r^2}{3\sigma^2}$$

and the velocity deficit coefficient is given by

$$\delta = \frac{r^2}{3\sigma^2}e^{-\frac{s^2}{2\sigma^2}}$$

If we also impose that δ should agree with (3.1) for $s = 0$, we get

$$\frac{r^2}{3\sigma^2} = \frac{2}{3}\left(\frac{r}{r+\kappa d}\right)^2$$

yielding $\sigma = \frac{r+\kappa d}{\sqrt{2}}$ which gives

$$\delta = \frac{2}{3}\left(\frac{r}{r+\kappa d}\right)^2 e^{-\frac{s^2}{(r+\kappa d)^2}} \tag{3.2}$$

In Figures 3.1 and 3.2 we can see how (3.2) compares with the original model. If the distance d is much smaller than r, the Jensen model might give a more realistic description of the wind velocity field than our variant. In this case, we consider what portion of the rotor circle is in the wake; this is of course less important to take into account for larger d. The angle in Figure 3.3 is $\cos^{-1}\frac{|s|}{2r}$, and the wind deficit coefficient is thus $\frac{2}{3}$ times the ratio of the gray area to the area of one circle, i.e.,

$$\frac{2}{3}\frac{2\cos^{-1}\frac{|s|}{2r} - \frac{|s|}{r}\sqrt{1-\left(\frac{s}{2r}\right)^2}}{\pi}$$

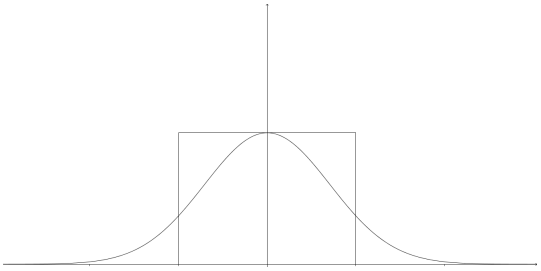

Figure 3.2: Top hat-shaped wake model vs. Gaussian wake model with a somewhat larger d

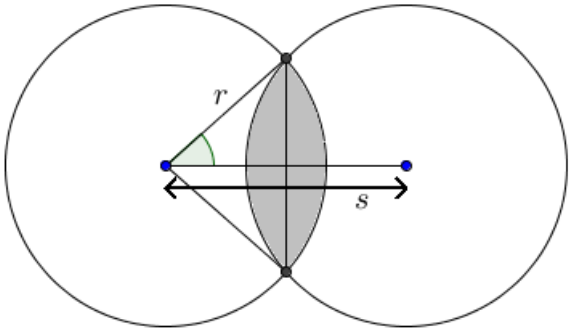

Figure 3.3: Intersection of two rotor circles

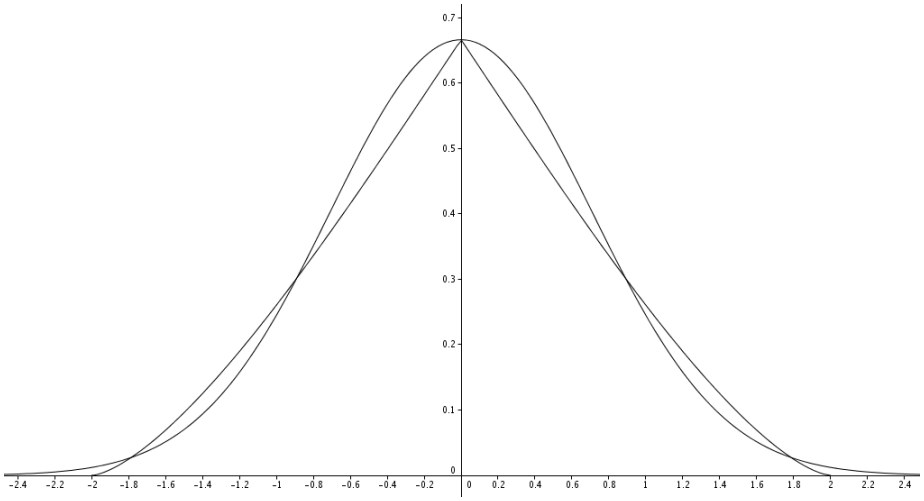

Figure 3.4: Gaussian distribution vs. overlap

for $|s| \leq 2r$. In Figure 3.4 we can see how this compares with (3.2) for $d = 0$. It would appear that (3.2) is a plausible estimate for the wind deficit coefficient when applied to other turbines whether d is large or small.

3.2 Multiple wakes

Now recall Katić *et al.*'s proposed formula for the velocity deficit in the case of several overlapping wakes:

$$v_j = v_0 - \sqrt{(\delta_{1j}v_1)^2 + ... + \left(\delta_{(j-1)j}v_{j-1}\right)^2}$$

Also recall that they assume $\kappa = 0.075$. However, later on they state that in the case of two turbines, it would be better to set κ to 0.11 after the second turbine, based on measurements done within 8 rotor radii. We propose to use the 3-norm instead of the 2-norm for combined wakes to avoid this inconsistency. If we consider three turbines with equidistant distribution on a line in the wind direction, the velocity deficit coefficient in front of the last turbine would be

$$\frac{2}{3}\sqrt{\left(\frac{r}{r+2\times 0.075d}\right)^4 + \left(\left(1-\frac{2}{3}\left(\frac{r}{r+0.075d}\right)^2\right)\left(\frac{r}{r+\beta d}\right)^2\right)^2} \tag{3.3}$$

if we assume the 2-norm and an alternative value β for κ after the second turbine, and

$$\frac{2}{3}\sqrt[3]{\left(\frac{r}{r+2\times 0.075d}\right)^6 + \left(\left(1-\frac{2}{3}\left(\frac{r}{r+0.075d}\right)^2\right)\left(\frac{r}{r+0.075d}\right)^2\right)^3} \tag{3.4}$$

if we assume the 3-norm and the fixed value 0.075 for κ. As we can see in Figure 3.5, these two intersect when d/r is near 5. For larger values, (3.4) gets closer and closer to (3.3) with $\beta = 0.075$ - but so does the ∞-norm (the maximum of the individual deficits), and there are no measurements that suggest that the combined deficit should be smaller than the ∞-norm. We conclude from this that using the 3-norm for estimating the deficit of the combination of two wakes is consistent with Katić *et al.*'s observations. And since it also generalizes in an obvious way to multiple wakes, we will incorporate it in our model.

Altering the ratio between the distances gives us Figures 3.6 and 3.7. The main characteristics are the same as in Figure 3.5.

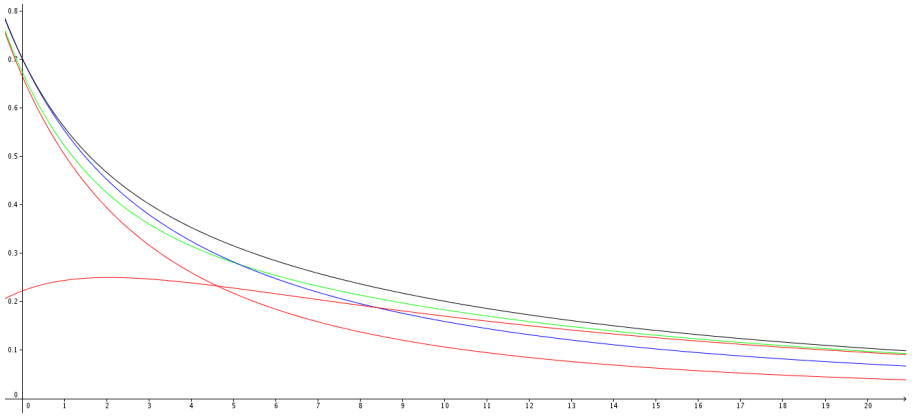

Figure 3.5: Black curve: 2-norm, $\beta = 0.075$
Blue curve: 2-norm, $\beta = 0.11$
Green curve: 3-norm
Red curves: Individual deficits

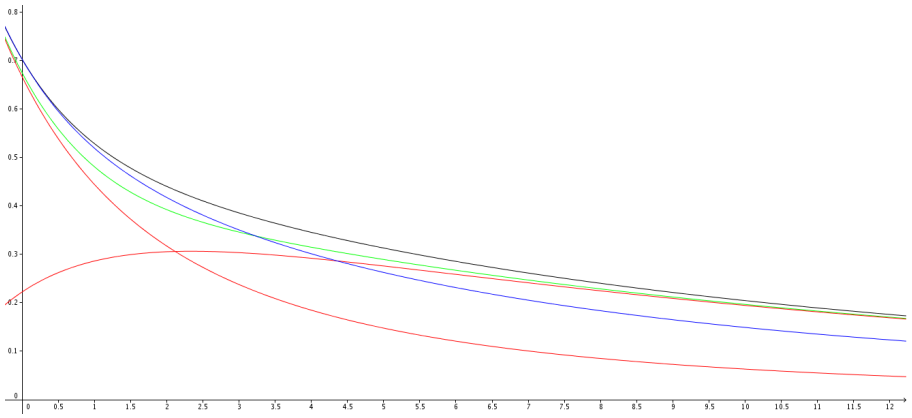

Figure 3.6: Distances between the first two and last two turbines: $2d$ and d respectively

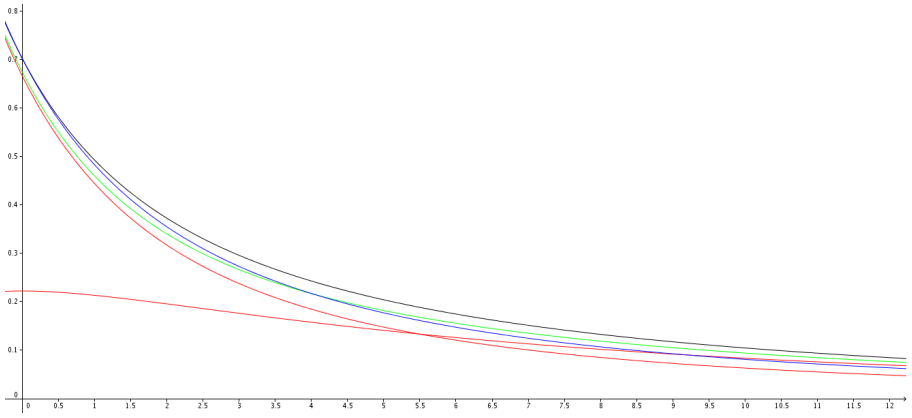

Figure 3.7: Distances between the first two and last two turbines: d and $2d$ respectively

3.3 Proposed wake model (summary)

We have arrived at the following wake model which we will apply from now on.

If turbine i is located in front of turbine j in the downstream direction, the velocity deficit coefficient at turbine j due to turbine i's presence is given by

$$\delta_{ij} = \frac{2}{3}\left(\frac{r}{r+\kappa d_{ij}}\right)^2 e^{-\frac{s_{ij}^2}{\left(r+\kappa d_{ij}\right)^2}}$$

where r is the rotor radius, $\kappa = 0.075$, d_{ij} is the length of the downstream component of the vector from turbine i to turbine j, and s_{ij} is the length of the orthogonal component.

If the turbines are sorted downstream such that $i < j$ implies $d_{ij} \geq 0$, the wind velocity deficit experienced by turbine j is given recursively for $j = 1, 2, ...$ by

$$v_j = v_0 - \sqrt[3]{\left(\delta_{1j} v_1\right)^3 + ... + \left(\delta_{(j-1)j} v_{j-1}\right)^3} \tag{3.5}$$

where v_0 is the initial wind velocity.

3.4 Varying wind conditions

In order to deal with varying wind directions we introduce permutations σ_θ that sort the turbines for any given wind direction θ, so that $\sigma_\theta(i) = j$ if turbine no. j is the ith one in the downstream direction of θ. In more mathematical terms, we require that with

$$\begin{pmatrix} d_{ij} \\ s_{ij} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{\sigma_\theta(j)} - x_{\sigma_\theta(i)} \\ y_{\sigma_\theta(j)} - y_{\sigma_\theta(i)} \end{pmatrix} \tag{3.6}$$

where the pairs (x_i, y_i) are turbine coordinates, we have $d_{ij} \geq 0$ for $i < j$. When appropriate, we can also require that σ_0 is the identity permutation in order to avoid ambiguity of the numbering.

In some cases it may be appropriate to represent these permutations by binary matrices A^θ_{ij}. With $A^\theta_{ij} = 1$ if $\sigma_\theta(i) = j$ and 0 otherwise, (3.6) becomes

$$\begin{cases} d_{ij} = \sum_{k=1}^{n} (x_k \cos\theta + y_k \sin\theta) \left(A^\theta_{jk} - A^\theta_{ik} \right) \\ s_{ij} = \sum_{k=1}^{n} (-x_k \sin\theta + y_k \cos\theta) \left(A^\theta_{jk} - A^\theta_{ik} \right) \end{cases}$$

Of course, the identity permutation corresponds to $A^0 = I$.

Chapter 4

Optimization methods

4.1 Complete mathematical models

With m denoting the number of distinct wind directions, we let $\theta_w = \frac{2\pi w}{m}$ for $w \in \{0, 1, ..., m-1\}$ and write $A(w)$ instead of A^{θ_w}. Let $R \subset \mathbb{R}^2$ denote the region in which the turbines are to be placed.

We present complete mathematical models that can be coded more or less directly into commercial solver software.

4.1.1 Problem 1

Before arriving at the model in compact form, let us take a closer look at some of the constraints involved.

$(x_i, y_i) \in R$ for $i = 1, 2, ..., n$ (coordinates for the turbines in region R)

$\sum_{i=1}^{n} A_{ik}(w) = \sum_{j=1}^{n} A_{kj}(w) = 1$ for $1 \leq k \leq n$ (ensures that each row and each column of $A(w)$ contains exactly one positive entry)

We can rewrite (3.5) so as to eliminate the root sign, and obtain

$$(v_0(w) - v_j(w))^3 = \sum_{i<j} (\delta_{ij}(w) v_i(w))^3$$

In summary, we have the following model:

$$\max_{x,y,A,d,s,\delta,v} \sum_{w<m} \left(p(w) \sum_{i=1}^{n} v_i(w)^3 \right)$$

$$\begin{aligned}
&(x_i, y_i) \in R && 1 \le i \le n \\
&A_{ij}(w) \in \{0,1\} && 1 \le i,j \le n, 0 \le w < m \\
&A(0) = I \\
&\sum_{i=1}^{n} A_{ik}(w) = 1 && 1 \le k \le n, 0 \le w < m \\
&\sum_{j=1}^{n} A_{kj}(w) = 1 && 1 \le k \le n, 0 \le w < m \\
&d_{ij}(w) = \sum_{k=1}^{n} \left(x_k \cos\theta_w + y_k \sin\theta_w\right) \left(A_{jk}(w) - A_{ik}(w)\right) && 1 \le i < j \le n, 0 \le w < m \\
&s_{ij}(w) = \sum_{k=1}^{n} \left(-x_k \sin\theta_w + y_k \cos\theta_w\right) \left(A_{jk}(w) - A_{ik}(w)\right) && 1 \le i < j \le n, 0 \le w < m \\
&\delta_{ij}(w) = \frac{2}{3} \left(\frac{r}{r + \kappa d_{ij}(w)} \right)^2 e^{-\frac{s_{ij}(w)^2}{(r+\kappa d_{ij}(w))^2}} && 1 \le i < j \le n, 0 \le w < m \\
&\left(v_0(w) - v_j(w)\right)^3 = \sum_{i<j} \left(\delta_{ij}(w) v_i(w)\right)^3 && 1 \le j \le n, 0 \le w < m \\
&d_{ij}(w) \ge 0 && 1 \le i < j \le n, 0 \le w < m
\end{aligned}$$

4.1.2 Problem 2

For Problem 2, most of the variables from Problem 1 can be pre-calculated and inserted as constants in the solver code. Thus x_k and y_k for $k = 1, 2, ..., n$ are fixed (and possibly placed in a regular grid), and we introduce the binary decision variables $\alpha_1, ..., \alpha_n$ so that α_k indicates whether or not there will be a turbine in location (x_k, y_k). We also introduce a fixed cost c associated with the installation of a single turbine; it would also have been possible to consider cabling costs or other aspects. Note that a constant cost per turbine makes no difference to Problem 1.

Since

$$\left(v_0(w) - v_j(w)\right)^3 = \begin{cases} \sum\limits_{i<j} (\delta_{ij}(w) v_i(w))^3, & \sum\limits_{i=1}^{n} A_{ji}\alpha_i = 1 \\ v_0(w)^3, & \text{otherwise} \end{cases}$$

it follows that

$$\left(v_0(w) - v_j(w)\right)^3 = \left(\sum_{i=1}^{n} A_{ji}\alpha_i \right) \left(\sum_{i<j} (\delta_{ij}(w) v_i(w))^3 \right) + \left(1 - \sum_{i=1}^{n} A_{ji}(w)\alpha_i \right) v_0(w)^3$$

$$\Rightarrow v_0(w)^3 - (v_0(w) - v_j(w))^3 = \left(\sum_{i=1}^{n} A_{ji}(w)\alpha_i\right)\left(v_0(w)^3 - \sum_{i<j}\left(\delta_{ij}(w)v_i(w)\right)^3\right)$$

Pre-calculated parameters:

$w \in \{0, 1, ..., m-1\}$; $\theta_w = \frac{\pi w}{16}$

x_i and y_i for $i = 1, 2, ..., n$ (coordinates for turbines)

$A(w)$ - binary $n \times n$-matrix

$A(0) = I$

$$\sum_{i=1}^{n} A_{ik}(w) = \sum_{j=1}^{n} A_{kj}(w) = 1$$

$$d_{ij}(w) = \sum_{k=1}^{n} \left(x_k \cos\theta_w + y_k \sin\theta_w\right)\left(A_{jk}(w) - A_{ik}(w)\right) \text{ for } 1 \le i < j \le n$$

$$s_{ij}(w) = \sum_{k=1}^{n} \left(-x_k \sin\theta_w + y_k \cos\theta_w\right)\left(A_{jk}(w) - A_{ik}(w)\right) \text{ for } 1 \le i < j \le n$$

$$\delta_{ij}(w) = \frac{2}{3}\left(\frac{r}{r+\kappa d_{ij}(w)}\right)^2 e^{-\frac{s_{ij}(w)^2}{\left(r+\kappa d_{ij}(w)\right)^2}} \text{ for } 1 \le i < j \le n$$

$d_{ij}(w) \ge 0$ for $1 \le i < j \le n$

Model:

$$\max_{\alpha, v}\left(\sum_{w<m}\left(p(w)\sum_{i=1}^{n} v_i(w)^3\right) - c\sum_{i=1}^{n}\alpha_i\right)$$

$$\alpha_i \in \{0, 1\} \qquad 1 \le i \le n$$

$$v_0(w)^3 - \left(v_0(w) - v_j(w)\right)^3 = \left(\sum_{i=1}^{n} A_{ji}(w)\alpha_i\right)\left(v_0(w)^3 - \sum_{i<j}\left(\delta_{ij}(w)v_i(w)\right)^3\right) \quad 1 \le j \le n, 0 \le w < m$$

4.2 Heuristic methods

An alternative to using exact methods is to implement heuristic algorithms that require less calculation. We consider two different methods for Problem 1 and one for Problem 2.

As we go along, we are going to test the methods on a simple case in order to get an idea of the performance. Based on these preliminary tests, we make a selection of solution methods to undergo a detailed experimental evaluation in Chapter 5. The case we will consider most frequently is the following.

Region $R_0 = \{(x, y)|0 \leq x \leq 10, 0 \leq y \leq 10\}$; rotor radius $r_0 = 0.1$.

The wind directions are given in Table 4.1 and are based on those of Oseberg (cf. Section 2.5). Recall that the data consisted of percentage numbers for the distributions on 16 wind directions. As the numbers add up to 92%, we suspect that all the numbers have been rounded downwards. Thus the weights we will use in the implementations are $2 \times$ percentage $+ 1$. We preferred a finer partition for the model and added a direction in the middle of any two consecutive directions in the table, and simply set the weight for that direction to be the mean of the weights for the two adjacent directions.

For simplicity, the mean wind velocity in any direction has been assumed to be equal to 1.0 unit. (This "mean" wind velocity should, strictly speaking, be the cubic root of the mean value of v^3.)

4.2.1 The Nelder-Mead optimization method

For Problem 1, it seems preferable to apply an optimization method that is derivative-free. We have considered a well known method of the desired type, namely the Nelder-Mead method.

The idea of the method is to keep track of $n+1$ points (simplex vertices) simultaneously, and at each step, either improve on the "worst" point along the line through the center of the others, or shrink the whole simplex towards the "best" point.

Note that for our application, a single vertex of the simplex contains the coordinates for all the turbines.

Algorithm 1 is the original Nelder-Mead method and is based on Nocedal and Wright [19]. At each step, the $n + 1$ vertices $x_1, ..., x_{n+1}$ of the current simplex are ordered so that

$$f(x_1) \geq ... \geq f(x_{n+1})$$

where $f()$ is the evaluation function. The centroid of the best n points is

Table 4.1: Test case wind directions based on Oseberg Platform A data

Direction	Given percentage	Weight
N	7%	15
NNNE		14
NNE	6%	13
NENNE		9
NE	2%	5
NEENE		4
ENE	1%	3
EENE		4
E	2%	5
EESE		6
ESE	3%	7
SEESE		10
SE	6%	13
SESSE		20
SSE	13%	27
SSSE		24
S	10%	21
SSSW		18
SSW	7%	15
SWSSW		15
SW	7%	15
SWWSW		15
WSW	7%	15
WWSW		13
W	5%	11
WWNW		10
WNW	4%	9
NWWNW		10
NW	5%	11
NWNNW		13
NNW	7%	15
NNNW		15

denoted by

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Points along the line joining $\bar{x}$ and the worst vertex x_{n+1} are denoted by

$$\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$$

and we let f_t denote $f(\bar{x}(t))$.

In order to handle constraints, we can either introduce a Lagrangian multiplier, or alter the algorithm itself. For example, if the region is rectangular with sides parallel to the coordinate axes, we can add to $f()$ the term

$$-\lambda \left(\sum_{x_i < x_{\min}} (x_{\min} - x_i) + \sum_{x_i > x_{\max}} (x_i - x_{\max}) \right.$$
$$\left. + \sum_{y_i < y_{\min}} (y_{\min} - y_i) + \sum_{y_i > y_{\max}} (y_i - y_{\max}) \right)$$

This was incorporated in the experiments with Algorithm 1 that we will return to in Section 4.2.3.

Alternatively, we can perform a truncated binary search whenever an infeasible vertex is encountered. That is, we incorporate a procedure binsearch(x, y) which calculates $z = \frac{x+y}{2}$ and replaces x by z if z is feasible, and replaces y by z otherwise. This is repeated until a minimal number of iterations is exceeded and x has changed at least once, and the new value of x is then returned. This results in Algorithm 2 where $\bar{x}(-1)$ is subject to being assigned new feasible values, and $\bar{x}(-\frac{1}{2})$ is adjusted accordingly (i.e., is set to $\frac{\bar{x}+\bar{x}(-1)}{2}$).

For both variants, it is assumed that the region is convex. Otherwise, it is not clear how the case when $\bar{x}$ is not feasible should be handled.

4.2.2 An alternative heuristic method for Problem 1

We present an alternative to Nelder-Mead - a simple heuristic optimization method in the "black box"-category. It is probably too primitive for having received any attention in the literature, and is described in Algorithm 3. As before, $f()$ denotes the evaluation function.

The idea is to try out configurations where each turbine is moved at random within a radius r (in the supremum norm, for simplicity) from its current position, update the best configuration if the new one is found to be better, and repeat with a slowly decreasing value of r.

An undesirable aspect of this algorithm is that after a considerable amount of turbines have been drawn towards the edge of the region, many

Algorithm 1 The Nelder-Mead optimization method

Select $n+1$ vertices $x_1, ..., x_{n+1}$ at random inside the region
Sort vertices so that $f(x_1) \geq ... \geq f(x_{n+1})$
while $f(x_1) - f(x_{n+1}) > \Delta$ **do**
  Compute $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
  if $f_{-1} > f(x_n)$ **then**
    if $f_{-2} > f_{-1} > f(x_1)$ **then**
      Replace x_{n+1} by $\bar{x}(-2)$
    else
      Replace x_{n+1} by $\bar{x}(-1)$
  else
    if $f_{-1/2} \geq f_{-1} > f(x_{n+1})$ **then**
      Replace x_{n+1} by $\bar{x}(-\frac{1}{2})$
    else if $f_{1/2} > f(x_{n+1}) \geq f_{-1}$ **then**
      Replace x_{n+1} by $\bar{x}(\frac{1}{2})$
    else
      Replace x_i by $\frac{x_1+x_i}{2}$ for $i = 2, 3, ..., n+1$
return x_1; $f(x_1)$

of the attempted next configurations to be tested will either be infeasible (and rejected), or away from the edge again. If the region is rectangular with sides parallel to the coordinate axes, then we can apply the variant given in Algorithm 4, in which a turbine that is initially placed outside of the region is moved to the border before the new configuration is evaluated. In other simple cases, a similar approach may be worth considering.

In this chapter we will normally refer to the method as Algorithm 3, even if Algorithm 4 applies, to avoid confusion.

It should be noted that this algorithm bears some similarity with the method of simulated annealing [20]. The idea of simulated annealing is to mimic annealing in metallurgy, which involves heating and controlled cooling of a material so that the atoms get unstuck from their initial positions and get more chances of finding configurations with lower internal energy. In each iteration of the algorithm, a neighbour to the current solution is chosen at random. The neighbour replaces the current solution with probability 1 if it is better, and with some probability that depends on the evaluations, and also of the time, otherwise. It is possible that for a suitable choice of the function that computes the probability, the iterated solutions generated by this method would behave more or less like those generated by Algorithm 3. We have not experimented on this, but left it as a possible trail to pursue in the future.

Algorithm 2 The Nelder-Mead optimization method with binary search

Select $n+1$ vertices $x_1, ..., x_{n+1}$ at random inside the region
Sort vertices so that $f(x_1) \geq ... \geq f(x_{n+1})$
while $f(x_1) - f(x_{n+1}) > \Delta$ **do**
  Compute $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
  $z \leftarrow 0$
  if $\bar{x}(-1)$ infeasible **then**
    $\bar{x}(-1) \leftarrow$ binsearch $(\bar{x}, \bar{x}(-1))$
  if $f_{-1} > f(x_n)$ **then**
    if $\bar{x}(-1)$ unchanged by binary search **and** $\bar{x}(-2)$ infeasible **then**
      $\bar{x}(-2) \leftarrow$ binsearch $(\bar{x}(-1), \bar{x}(-2))$
    if $\bar{x}(-1)$ unchanged by binary search **and** $f_{-2} > f_{-1} > f(x_1)$ **then**
      Replace x_{n+1} by $\bar{x}(-2)$
    else
      Replace x_{n+1} by $\bar{x}(-1)$
  else
    if $f_{-1/2} \geq f_{-1} > f(x_{n+1})$ **then**
      Replace x_{n+1} by $\bar{x}(-\frac{1}{2})$
    else if $f_{1/2} > f(x_{n+1}) \geq f_{-1}$ **then**
      Replace x_{n+1} by $\bar{x}(\frac{1}{2})$
    else
      Replace x_i by $\frac{x_1 + x_i}{2}$ for $i = 2, 3, ..., n+1$
return x_1; $f(x_1)$

Algorithm 3 A heuristic method for Problem 1, version 1

$C \leftarrow \{(x_1, y_1), ..., (x_n, y_n)\}$ (random feasible configuration)
$r \leftarrow r_0 \approx \frac{1}{2}$diameter$(C)$
$e \leftarrow f(C)$
while $r > r_{\min}$ **do**
  for $i = 1$ **to** N **do**
    for $j = 1$ **to** n **do**
      repeat
        $\Delta x_j \leftarrow$ random value $\in [-r, r]$
        $\Delta y_j \leftarrow$ random value $\in [-r, r]$
      until $(x_j + \Delta x_j, y_j + \Delta y_j)$ inside region
    $C' \leftarrow \{(x_1 + \Delta x_1, y_1 + \Delta y_1), ..., (x_n + \Delta x_n, y_n + \Delta y_n)\}$
    $e' \leftarrow f(C')$
    if $e' > e$ **then**
      $C \leftarrow C'$
      $e \leftarrow e'$
  $r \leftarrow r \times 0.9$
return C; e

Algorithm 4 A heuristic method for Problem 1, version 2

$C \leftarrow \{(x_1, y_1), ..., (x_n, y_n)\}$ (random feasible configuration)
$r \leftarrow r_0 \approx \frac{1}{2}\text{diameter}(C)$
$e \leftarrow f(C)$
while $r > r_{\min}$ **do**
  for $i = 1$ **to** N **do**
    for $j = 1$ **to** n **do**
      Generate $\Delta x_j, \Delta y_j \in [-r, r]$ at random
      if $x_j + \Delta x_j < x_{\min}$ **then**
        $\Delta x_j \leftarrow x_{\min} - x_j$
      if $x_j + \Delta x_j > x_{\max}$ **then**
        $\Delta x_j \leftarrow x_{\max} - x_j$
      if $y_j + \Delta y_j < y_{\min}$ **then**
        $\Delta y_j \leftarrow y_{\min} - y_j$
      if $y_j + \Delta y_j > y_{\max}$ **then**
        $\Delta y_j \leftarrow y_{\max} - y_j$
    $C' \leftarrow \{(x_1 + \Delta x_1, y_1 + \Delta y_1), ..., (x_n + \Delta x_n, y_n + \Delta y_n)\}$
    $e' \leftarrow f(C')$
    if $e' > e$ **then**
      $C \leftarrow C'$
      $e \leftarrow e'$
  $r \leftarrow r \times 0.9$
return C; e

Table 4.2: Results of heuristic algorithms for Problem 1 on simple test case

Method	Profit
Algorithm 1 (Nelder-Mead, Lagrangian with $\lambda = 0.08$)	19.3332
Algorithm 1 (Nelder-Mead, Lagrangian with $\lambda = \infty$)	19.2905
Algorithm 2 (Nelder-Mead, binary search)	19.3119
Algorithm 3 (Alternative heuristic method)	19.4285

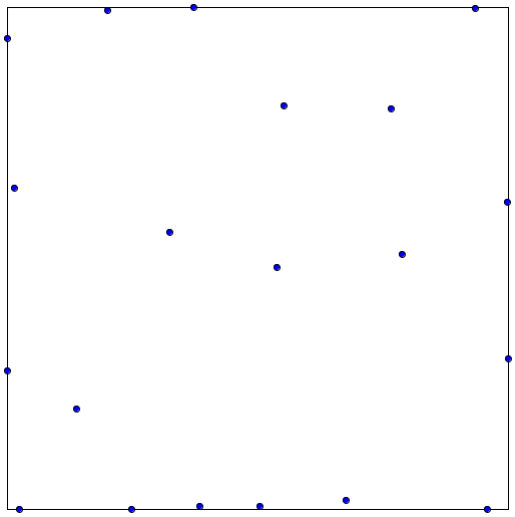

Figure 4.1: Placement generated with Algorithm 1, $\lambda = 0.08$
Profit = 19.3332

4.2.3 Problem 1: simple case experiments

We tried out Algorithms 1-3 on region R_0 with $r = r_0$ and $n = 20$. The parameter Δ was set to 10^{-7} for Nelder-Mead. Algorithm 1 was run with $\lambda = 0.08$ and $\lambda = \infty$; the former was chosen because it consistently produced feasible placements whereas smaller values tended to produce infeasible placements. The lower bound for the number of binary search iterations, T, was set to 5 in Algorithm 2, and $r_{\min}$ was set to 10^{-6} and N to 500 in Algorithm 3. The best results after 30 runs of each algorithm are given in Table 4.2. The best placement found with each algorithm is shown in Figures 4.1-4.4.

Apparently, Algorithm 3 found the best results, but it was also the slowest: about 6 minutes for one run, while the Nelder-Mead variants took around 1 minute. (This can be of course be adjusted by choosing a smaller N in Algorithm 3, as we will do in the next chapter.) In view of the performance of Algorithm 3, we also tried it out with R_0 replaced by a circle

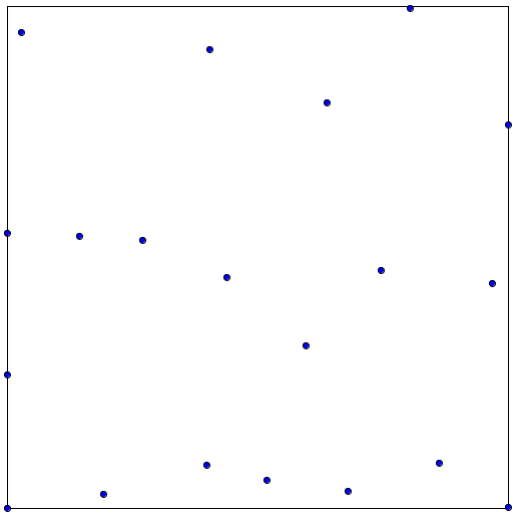

Figure 4.2: Placement of 20 turbines generated with Algorithm 1, $\lambda = \infty$
Profit = 19.2905

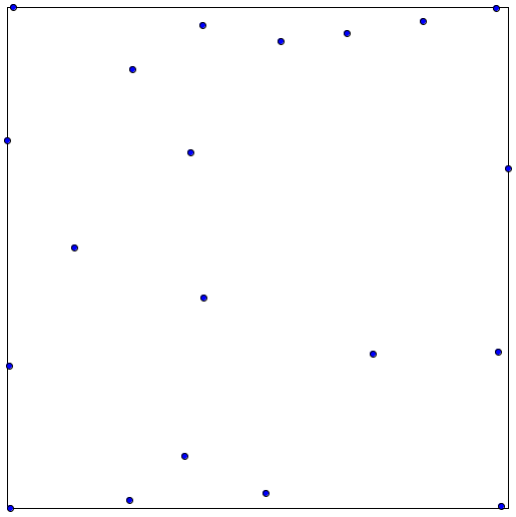

Figure 4.3: Placement of 20 turbines generated with Algorithm 2
Profit = 19.3119

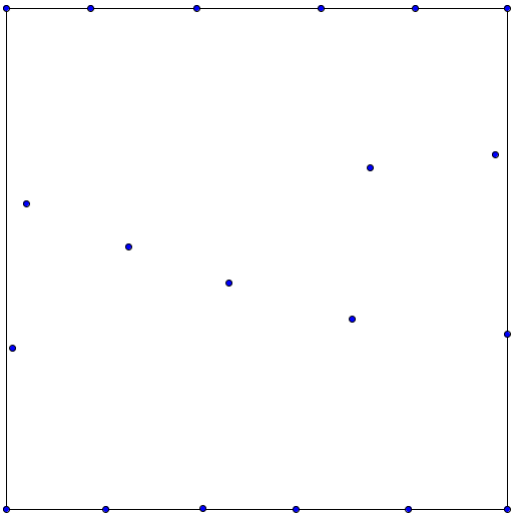

Figure 4.4: Placement of 20 turbines generated with Algorithm 3
Profit = 19.4285

of diameter 10, again taking the best out of 30 runs. This gave a profit of 19.2070 for the placement shown in Fig. 4.5. Notice that many of the turbines are placed on the perimeter, although at a couple of places they "cut corners" in order to lie on a (more or less) straight line.

Can we give some plausible reason as to why the Nelder-Mead method does not perform as well as Algorithm 3? One possible explanation is that a placement given by the mean of two or more reasonably good turbine coordinate vectors might tend to have the turbines somewhat lumped together near the centre of the region, and not be very good at all.

A similar argument applies to a metaheuristic method known as particle swarm optimization [21]. The idea of this method is to keep track of a set ("swarm") of different possible solutions ("particles"). In each iteration, the "velocity" of each particle is updated according to that particle's best position so far, and also according to the best position of any particle found so far. Then, the position of each particle is updated according to its velocity.

We have not experimented on particle swarm optimization here. In addition to the apparent weakness that *moving towards* good solutions may be inefficient for the placement of multiple turbines, we may also anticipate that the constraints involved would be even more cumbersome to handle than for the Nelder-Mead method. Like the method of simulated annealing, however, this could be something to investigate further in a possible future extension of this project.

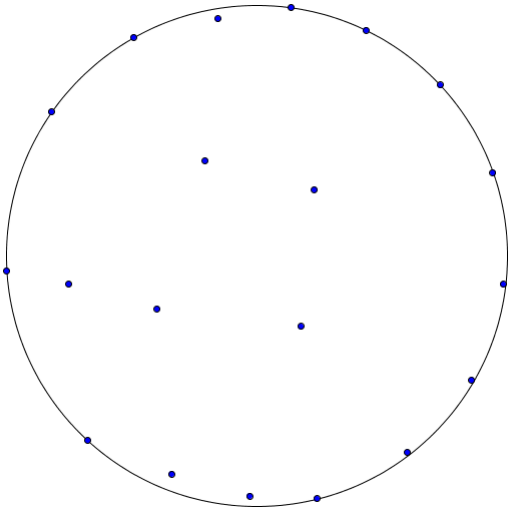

Figure 4.5: Placement of 20 turbines generated with Algorithm 3
Profit = 19.2070

Table 4.3: Location coordinates suggested by Algorithm 3 applied to a 1-dimensional case; $n = 3$ (cf. Section 2.4)

$r = 0.1$	$r = 0.01$	$r = 0.001$
$x_1 = 0.0000$	$x_1 = 0.0000$	$x_1 = 0.0000$
$x_2 = 5.0225$	$x_2 = 5.0095$	$x_2 = 5.0086$
$x_3 = 10.0000$	$x_3 = 10.0000$	$x_3 = 10.0000$

4.2.4 1-dimensional case revisited

We also revisited the 1-dimensional case. It turns out that Algorithm 3 modified for the 1-dimensional case with a small number of turbines now gives virtually the same results on every run. The results are given in Tables 4.3-4.6. We have assumed that the length of the wind farm is 10 units and that the wind direction is always in positive x-direction.

For the smaller values of n, the figures were confirmed by the optimization software BARON. The GAMS code is given in Appendix A. However, while Algorithm 3 produced the same numbers to several decimal places on repeated runs, BARON gave numbers that were slightly off since it stops refining the results once the upper and lower bounds for the objective function are sufficiently close, and the limits can not be set smaller than 10^{-9}. (The initial wind speed in set to 1000 in the GAMS code, which gave higher accuracy than lower values.)

We mentioned at the end of Section 2.4 that the optimal placement is

Table 4.4: Location coordinates suggested by Algorithm 3 applied to a 1-dimensional case; $n = 4$

$r = 0.1$	$r = 0.01$	$r = 0.001$
$x_1 = 0.0000$	$x_1 = 0.0000$	$x_1 = 0.0000$
$x_2 = 3.3788$	$x_2 = 3.3396$	$x_2 = 3.3385$
$x_3 = 6.6663$	$x_3 = 6.6749$	$x_3 = 6.6741$
$x_4 = 10.0000$	$x_4 = 10.0000$	$x_4 = 10.0000$

Table 4.5: Location coordinates suggested by Algorithm 3 applied to a 1-dimensional case; $n = 5$

$r = 0.1$	$r = 0.01$	$r = 0.001$
$x_1 = 0.0000$	$x_1 = 0.0000$	$x_1 = 0.0000$
$x_2 = 2.5655$	$x_2 = 2.5049$	$x_2 = 2.5035$
$x_3 = 5.0232$	$x_3 = 5.0063$	$x_3 = 5.0052$
$x_4 = 7.4795$	$x_4 = 7.5068$	$x_4 = 7.5061$
$x_5 = 10.0000$	$x_5 = 10.0000$	$x_5 = 10.0000$

likely almost, but not quite, equidistant as $u \to 0$ regardless of what ν-norm is used to model multiple wakes. These experiments appear to confirm this for $\nu = 3$ to some extent.

Thus, when we consider Problem 2, it makes sense to base the set of slots on regular grids. Note that in the third column in each of these experiments, all the turbines have moved slightly in the wind direction away from their spot in an equidistant distribution. With shifting wind directions, this effect would most likely be more or less eliminated - although not completely: an example is given in Table 4.7.

4.2.5 Heuristics for Problem 2

Algorithm 5 gives an overview of a natural approach to Problem 2, and a more detailed version is given by Algorithm 6. The parameter $\vec{\alpha}$ denotes the incidence vector of occupied slots among all the n slots. The condition $\text{distance}(i, j) \leq \sqrt{5}$ assumes a regular square grid structure on the slots and says essentially that only movements up to a knight's move away in the grid are considered. Alternative conditions to limit the number of possible moves are of course possible.

Table 4.6: Location coordinates suggested by Algorithm 3 applied to a 1-dimensional case; $n = 21$

$r = 0.1$	$r = 0.01$	$r = 0.001$
$x_1 = 0.0000$	$x_1 = 0.0000$	$x_1 = 0.0000$
$x_2 = 0.6892$	$x_2 = 0.5066$	$x_2 = 0.5006$
$x_3 = 1.1749$	$x_3 = 1.0063$	$x_3 = 1.0007$
$x_4 = 1.6621$	$x_4 = 1.5059$	$x_4 = 1.5008$
$x_5 = 2.1493$	$x_5 = 2.0055$	$x_5 = 2.0009$
$x_6 = 2.6365$	$x_6 = 2.5050$	$x_6 = 2.5009$
$x_7 = 3.1236$	$x_7 = 3.0046$	$x_7 = 3.0010$
$x_8 = 3.6106$	$x_8 = 3.5041$	$x_8 = 3.5010$
$x_9 = 4.0975$	$x_9 = 4.0037$	$x_9 = 4.0011$
$x_{10} = 4.5844$	$x_{10} = 4.5032$	$x_{10} = 4.5011$
$x_{11} = 5.0712$	$x_{11} = 5.0028$	$x_{11} = 5.0012$
$x_{12} = 5.5579$	$x_{12} = 5.5023$	$x_{12} = 5.5012$
$x_{13} = 6.0446$	$x_{13} = 6.0019$	$x_{13} = 6.0013$
$x_{14} = 6.5312$	$x_{14} = 6.5014$	$x_{14} = 6.5014$
$x_{15} = 7.0175$	$x_{15} = 7.0010$	$x_{15} = 7.0014$
$x_{16} = 7.5037$	$x_{16} = 7.5005$	$x_{16} = 7.5015$
$x_{17} = 7.9893$	$x_{17} = 8.0000$	$x_{17} = 8.0015$
$x_{18} = 8.4741$	$x_{18} = 8.4996$	$x_{18} = 8.5016$
$x_{19} = 8.9574$	$x_{19} = 8.9991$	$x_{19} = 9.0016$
$x_{20} = 9.4374$	$x_{20} = 9.4984$	$x_{20} = 9.5015$
$x_{21} = 10.0000$	$x_{21} = 10.0000$	$x_{21} = 10.0000$

Table 4.7: Location coordinates suggested by Algorithm 3 applied to a 1-dimensional case; $n = 5$ and wind uniformly distributed in both directions

$r = 0.1$	$r = 0.01$	$r = 0.001$
$x_1 = 0.00000$	$x_1 = 0.00000$	$x_1 = 0.00000$
$x_2 = 2.54308$	$x_2 = 2.49905$	$x_2 = 2.49868$
$x_3 = 5.00000$	$x_3 = 5.00000$	$x_3 = 5.00000$
$x_4 = 7.45692$	$x_4 = 7.50095$	$x_4 = 7.50132$
$x_5 = 10.00000$	$x_5 = 10.00000$	$x_5 = 10.00000$

Algorithm 5 Overview of a heuristic method for Problem 2

Start with no installed turbines
repeat
  Install turbines in most profitable vacant slot
until no improvement possible
repeat
  Select most profitable action from:
  – installing turbine in vacant slot
  – deinstalling existing turbine
  – moving turbine to nearby vacant slot
until no improvement possible
return best placement; net profit

4.2.6 Problem 2: simple case experiments

For a square-shaped, continuous region like R_0, it is possible to incorporate the heuristics for Problem 2 by way of Algorithm 7.

We ran an experiment with $r = r_0$, cost per turbine $c = 0.85$, $k_{\min} = 10$, $k_{\max} = 30$. The results are given in Table 4.8.

As we can see, the net profit peaked at $k = 17$.

We would like to compare the heuristic methods on Problem 1 and Problem 2. Algorithm 3 does not perform particularly well with more than 20 turbines, since it is rather unlikely that any given next step does not contain some entries that point in the opposite direction of the the nearest local optimum. We attempted to tackle this problem by first placing 10 turbines using Algorithm 3 and fixing them, then placing 10 additional ones and so on up to 50. For this experiment, we also incorporated the cost $c = 0.85$ per turbine. The results are given in Table 4.9. (There was only one run since it took more than one hour.)

The conclusion is that this modified version of Algorithm 3 does not work very well, at least when compared with Algorithm 7.

Next, we tried a Lagrangian relaxation of the cardinality contraint; i.e., we adjusted the cost c in order for Algorithm 7 to find a placement comparable to our earlier experiments with 20 turbines. We did not succeed in making it arrive at exactly 20 turbines: Values from 0.932 to 0.938 all seemed to give 22 turbines, while values from 0.939 and upwards gave fewer than 20 turbines.

In fact, all values of c from 0.932 to 0.936 that were tested gave the exact same placement with profit 21.3027 (costs not included), while 0.937 and 0.938 gave a different placement with profit 21.3090. Both placements had $k = 19$ and are shown in Figures 4.6 and 4.7.

Then we ran Algorithm 3 again, with $n = 22$. Out of the 30 runs, only one run produced a profit above 21.3. It is shown in Fig. 4.8. The profit is

Algorithm 6 A heuristic method for Problem 2

$\vec{\alpha} \leftarrow \vec{0}$, $e \leftarrow f(\vec{\alpha}) = 0$
repeat
  $e_1 \leftarrow 0$
  for $i = 1$ **to** n **do**
    if $\alpha(i) = 0$ **then**
      $\alpha(i) \leftarrow 1$
      $e_2 \leftarrow f(\vec{\alpha})$
      if $e_2 > e_1$ **then**
        $u \leftarrow i$
        $e_1 \leftarrow e_2$
      $\alpha(i) \leftarrow 0$
  if $e_1 > e$ **then**
    $\alpha(u) \leftarrow 1$
    $e \leftarrow e_1$
until $e_1 < e$ (no improvement found)
repeat
  $e_1 \leftarrow 0, e_2 \leftarrow 0$
  for $i = 1$ **to** n **do**
    $\alpha(i) \leftarrow 1 - \alpha(i)$
    $e_3 \leftarrow f(\vec{\alpha})$
    if $e_3 > e_1$ **then**
      $u \leftarrow i$
      $e_1 \leftarrow e_3$
    $\alpha(i) \leftarrow 1 - \alpha(i)$ (install or deinstall)
  $v \leftarrow 0, w \leftarrow 0$
  for $i = 1$ **to** $n - 1$ **do**
    for $j = i + 1$ **to** n **do**
      if $\alpha(i) + \alpha(j) = 1$ **and** distance$(i, j) \leq \sqrt{5}$ **then**
        $\alpha(i) \leftarrow 1 - \alpha(i)$
        $\alpha(j) \leftarrow 1 - \alpha(j)$ (swap i and j)
        $e_3 \leftarrow f(\vec{\alpha})$
        if $e_3 > e_2$ **then**
          $v \leftarrow i, w \leftarrow j$
          $e_2 \leftarrow e_3$
        $\alpha(i) \leftarrow 1 - \alpha(i)$
        $\alpha(j) \leftarrow 1 - \alpha(j)$
  if $e_2 > e_1$ **and** $e_2 > e$ **then**
    $\alpha(v) \leftarrow 1 - \alpha(v)$
    $\alpha(w) \leftarrow 1 - \alpha(w)$
    $e \leftarrow e_2$
  if $e_1 > e$ **and** $e_1 > e_2$ **then**
    $\alpha(u) \leftarrow 1 - \alpha(u)$
    $e \leftarrow e_1$
until $e_1 < e$ **and** $e_2 < e$
return $\vec{\alpha}$; e

Algorithm 7 Heuristics for Problem 2 on a square-shaped continuous region

for $k = k_{\min}$ **to** $k_{\max}$ **do**
  $S \leftarrow \left\{ \left(\frac{i}{k-1} \times l, \frac{j}{k-1} \times l \right) : 0 \leq i, j \leq k-1 \right\}$
  Find net profit e of S with Algorithm 6
  return n; no. of turbines; e

Table 4.8: Experimental results of Algorithm 7

k	No. of turbines	Net profit	Running time
10	39	3.1990	26 s
11	38	3.3582	53 s
12	41	3.4043	50 s
13	43	3.4651	1 min 55 s
14	44	3.5015	2 min 48 s
15	44	3.5411	2 min 36 s
16	46	3.5731	4 min 50 s
17	48	3.5950	5 min 53 s
18	50	3.5651	7 min 56 s
19	52	3.5455	9 min 45 s
20	51	3.5023	17 min 9 s
21	44	3.5040	11 min 41 s
22	42	3.3566	10 min 42 s
23	41	3.4220	14 min 40 s
24	44	3.3799	29 min 42 s
25	43	3.3783	25 min 22 s
26	43	3.4292	43 min 28 s
27	42	3.4018	36 min 34 s
28	42	3.3692	49 min 22 s
29	41	3.3910	57 min 51 s
30	42	3.4238	70 min 48 s

Table 4.9: Experimental results of Algorithm 3 with increasing number of turbines

No. of turbines	Net profit
10	1.3941
20	2.3939
30	2.9494
40	3.2376
50	3.1236

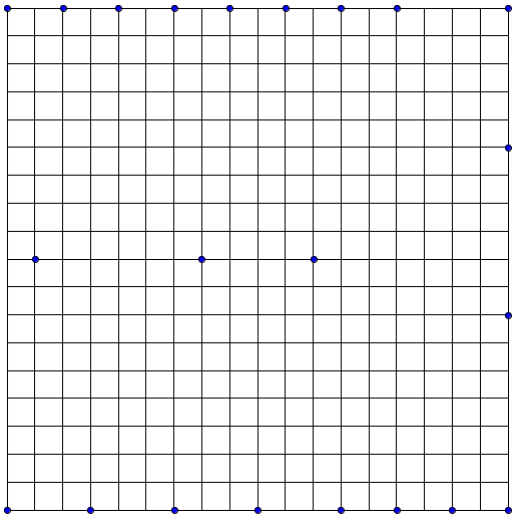

Figure 4.6: Placement generated with Algorithm 7 for $c = 0.934$
Profit = 21.3027

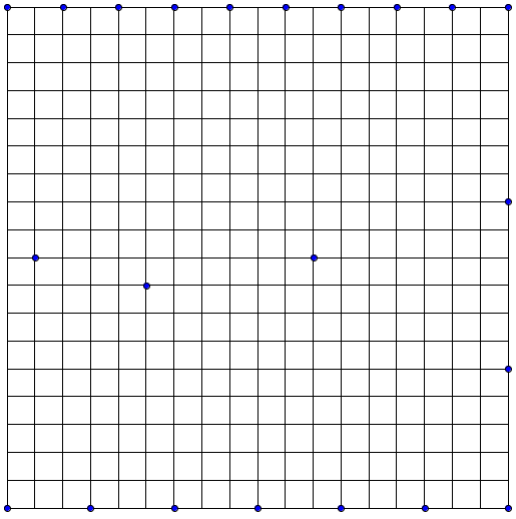

Figure 4.7: Placement generated with Algorithm 7 for $c = 0.938$
Profit = 21.3090

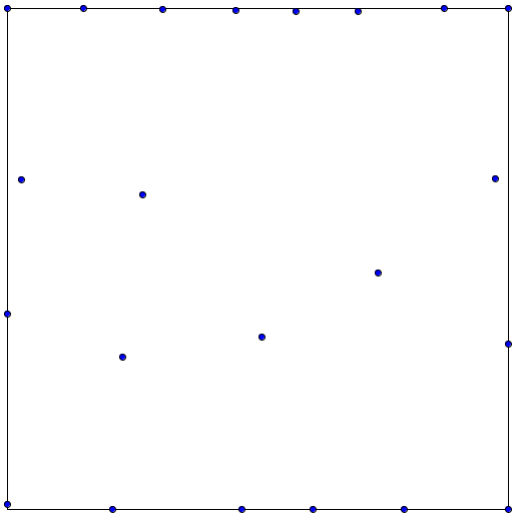

Figure 4.8: Placement of 22 turbines generated with Algorithm 3
Profit = 21.3062

21.3062, between the two results found with Algorithm 7.

We believe these experiments favour Algorithm 7, since only one run of Algorithm 3 out of 30 produced a profit on level with those of Algorithm 7. This matter will be investigated further in Chapter 5.

Returning to the original experiment of this subsection, we observed that many of the turbines in the placement found with $k = 17$ were placed near the North and South edges. Therefore, we added slots along those edges only (i.e., one extra slot midways between any two consecutive original ones). But the result of this was that the net profit *dropped* to 3.5257. We concluded that the idea of extra slots along the edges was not worth pursuing.

Chapter 5

Experiments

In the last chapter, we found mathematical models that should be possible to implement with solver software, and we tried out various heuristic methods on a rather simple case with a square-shaped wind farm. Our next task is to bring the theory to life, so to speak; to actually do the implementations of the mathematical models and to try out the heuristic methods on farms of a more general shape.

One way of representing the area occupied by a wind farm is to provide a set of linear constraints. This is suitable for the heuristic methods we considered for Problem 1. In fact, both variants of Nelder-Mead as well as the possible improvement of Algorithm 3, given by Algorithm 4, are easily adapted to this situation. The most prominent disadvantage is that any region thus defined is necessarily convex, but as we have already mentioned, applying Nelder-Mead to non-convex regions would be problematic in any case (confer Section 4.2.1).

A more general way to represent the area is by a binary matrix, in which each entry corresponds to a "pixel" in the farm that may or may not receive a turbine. Algorithm 3 can easily be modified in a way that would work for this representation, as seen in Algorithm 8. Algorithm 7 can run in the form of Algorithm 9, which tries out various ways to place a regular grid. For each row in the grid, the rightmost and the leftmost positive matrix entry is added to the set of possible locations (if not already included), as well as the uppermost and lowermost positive entry for each grid column. This is done in order to make best possible use of the available area. In order to justify this part, we can also run the algorithm without adding the endpoints and compare the results.

It is not hard to modify Algorithm 9 so that it would handle linear constraints instead of binary matrices. However, the emphasis is put on comparing the various algorithms for Problem 1 with each other, to compare Algorithm 3 with Algorithm 9, and to do the latter with the highest degree of generality. Running Algorithm 9 on wind farms given by linear constraints

is therefore somewhat redundant.

Algorithm 8 A discrete version of Algorithm 3

$C \leftarrow \{(x_1, y_1), ..., (x_n, y_n)\}$ random feasible configuration
$r \leftarrow r_0 \approx \frac{1}{2}\text{diameter}(C)$
$e \leftarrow f(C)$
while $r > 1$ **do**
  for $i = 1$ **to** N **do**
    for $j = 1$ **to** n **do**
      repeat
        $\Delta x_j \leftarrow$ random value $\in [-r, r]$
        $\Delta y_j \leftarrow$ random value $\in [-r, r]$
      until $(x_j + \Delta x_j, y_j + \Delta y_j)$ inside region and
      distinct from $(x_{j'} + \Delta x_{j'}, y_{j'} + \Delta y_{j'})$ for all $j' < j$
    $C' \leftarrow \{(x_1 + \Delta x_1, y_1 + \Delta y_1), ..., (x_n + \Delta x_n, y_n + \Delta y_n)\}$
    $e' \leftarrow f(C')$
    if $e' > e$ **then**
      $C \leftarrow C'$
      $e \leftarrow e'$
  $r \leftarrow [r \times 0.9]$
return C; e

5.1 Overview of experiments

Appendix B contains GAMS code written for the purpose of solving instances of Problem 1. We will use the region R_0 with $r = r_0$, and consider a rather simple case with 24 wind directions and only two turbines. (We believe it would not be interesting to consider a smaller number of wind directions since the turbines might be lined up in a direction with only an insignificant deficit.)

Problem 2 should be computationally simpler. Therefore, we will try nine (canonical) locations in R_0, $c = 0.5$ and eight wind directions. The code is given in Appendix C, and the results are discussed in Section 5.3.1.

Moving on to the heuristic methods, we introduce

$$\Omega = \sum_{w<m} p(w)v_0(w)^3$$

which is the mean profit of a single turbine, given the probability $p(w)$ and mean wind velocity $v_0(w)^3$ for each wind direction w. (Again (confer the first part of Section 4.2), this should be the cubic root of the mean value of v^3.)

Algorithm 9 A heuristic algorithm for handling wind farm of arbitrary shape, represented by binary $m \times n$ matrix

Read data from file
for $k = k_{\min}$ **to** $k_{\max}$ **do**
  $e_{\max} \leftarrow 0$
  for $i = 0$ **to** $k - 1$ **do**
    for $j = 0$ **to** $k - 1$ **do**

      (Add points from regular grid)
      $S \leftarrow \emptyset$
      for $u = 0$ **to** $m - 1$ **do**
        if $u \equiv i \mod k$ **then**
          for $v = 0$ **to** $n - 1$ **do**
            if $v \equiv j \mod k$ **and** $A(u, v) = 1$ **then**
              $S \leftarrow S \cup \{(u, v)\}$

      (Add column bottom and top points)
      for $u = 0$ **to** $m - 1$ **do**
        if $u \equiv i \mod k$ **then**
          $v_1 \leftarrow \min\{v : A(u, v) = 1\} \cup \{n - 1\}$
          if $A(u, v_1) = 1$ **then**
            if $v_1 \not\equiv j \mod k$ **then**
              $S \leftarrow S \cup \{(u, v_1)\}$
            $v_2 \leftarrow \max\{v : A(u, v) = 1\}$
            if $v_2 > v_1$ **and** $v_2 \not\equiv j \mod k$ **then**
              $S \leftarrow S \cup \{(u, v_2)\}$

      (Add leftmost and rightmost row points)
      for $v = 0$ **to** $n - 1$ **do**
        if $v \equiv j \mod k$ **then**
          $u : 1 \leftarrow \min\{u : A(u, v) = 1\} \cup \{n - 1\}$
          if $A(u_1, v) = 1$ **then**
            if $u_1 \not\equiv i \mod k$ **then**
              $S \leftarrow S \cup \{(u_1, v)\}$
            $u_2 \leftarrow \max\{u : A(u, v) = 1\}$
            if $u_2 > u_1$ **and** $u_2 \not\equiv i \mod k$ **then**
              $S \leftarrow S \cup \{(u_2, v)\}$

      Find net profit e of S with Algorithm 6
      if $e > e_{\max}$ **then**
        Store configuration S
        $e_{\max} \leftarrow e$
  return mask width k, best configuration found, $e_{\max}$

We have chosen to run Algorithms 1, 2 and 4 for 50 test cases given by linear constraints (see Section 5.2) with $n = 20$ turbines, rotor radius = 1.5, and record the best of 10 runs in each instance.

For the Nelder-Mead algorithms, we let $\Delta = 10^{-7}\Omega$, and the Lagrangian coefficient in Algorithm 1 is set to 0.8Ω. For Algorithm 4, r_0 is set to 50.0, and $r_{\min}$ is set to 10^{-5}. The parameter N is set to 100 in order to get a running time comparable to the Nelder-Mead variants.

Next we are going to run Algorithm 9, with and without row and column endpoints, for 50 test cases given by binary matrices (see Section 5.2). The purpose of this is to prepare for a comparison with Algorithm 8. With different wind conditions generated for each case and a fixed cost, inevitably either no turbines will be suggested installed in some of the cases, or a rather large number will be suggested installed in other cases. For our purpose, the cost c is set to 60.0, because it was observed that with this value, Ω would lie in the interval $[c, \frac{4}{3}c]$ in relatively many cases. We believe this is suitable, based on the results of the experiments from the previous chapter. The rotor radius is set to 1.0, and a pixel is set to 1.0×1.0.

For the cases in which $\Omega > c$, we will run Algorithm 8 with n determined by the output from Algorithm 9. The search radius r starts at 50 and terminates at 1. This yields fewer different values of r than in the continuous case, and therefore, N has been increased to 1000.

Results from all the heuristic methods are given in Section 5.3.2.

5.2 Overview of cases

We generated 50 cases based on linear constraints in the manner given in Algorithm 10. The wind data (contained in $\{p(i)\}_{0 \leq i < m}$ and $\{v(i)\}_{0 \leq i < m}$) are generated so that the difference between consecutive values is always relatively small, when we take into consideration that $m - 1$ and 0 are consecutive indices in this setting. For example, before the last scaling of

the $p(i)$, we have

$$\begin{aligned}
p(0) - p(m-1) =& p(0) - p(m-2) - \Delta p(m-1) + \frac{\sum_{j=0}^{m-1} \Delta p(j)}{m} \\
=& p(0) - p(m-3) - \Delta p(m-2) - \Delta p(m-1) + 2\frac{\sum_{j=0}^{m-1} \Delta p(j)}{m} \\
=& (...) \\
=& p(0) - p(0) - \Delta p(1) - \Delta p(2) - ... - \Delta p(m-1) \\
& + (m-1)\frac{\sum_{j=0}^{m-1} \Delta p(j)}{m} \\
=& \Delta p(0) - \frac{\sum_{j=0}^{m-1} \Delta p(j)}{m}
\end{aligned}$$

An upper bound on the absolute value of the difference between consecutive values is therefore given by

$$\max\{\Delta p(i)\} + \left| \frac{\sum_{j=0}^{m-1} \Delta p(j)}{m} \right| \leq 1 + 1 = 2$$

A similar result holds for the $v(i)$.

The cases can be found on the website [22], and are labelled lc01.txt, ..., lc50.txt.

The same website also contains 50 cases based on binary matrices (bm01.txt, ..., bm50.txt), and the procedure for generating them is given in Algorithm 11.

5.3 Results

5.3.1 Exact solutions to Problem 1 and Problem 2

The programs in Appendices B and C were run with the solver software BARON.

Unfortunately, even in our Problem 1 case of small size we did not get useful results. The preprocessing apparently found a feasible solution with profit just slightly greater than the trivial lower bound of v_0^3, and the upper bound was (trivially) $2v_0^3$. The program was set to run for 90 minutes. After

Algorithm 10 Procedure for generating wind farm test cases based on linear constraints

(Generate linear constraints)
Start with no constraints
repeat
  $\theta \leftarrow$ random value $\in [0, 2\pi)$
  $b \leftarrow$ random value $\in [20, 100]$
  Add linear constraint: $(\cos\theta)x + (\sin\theta)y \leq b$
until all integer points (x, y) with $\max\{|x|, |y|\} = 100$ infeasible
Remove redundant constraints

(Generate wind direction probability distribution)
repeat
  $p(0) \leftarrow$ random value $\in [0, 5.0]$
  for $i = 0$ **to** $m - 1$ **do**
    $\Delta p(i) \leftarrow$ random value $\in [-1, 1]$
  $s \leftarrow p(0)$
  for $i = 1$ **to** $m - 1$ **do**
    $p(i) \leftarrow p(i-1) + \Delta p(i) - \frac{\sum_{j=0}^{m-1} \Delta p(j)}{m}$
    if $p(i) > 0$ **then**
      $s \leftarrow s + p(i)$
  for $i = 0$ **to** $m - 1$ **do**
    if $p(i) > 0$ **then**
      $p(i) \leftarrow p(i)/s$
    else
      $p(i) \leftarrow 0$
until at least $\frac{3m}{4}$ positive values

(Generate mean wind velocities)
$v(0) \leftarrow$ random value $\in [0, 5.0]$
for $i = 0$ **to** $m - 1$ **do**
  $\Delta v(i) \leftarrow$ random value $\in [-0.5, 0.5]$
for $i = 1$ **to** $m - 1$ **do**
  $v(i) \leftarrow v(i-1) + \Delta v(i) - \frac{\sum_{j=0}^{m-1} \Delta v(j)}{m}$
for $i = 1$ **to** $m - 1$ **do**
  if $v(i) < 0$ **then**
    $v(i) \leftarrow$ random value $\in [0, 1]$

Algorithm 11 Procedure for generating wind farm test cases based on binary matrices

(Generate binary matrices)
$A \leftarrow 0$ (100×100 matrix)
for $k = 1$ **to** 4 **do**
  $x_k \leftarrow$ random integer value $\in [26, 75]$
  $y_k \leftarrow$ random integer value $\in [26, 75]$
for $i = 1$ **to** 100 **do**
  for $j = 1$ **to** 100 **do**
    for $k = 1$ **to** 4 **do**
      $z_k \leftarrow$ random value $\in [15^2, 25^2]$
      $d_k \leftarrow (i - x_k)^2 + (j - y_k)^2$
      if $d_k < z_k$ **then**
        $A_{ij} \leftarrow 1$

Generate wind direction probability distribution
and mean wind velocities: Confer Algorithm 10

48 minutes, the number of open nodes had passed 11,000, and the program reported shortage of memory. The bounds had not improved visibly since the first iteration at this point, and the program was terminated.

When we ran the Problem 2 code, a lower bound could be found by taking $\alpha(k) = 1$ for eack k. This yields a value near $8.6457 \cdot 10^6$ for the objective function, and is optimal, since a trivial bound when one turbine is removed, is given by $(9-1)(v_0^3) = 8.0 \cdot 10^6$. The program found this lower bound already in the preprocessing. The upper bound, however, decreased rather slowly. The earliest estimate was the trivial upper bound of $9.0 \cdot 10^6$. After 5000 iterations (and almost 11 minutes running time), it had dropped to approximately $8.86 \cdot 10^6$, and then to approximately $8.77 \cdot 10^6$ after 20,000 iterations (almost 40 minutes running time). The program terminated after 90 minutes running time, at which point the upper bound was still above $8.72 \cdot 10^6$.

5.3.2 Results of heuristic methods

Algorithm 1 occasionally produced infeasible solutions. Even though the violations of the linear constraints were mostly very small, they were so rare that that they were simply discarded.

An upper bound for the total profit can be found by dismissing the wake effect completely, and is given by $n\Omega$. The results of the experiments are seen in Table 5.1. The columns with heading "Rel. eval." contain evaluations relative to this upper bound, and the columns with heading "Time" contain the mean running time for the 10 runs.

For the cases based on binary matrices, Algorithm 9 was run with one single mask width at a time from 7 down to 3, in order that the running times could be compared. Next, we repeated the experiments with Algorithm 9 replaced by a version in which row endpoints and column endpoints were not added to the grid. In the cases 1, 3, 4, 5, 6, 7, 11, 12, 14, 15, 16, 17, 19, 21, 22, 23, 24, 25, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 46 and 48, the algorithm suggested that no turbine be installed, due to the fact that $\Omega \leq c$. The running time of Algorithm 9 in these cases varied from 3.3 to 8 seconds for mask width 7, 3.5 to 9 seconds for mask width 6, 4.4 to 11 seconds for mask width 5, 5.4 to 16 seconds for mask width 4, and from 8 to 24 seconds for mask width 3. When endpoints were not added, the running time varied from 1.6 to 4.2 seconds for mask width 7, 2.1 to 5.1 seconds for mask width 6, 2.6 to 6.7 seconds for mask width 5, 3.3 to 10 seconds for mask width 5, and from 5.0 to 18 seconds for mask width 3. The results from the remaining cases are given in Table 5.2 and Table 5.3. The first line of each entry contains the best net profit and the corresponding number of turbines, and the second line contains the running time.

Table 5.1: Best relative evaluation in 10 runs

	Algorithm 1		Algorithm 2		Algorithm 4	
Case	Rel. eval.	Time	Rel. eval.	Time	Rel. eval.	Time
lc01.txt	0.9410	63.7s	0.9442	65.5s	0.9534	75.4s
lc02.txt	0.9392	60.1s	0.9369	68.9s	0.9494	73.7s
lc03.txt	0.9332	53.8s	0.9362	88.4s	0.9447	74.2s
lc04.txt	0.9275	51.6s	0.9188	73.0s	0.9379	76.7s
lc05.txt	0.9522	59.7s	0.9530	66.3s	0.9591	71.8s
lc06.txt	0.9488	57.9s	0.9505	76.0s	0.9578	72.7s
lc07.txt	0.9541	72.9s	0.9545	81.2s	0.9640	77.7s
lc08.txt	0.9378	48.1s	0.9360	48.6s	0.9477	51.5s
lc09.txt	0.9400	50.5s	0.9414	60.8s	0.9515	66.3s
lc10.txt	0.9120	59.0s	0.9165	72.9s	0.9333	76.3s
lc11.txt	0.9206	43.4s	0.9210	45.2s	0.9360	57.3s
lc12.txt	0.9001	71.3s	0.8995	61.9s	0.9165	73.7s
lc13.txt	0.9172	42.2s	0.9116	48.8s	0.9291	58.3s
lc14.txt	0.9130	60.2s	0.9098	77.4s	0.9277	78.2s
lc15.txt	0.9242	51.1s	0.9236	44.0s	0.9383	62.8s
lc16.txt	0.9265	59.8s	0.9255	80.5s	0.9361	78.5s
lc17.txt	0.9515	66.1s	0.9467	75.9s	0.9586	80.8s
lc18.txt	0.9532	47.4s	0.9555	47.5s	0.9683	56.2s
lc19.txt	0.9449	54.4s	0.9497	59.3s	0.9640	67.6s
lc20.txt	0.9414	59.5s	0.9392	75.7s	0.9490	78.0s
lc21.txt	0.8627	52.3s	0.8767	63.0s	0.8955	64.7s

(To be continued on next page)

Table 5.1: Best relative evaluation in 10 runs (continued)

	Algorithm 1		Algorithm 2		Algorithm 4	
Case	Rel. eval.	Time	Rel. eval.	Time	Rel. eval.	Time
lc22.txt	0.9210	77.4s	0.9262	71.3s	0.9489	80.0s
lc23.txt	0.9432	49.6s	0.9455	59.1s	0.9538	61.4s
lc24.txt	0.9301	64.8s	0.9320	64.9s	0.9408	75.6s
lc25.txt	0.9222	52.2s	0.9217	64.3s	0.9335	73.2s
lc26.txt	0.9405	58.0s	0.9387	78.4s	0.9513	75.8s
lc27.txt	0.9092	59.7s	0.9107	58.6s	0.9316	69.1s
lc28.txt	0.9017	52.1s	0.9002	52.7s	0.9163	68.3s
lc29.txt	0.9581	67.4s	0.9599	66.9s	0.9671	69.8s
lc30.txt	0.9106	62.8s	0.9196	67.5s	0.9347	73.6s
lc31.txt	0.9250	59.9s	0.9305	77.4s	0.9440	78.8s
lc32.txt	0.9388	50.9s	0.9381	61.8s	0.9512	67.5s
lc33.txt	0.8318	57.4s	0.8426	75.2s	0.8692	71.9s
lc34.txt	0.9511	53.6s	0.9474	59.4s	0.9663	62.8s
lc35.txt	0.8848	56.2s	0.8911	61.6s	0.9108	66.6s
lc36.txt	0.9380	66.9s	0.9413	78.0s	0.9528	74.5s
lc37.txt	0.9120	63.7s	0.9193	71.2s	0.9308	81.0s
lc38.txt	0.9346	60.5s	0.9387	70.4s	0.9550	73.4s
lc39.txt	0.8691	47.7s	0.8647	69.6s	0.8853	68.7s
lc40.txt	0.9209	56.3s	0.9255	71.8s	0.9373	73.5s
lc41.txt	0.9153	48.0s	0.9153	57.0s	0.9259	60.6s
lc42.txt	0.9303	57.5s	0.9279	62.4s	0.9427	75.0s
lc43.txt	0.8957	46.1s	0.8953	68.1s	0.9120	66.7s
lc44.txt	0.9393	64.7s	0.9397	73.8s	0.9506	72.6s
lc45.txt	0.9222	61.5s	0.9230	80.4s	0.9348	76.7s
lc46.txt	0.9346	60.1s	0.9339	80.4s	0.9532	78.3s
lc47.txt	0.9268	48.2s	0.9246	65.2s	0.9472	63.1s
lc48.txt	0.9277	57.3s	0.9245	53.2s	0.9448	63.5s
lc49.txt	0.9189	61.0s	0.9167	65.5s	0.9319	71.7s
lc50.txt	0.8757	50.7s	0.8783	65.8s	0.8970	66.8s
Mean	0.9234	57.1s	0.9244	66.7s	0.9388	70.7s

Finally, the results of the last experiments are given in Table 5.4. The column with heading "n" contains the number of turbines, based on the results given in Tables 5.2 and 5.3. The column with heading "Evaluation" contains the best profit found in 10 runs of Algorithm 8. The next column with heading "Net profit" contains the differences between the evaluations in the third column and the costs nc with the values for n given in the second column (recall that $c = 60.0$). These values can then be compared with the entries in Tables 5.2 and 5.3. The last column contains the total running

Table 5.2: Experimental results of Algorithm 9

Case	Mask w. 7	Mask w. 6	Mask w. 5	Mask w. 4	Mask w. 3
bm02.txt	328.6 (32)	338.3 (33)	338.1 (34)	338.2 (35)	352.3 (35)
	7m48s	9m16s	10m5s	16m30s	34m55s
bm08.txt	105.8 (18)	106.8 (19)	107.1 (19)	107.0 (19)	107.9 (19)
	2m37s	3m5s	5m9s	8m52s	14m4s
bm09.txt	1214 (66)	1225 (62)	1242 (64)	1286 (68)	1304 (69)
	36m9s	43m10s	50m51s	72m25s	135m34s
bm10.txt	34.98 (11)	35.66 (11)	35.59 (11)	35.81 (11)	34.63 (10)
	37s	40s	52s	1m35s	4m1s
bm13.txt	514.0 (37)	524.8 (40)	539.0 (41)	540.5 (43)	549.8 (41)
	10m29s	12m55s	18m22s	29m47s	48m9s
bm18.txt	69.00 (15)	69.45 (15)	70.24 (15)	69.30 (16)	70.24 (15)
	1m42s	2m18s	3m12s	6m38s	13m16s
bm20.txt	6208 (119)	6760 (151)	7191 (203)	7343 (216)	7523 (217)
	83m48s	163m28s	399m16s	1091m21s	1889m57s
bm26.txt	0.1926 (2)	0.1903 (2)	0.1958 (2)	0.1903 (2)	0.1846 (2)
	8s	8s	12s	16s	35s
bm29.txt	4624 (103)	4918 (131)	5079 (165)	5222 (156)	5293 (167)
	49m42s	95m7s	230m12s	515m19s	763m37s
bm43.txt	53.14 (12)	52.80 (12)	53.57 (12)	53.40 (12)	54.97 (13)
	46s	49s	1m6s	2m4s	4m11s
bm44.txt	713.0 (46)	729.6 (47)	739.5 (47)	750.8 (49)	762.1 (49)
	15m3s	17m46s	23m35s	35m21s	62m14s
bm45.txt	14.36 (8)	14.66 (8)	14.43 (8)	15.14 (8)	14.75 (8)
	36s	43s	1m7s	2m11s	4m25s
bm47.txt	2298 (84)	2337 (91)	2396 (89)	2519 (94)	2588 (99)
	33m58s	56m53s	81m30s	129m1s	221m51s
bm49.txt	324.5 (35)	328.2 (34)	330.9 (35)	334.6 (36)	333.5 (34)
	9m53s	11m50s	17m26s	24m51s	47m59s
bm50.txt	11.75 (7)	11.28 (7)	11.84 (7)	11.80 (7)	11.37 (7)
	30s	34s	46s	1m37s	3m53s

Table 5.3: Experimental results of Algorithm 9, endpoints not added

Case	Mask w. 7	Mask w. 6	Mask w. 5	Mask w. 4	Mask w. 3
bm02.txt	279.6 (27)	283.5 (27)	292.0 (28)	306.0 (29)	315.6 (30)
	1m53s	2m14s	3m1s	5m48s	15m18s
bm08.txt	98.14 (17)	97.63 (18)	101.9 (17)	103.0 (19)	103.2 (18)
	51s	1m1s	1m33s	3m14s	9m24s
bm09.txt	1044 (53)	1065 (57)	1099 (58)	1142 (59)	1173 (62)
	7m7s	11m41s	18m14s	28m26s	66m1s
bm10.txt	31.82 (9)	31.80 (10)	32.92 (10)	33.40 (10)	33.37 (9)
	12s	15s	22s	38s	2m1s
bm13.txt	435.6 (33)	450.3 (35)	471.3 (35)	478.5 (38)	502.5 (40)
	2m54s	3m39s	5m42s	10m20s	23m28s
bm18.txt	64.12 (14)	64.19 (13)	65.88 (13)	66.68 (15)	68.36 (15)
	36s	44s	1m1s	2m15s	6m23s
bm20.txt	5000 (86)	5630 (115)	6224 (164)	6566 (188)	6868 (197)
	17m52s	43m11s	124m44s	465m47s	1025m42s
bm26.txt	0.1880 (2)	0.1671 (2)	0.1760 (2)	0.1872 (2)	0.1780 (2)
	3s	4s	5s	8s	17s
bm29.txt	3694 (72)	4104 (95)	4446 (130)	4704 (136)	4874 (151)
	8m16s	19m54s	58m13s	191m17s	424m7s
bm43.txt	48.52 (11)	48.35 (11)	48.90 (11)	50.76 (12)	51.10 (12)
	14s	17s	23s	43s	1m45s
bm44.txt	623.7 (41)	622.4 (39)	650.2 (41)	665.7 (44)	693.0 (44)
	4m4s	5m27s	7m10s	11m37s	28m9s
bm45.txt	13.41 (7)	13.42 (7)	14.04 (8)	14.05 (8)	14.05 (7)
	14s	17s	25s	49s	2m30s
bm47.txt	1863 (62)	1983 (72)	2063 (76)	2209 (81)	2371 (92)
	5m12s	14m0s	27m28s	48m45s	113m59s
bm49.txt	294.1 (30)	292.0 (30)	294.5 (31)	307.5 (30)	315.6 (32)
	2m55s	4m15s	4m59s	10m32s	27m18s
bm50.txt	10.53 (6)	10.69 (6)	10.64 (7)	11.15 (7)	10.76 (6)
	11s	14s	21s	41s	2m0s

Table 5.4: Experimental results of Algorithm 8

Case	n	Evaluation	Net profit	Running time
b02.txt	35	2381.0	281.0	57m32s
b08.txt	19	1246.2	106.2	17m1s
b09.txt	69	5172	1032	213m5s
b10.txt	11	695.54	35.54	5m12s
b13.txt	41	2936.5	476.5	70m36s
b18.txt	15	968.88	68.88	10m49s
b20.txt	217	18913	5893	1959m25s
b26.txt	2	120.1963	0.1963	13s
b29.txt	167	14206	4186	1092m20s
b43.txt	13	834.23	54.23	7m19s
b44.txt	49	3570.6	630.6	109m20s
b45.txt	8	495.06	15.06	3m3s
b47.txt	99	7940	2000	386m58s
b49.txt	36	2461.2	301.2	63m35s
b50.txt	7	431.78	11.78	2m19s

times for the 10 runs.

5.4 Observations

Algorithms 1 and 2 performed almost equally well, while Algorithm 4 consistently produced better solutions at a modest increase in computational cost.

From the last line of Table 5.1 we can see that the mean running time for Algorithm 2 was about $\frac{1}{6}$ higher than that of Algorithm 1, while the mean running time for Algorithm 4 was just slightly higher than that of Algorithm 2 (by about 6%). For a given algorithm, we would expect that the mean running times for the individual cases be influenced by the number of linear constraints, but initial configurations can probably also make an impact. For example, for Algorithm 4, the case with the longest running time was wf37.txt, which has only got four constraints.

Table 5.2 suggests that a smaller mask width usually (not always) gives better results for Algorithm 9, but generally leads to a longer running time, as one would expect.

From Tables 5.2-5.4 we can see that Algorithm 9 tends to give better results than Algorithm 8, except in extreme cases with very few turbines, such as bm26.txt and bm50.txt. With the same exceptions, the running time of Algorithm 8 is of the same order of magnitude as that of Algorithm

9 with mask width 3. However, when the (optimal) number of turbines is high, Algorithm 9 with larger mask width still outperforms Algorithm 8, while being faster.

The running time of the simplified version of Algorithm 9 (Table 5.3) is shorter than that of Algorithm 9 (Table 5.2) due to the smaller number of possible slots for turbines. It appears that the running times in the second last column of Table 5.2 (mask width 4, with endpoints) are close to the running times in the last column of Table 5.3 (mask width = 3, without endpoint). On comparing these two columns, we can see that the net profit is generally higher in the former. From this, we conclude that it is better to include endpoints than to reduce the mask width. This is presumably due to the fact that even though the number of slots is roughly the same, they are more spread out when endpoints are included.

Of course, the running time of Algorithm 8 can also be tuned, by using different values for N. The differences in the net profits are therefore presumably the most noteworthy.

The large variations in the running times seen in Table 5.4 can be explained by the recursive nature of (3.5). This implies that the number of calculations that must be carried out in each step of Algorithm 8 is $O(n^2)$.

Chapter 6

Conclusion

We started out by considering a simple wake model known as the Jensen model. In Katić *et al.*'s paper, this model is refined and equipped with a wake combination model.

Based upon the authors' own comments and data, we proposed a further improvement of their model. Then, we provided a complete mathematical model for Problem 1 (with a fixed number of turbines), and one for Problem 2 (with a fixed and finite set of possible locations).

Finally, we set out to find a suitable optimization method. The main focus was on heuristic methods. Exact methods were tried out, but the convergence was very slow, even on very simple cases.

Among the different heuristic optimization methods that we tested, Algorithm 9 seemed to be the most promising one. When there are many turbines involved, it is probably advantageous to consider one turbine at a time (as Algorithm 9 more or less does), instead of trying to move all of them simultaneously (as the other algorithms tend to do). It is also an advantage that a large number of the parameters can be pre-calculated, although storage might become a problem on a large grid.

Experiments documented in this thesis suggest that the combination of the abovementioned wake model and algorithm will be a useful tool in designing wind farms.

Bibliography

[1] http://en.wikipedia.org/wiki/Wind_farm

[2] http://vestavindoffshore.no/havsul

[3] Yngve Heggelund, Inge Morten Skaar, *Fast evaluation of wind farm flow for use in layout optimization,* NORCOWE-RR-C-10-WP4-001.

[4] http://www.ewec2011.info/fileadmin/ewec2011_files/images/conference/Side_Events/Research_over_from_Risoe_-_P.H._Madsen.ppt&ei=vR9GT6_QHoHd4QS0xbDiDg&usg=AFQjCNHMgS6Dl6Weawk9IG4AUv__7eeHfw&sig2=nlWBROq6sfG6lfVy_3w86g&cad=rja

[5] I. Katić, J. Højstrup, N. O. Jensen, *A simple model for cluster efficiency,* Proceedings of EWEC'86, Rome, Italy (1986), 407–410.

[6] http://www.wasp.dk/Products/WAsP/WakeEffectModel.html

[7] http://www.risoe.dtu.dk/News_archives/News/2011/0630_TOPFARM.aspx?sc_lang=en

[8] Pierre-Elouan Réthoré, Peter Fuglsang, Gunner C. Larsen, Thomas Buhl, Torben J. Larsen, Helge A. Madsen, *TopFarm: Multi-fidelity Optimization of Offshore Wind Farm,* Proceedings of the Twenty-first (2011) International and Polar Engineering Conference, 516–524.

[9] http://windenergyresearch.org/2010/10/state-of-the-art-in-wind-farm-layout-optimization

[10] http://en.wikipedia.org/wiki/Genetic_algorithms

[11] http://en.wikipedia.org/wiki/Mathematical_optimization#Heuristics

[12] http://en.wikipedia.org/wiki/Betz'_law

[13] N. O. Jensen, *A note on wind generator interaction,* Risø M 2411. Risø National Laboratory, Roskilde (Denmark), 1983.

[14] Hans Georg Beyer, Bernhard Lange, Hans-Peter Waldl, *Modelling Tools for Wind Farm Upgrading,* Proceedings of the EUWEC, 1996, four unnumbered pages.

[15] S. Frandsen, R. Barthelmie, S. Pryor, O. Rathmann, S. Larsen, J. Højstrup and M. Thøgersen, *Analytical modelling of wind speed deficit in large offshore wind farms,* Wind Energy 9(1-2): 39-53 (2006).

[16] G. C. Larsen, *A Simple Wake Calculation Procedure,* Risø M-2760, Risø National Laboratory, Roskilde (Denmark), 1988.

[17] http://www.windfinder.com/windstats

[18] R. J. Barthelmie, L. Folkerts, G. C. Larsen, K. Rados, S. C. Pryor, S. T. Frandsen, B. Lange, G. Schepers, *Comparison of Wake Model Simulations with Offshore Wind Turbine Wake Profiles Measured by Sodar,* Journal of Athmospheric and Oceanic Technology, 23(2006), 888–901.

[19] Jorge Nocedal, Stephen J. Wright, *Numerical Optimization,* Springer, 2006 (second edition), 238–240.

[20] http://en.wikipedia.org/wiki/Simulated_annealing

[21] http://en.wikipedia.org/wiki/Particle_swarm_optimization

[22] http://www.neutreeko.net/wind.htm

Appendix A

GAMS code for 1-dimensional version

```
option optcr = 1.0e-12;
option optca = 1.0e-9;
option reslim = 360;

Parameter k2;
          k2 = 0.075;

Parameter r;
          r = 0.1;

Parameter v0;
          v0 = 1000.0;

Parameter n;
          n = 11;

Set i /1*11/;
alias (i, j);

Positive variables x(i);
x.up(i) = 10.0;
x.l(i) = 10.0 * (ord(i) - 1) / (n - 1);

Variables delta(i, j), v(i), obj;

Equations orden(i, j), defcoeff(i, j), velocity(j), pwr;

   orden(i, j)$(ord(i) < ord(j)).. x(i) =l= x(j);
```

```
    defcoeff(i, j)$(ord(i) < ord(j)).. delta(i, j) =e=
       (2.0 / 3.0) * power(r / (r + k2 * (x(j) - x(i))), 2);
    velocity(j).. power(v0 - v(j), 3) =e= sum(i$(ord(i) <
       ord(j)), power(delta(i, j) * v(i), 3));
    pwr.. obj =e= sum(i, power(v(i), 3));

option minlp = baron;

model wind1 /all/;

Solve wind1 maximizing obj using minlp;
```

Appendix B

GAMS code for Problem 1

```
option optcr = 1.0e-9;
option optca = 1.0e-9;
option reslim = 5400;

Parameter k2;
          k2 = 0.075;

Parameter r;
          r = 0.1;

Parameter v0;
          v0 = 100.0;

Parameter n;
          n = 2;

Set i /1*2/;
alias (i, j);
Set k /1*2/;
Set w /0*23/;

parameter p(w)
   /0 0.041666667
    1 0.041666667
    2 0.041666667
    3 0.041666667
    4 0.041666667
    5 0.041666667
    6 0.041666667
    7 0.041666667
```

```
     8 0.041666667
     9 0.041666667
    10 0.041666667
    11 0.041666667
    12 0.041666667
    13 0.041666667
    14 0.041666667
    15 0.041666667
    16 0.041666667
    17 0.041666667
    18 0.041666667
    19 0.041666667
    20 0.041666667
    21 0.041666667
    22 0.041666667
    23 0.041666667/;

parameters cos(w)
   /0   1.0
    1   0.965926
    2   0.866025
    3   0.707107
    4   0.5
    5   0.258819
    6   0.0
    7   -0.258819
    8   -0.5
    9   -0.707107
   10   -0.866025
   11   -0.965926
   12   -1.0
   13   -0.965926
   14   -0.866025
   15   -0.707107
   16   -0.5
   17   -0.258819
   18   0.0
   19   0.258819
   20   0.5
   21   0.707107
   22   0.866025
   23   0.965926/;

parameter sin(w)
```

```
/0   0.0
 1   0.258819
 2   0.5
 3   0.707107
 4   0.866025
 5   0.965926
 6   1.0
 7   0.965926
 8   0.866025
 9   0.707107
10   0.5
11   0.258819
12   0.0
13   -0.258819
14   -0.5
15   -0.707107
16   -0.866025
17   -0.965926
18   -1.0
19   -0.965926
20   -0.866025
21   -0.707107
22   -0.5
23   -0.258819/;

Positive variables x(k), y(k);
x.up(k) = 10.0;
x.l(k)$(ord(k)=1) = 1.0;
x.l(k)$(ord(k)=2) = 9.0;
y.up(k) = 10.0;
y.l(k)$(ord(k)=1) = 1.0;
y.l(k)$(ord(k)=2) = 1.0;

Binary variables A(i, k, w);

Variables delta(i, j, w), d(i, j, w), s(i, j, w), v(i, w), obj;
delta.up(i, j, w) = 1.0;
delta.lo(i, j, w) = -1.0;
d.up(i, j, w) = 14.2;
d.lo(i, j, w) = -14.2;
s.up(i, j, w) = 14.2;
s.lo(i, j, w) = -14.2;
v.up(i, w) = 100.0;
v.lo(i, w) = 0.0;
```

```
obj.lo = 0.0;

Equations permmat1(i,w), permmat2(k,w), xcoord(i, j, w),
ycoord(i, j, w), sign(i, j, w), deficitcoeff(i, j, w),
   velocity(j, w), velocity2(j, w), pwr;

   permmat1(i,w).. sum(k, A(i,k,w)) =e= 1.0;
   permmat2(k,w).. sum(i, A(i,k,w)) =e= 1.0;
   xcoord(i, j, w).. d(i, j, w) =e= sum(k, (cos(w)*x(k)
      +sin(w)*y(k)) * (A(j, k, w) - A(i, k, w)));
   ycoord(i, j, w).. s(i, j, w) =e= sum(k, (-sin(w)*x(k)
      +cos(w)*y(k)) * (A(j, k, w) - A(i, k, w)));
   sign(i, j, w)$(ord(i) < ord(j)).. d(i, j, w) =g= 0;
   deficitcoeff(i, j, w).. delta(i, j, w) =e= (2.0/3.0)
      * power(r/(r+k2*d(i, j, w)), 2) * exp(-power(s(i,
      j, w) / (r + k2 * d(i, j, w)), 2));
   velocity(j,w)$(ord(j)=1).. v(j, w) =e= v0;
   velocity2(j, w)$(ord(j)>1).. power(v0 - v(j, w), 3)
      =e= sum(i$(ord(i) < ord(j)), power(delta(i, j, w)
      * v(i, w), 3));
   pwr.. obj =e= sum(w, sum(i, p(w) * power(v(i, w), 3)));

option minlp = baron;

model wind2 /all/;

Solve wind2 maximizing obj using minlp;
```

Appendix C

GAMS code for Problem 2

```
option optcr = 1.0e-9;
option optca = 1.0e-9;
option reslim = 5400;

Parameter v0;
          v0 = 100.0;

Parameter n;
          n = 9;

Parameter c;
          c = 0.5;

Set i /1*9/;
alias (i, j);
Set k /1*9/;
Set w /0*7/;

parameter p(w)
   /0 0.125
    1 0.125
    2 0.125
    3 0.125
    4 0.125
    5 0.125
    6 0.125
    7 0.125/;

table A(i, k, w)
    0 1 2 3 4 5 6 7
```

1.1	1	1	1	0	0	0	0	0
1.2	0	0	0	0	0	0	0	0
1.3	0	0	0	1	1	0	0	0
1.4	0	0	0	0	0	0	0	0
1.5	0	0	0	0	0	0	0	0
1.6	0	0	0	0	0	0	0	0
1.7	0	0	0	0	0	0	1	1
1.8	0	0	0	0	0	0	0	0
1.9	0	0	0	0	0	1	0	0
2.1	0	0	0	0	0	0	0	0
2.2	0	1	1	1	0	0	0	0
2.3	0	0	0	0	0	0	0	0
2.4	1	0	0	0	0	0	0	1
2.5	0	0	0	0	0	0	0	0
2.6	0	0	0	0	1	1	0	0
2.7	0	0	0	0	0	0	0	0
2.8	0	0	0	0	0	0	1	0
2.9	0	0	0	0	0	0	0	0
3.1	0	0	0	0	0	0	0	0
3.2	0	0	0	0	0	0	0	0
3.3	0	0	1	0	0	0	0	0
3.4	0	1	0	0	0	0	0	0
3.5	0	0	0	0	0	0	0	0
3.6	0	0	0	1	0	0	0	0
3.7	1	0	0	0	0	0	0	0
3.8	0	0	0	0	0	1	0	1
3.9	0	0	0	0	1	0	1	0
4.1	0	0	0	1	0	0	0	1
4.2	1	0	0	0	1	0	0	0
4.3	0	1	0	0	0	1	0	0
4.4	0	0	1	0	0	0	1	0
4.5	0	0	0	0	0	0	0	0
4.6	0	0	0	0	0	0	0	0
4.7	0	0	0	0	0	0	0	0
4.8	0	0	0	0	0	0	0	0
4.9	0	0	0	0	0	0	0	0
5.1	0	0	0	0	0	0	0	0
5.2	0	0	0	0	0	0	0	0
5.3	0	0	0	0	0	0	0	0
5.4	0	0	0	0	0	0	0	0
5.5	1	1	1	1	1	1	1	1
5.6	0	0	0	0	0	0	0	0
5.7	0	0	0	0	0	0	0	0
5.8	0	0	0	0	0	0	0	0

```
5.9  0  0  0  0  0  0  0  0
6.1  0  0  0  0  0  0  0  0
6.2  0  0  0  0  0  0  0  0
6.3  0  0  0  0  0  0  0  0
6.4  0  0  0  0  0  0  0  0
6.5  0  0  0  0  0  0  0  0
6.6  0  0  1  0  0  0  1  0
6.7  0  1  0  0  0  1  0  0
6.8  1  0  0  0  1  0  0  0
6.9  0  0  0  1  0  0  0  1
7.1  0  0  0  0  1  0  1  0
7.2  0  0  0  0  0  1  0  1
7.3  1  0  0  0  0  0  0  0
7.4  0  0  0  1  0  0  0  0
7.5  0  0  0  0  0  0  0  0
7.6  0  1  0  0  0  0  0  0
7.7  0  0  1  0  0  0  0  0
7.8  0  0  0  0  0  0  0  0
7.9  0  0  0  0  0  0  0  0
8.1  0  0  0  0  0  0  0  0
8.2  0  0  0  0  0  0  1  0
8.3  0  0  0  0  0  0  0  0
8.4  0  0  0  0  1  1  0  0
8.5  0  0  0  0  0  0  0  0
8.6  1  0  0  0  0  0  0  1
8.7  0  0  0  0  0  0  0  0
8.8  0  1  1  1  0  0  0  0
8.9  0  0  0  0  0  0  0  0
9.1  0  0  0  0  0  1  0  0
9.2  0  0  0  0  0  0  0  0
9.3  0  0  0  0  0  0  1  1
9.4  0  0  0  0  0  0  0  0
9.5  0  0  0  0  0  0  0  0
9.6  0  0  0  0  0  0  0  0
9.7  0  0  0  1  1  0  0  0
9.8  0  0  0  0  0  0  0  0
9.9  1  1  1  0  0  0  0  0 ;

table delta(i, j, w)
     0        1        2        3        4        5        6        7
1.1  0        0        0        0        0        0        0        0
1.2  0        0        0        0        0        0        0        0
1.3  0        0        0        0        0        0        0        0
1.4  0.0295   0        0.0295   0        0.0295   0        0.0295   0
```

1.5	0	0.0168	0	0.0168	0	0.0168	0	0.0168
1.6	0	0	0	0	0	0	0	0
1.7	0.0092	0	0.0092	0	0.0092	0	0.0092	0
1.8	0	0	0	0	0	0	0	0
1.9	0	0.0049	0	0.0049	0	0.0049	0	0.0049
2.1	0	0	0	0	0	0	0	0
2.2	0	0	0	0	0	0	0	0
2.3	0	0	0	0	0	0	0	0
2.4	0	0	0	0	0	0	0	0
2.5	0.0295	0	0.0295	0	0.0295	0	0.0295	0
2.6	0	0	0	0	0	0	0	0
2.7	0	0.0168	0	0.0168	0	0.0168	0	0.0168
2.8	0.0092	0	0.0092	0	0.0092	0	0.0092	0
2.9	0	0	0	0	0	0	0	0
3.1	0	0	0	0	0	0	0	0
3.2	0	0	0	0	0	0	0	0
3.3	0	0	0	0	0	0	0	0
3.4	0	0	0	0	0	0	0	0
3.5	0	0	0	0	0	0	0	0
3.6	0.0295	0	0.0295	0	0.0295	0	0.0295	0
3.7	0	0	0	0	0	0	0	0
3.8	0	0.0168	0	0.0168	0	0.0168	0	0.0168
3.9	0.0092	0	0.0092	0	0.0092	0	0.0092	0
4.1	0	0	0	0	0	0	0	0
4.2	0	0	0	0	0	0	0	0
4.3	0	0	0	0	0	0	0	0
4.4	0	0	0	0	0	0	0	0
4.5	0	0	0	0	0	0	0	0
4.6	0	0	0	0	0	0	0	0
4.7	0.0295	0	0.0295	0	0.0295	0	0.0295	0
4.8	0	0	0	0	0	0	0	0
4.9	0	0	0	0	0	0	0	0
5.1	0	0	0	0	0	0	0	0
5.2	0	0	0	0	0	0	0	0
5.3	0	0	0	0	0	0	0	0
5.4	0	0	0	0	0	0	0	0
5.5	0	0	0	0	0	0	0	0
5.6	0	0	0	0	0	0	0	0
5.7	0	0	0	0	0	0	0	0
5.8	0.0295	0	0.0295	0	0.0295	0	0.0295	0
5.9	0	0.0168	0	0.0168	0	0.0168	0	0.0168
6.1	0	0	0	0	0	0	0	0
6.2	0	0	0	0	0	0	0	0
6.3	0	0	0	0	0	0	0	0

```
6.4 0       0       0       0       0       0       0       0
6.5 0       0       0       0       0       0       0       0
6.6 0       0       0       0       0       0       0       0
6.7 0       0       0       0       0       0       0       0
6.8 0       0       0       0       0       0       0       0
6.9 0.0295  0       0.0295  0       0.0295  0       0.0295  0
7.1 0       0       0       0       0       0       0       0
7.2 0       0       0       0       0       0       0       0
7.3 0       0       0       0       0       0       0       0
7.4 0       0       0       0       0       0       0       0
7.5 0       0       0       0       0       0       0       0
7.6 0       0       0       0       0       0       0       0
7.7 0       0       0       0       0       0       0       0
7.8 0       0       0       0       0       0       0       0
7.9 0       0       0       0       0       0       0       0
8.1 0       0       0       0       0       0       0       0
8.2 0       0       0       0       0       0       0       0
8.3 0       0       0       0       0       0       0       0
8.4 0       0       0       0       0       0       0       0
8.5 0       0       0       0       0       0       0       0
8.6 0       0       0       0       0       0       0       0
8.7 0       0       0       0       0       0       0       0
8.8 0       0       0       0       0       0       0       0
8.9 0       0       0       0       0       0       0       0
9.1 0       0       0       0       0       0       0       0
9.2 0       0       0       0       0       0       0       0
9.3 0       0       0       0       0       0       0       0
9.4 0       0       0       0       0       0       0       0
9.5 0       0       0       0       0       0       0       0
9.6 0       0       0       0       0       0       0       0
9.7 0       0       0       0       0       0       0       0
9.8 0       0       0       0       0       0       0       0
9.9 0       0       0       0       0       0       0       0 ;

Binary variables alpha(k);

Variables v(i, w), obj;
v.up(i, w) = 100.0;
v.lo(i, w) = -100.0;
obj.lo = 0.0;

Equations velocity(j, w), pwr;

    velocity(j, w).. power(v(j, w), 3) - 3 * v0 * power(v(j,
```

```
      w), 2) + 3 * v0 * v0 * v(j, w) =e= (sum(k, A(j, k, w)
      * alpha(k))) * (power(v0, 3) - sum(i$(ord(i) < ord(j)),
      power(delta(i, j, w) * v(i, w), 3)));
   pwr.. obj =e= sum(w, sum(i, p(w) * power(v(i, w), 3)))
      - c*sum(k, alpha(k));

option minlp = baron;

model wind3 /all/;

Solve wind3 maximizing obj using minlp;
```