MODELING AND MAXIMIZING POWER IN WIND TURBINE ARRAYS

BY

LUCAS BUCCAFUSCA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Associate Professor Carolyn L. Beck

ABSTRACT

This work considers a specific application domain, that of wind turbine arrays, and explores algorithms for determining individual axial induction factors that optimize overall energy extraction. Large wind turbine arrays, or wind farms, can be viewed as large coupled networks, for which the application of traditional optimization techniques are impractical.

A brief discussion on wind farm models and traditional optimization approaches leads us to a spatial dynamic programming approach to maximize power extraction under the condition of uniform wind. This differs from prior work in which only a dynamic programming approach for a more restrictive near-field model has been analyzed.

In our work, we propose an algorithm that leads to solutions for both the near-field and far-field models. Simulation results are discussed, which demonstrate our algorithm provides improved performance compared to prior work using only near-field approaches.

To my parents, Monica and Osvaldo, for their love and encouragement
To my sister, Lisa, for always being there
And to Vidya, who never left my side

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Carolyn Beck, for all of her guidance throughout the research process. I will always be grateful for her patience, dedication, and unparalleled continuous support. I could not have imagined having a better advisor and mentor.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Wind energy was the largest source of new power generation in the United States in 2015 [1]. With nearly $100 billion invested in new wind projects since 2008, wind farms are at the forefront of sustainable energy. In addition to building new wind farms, new control algorithms are being designed to make current wind farms more cost efficient [2].

Studies on the optimization and analysis of wind turbines focus either on single turbines using a complex wake model or on large groups of turbines using a simple wind model. Due to the complexity of the dynamics of the system [3], it becomes infeasible to handle detailed wind models with large numbers of turbines. In extending the results provided by elaborate models to arrays of turbines, many of the aerodynamic interactions between turbines are overlooked. While the single turbine models can be used to accurately maximize power extraction for the single turbine case, power losses arise from turbines negatively impacting each other; when a turbine lies in another turbine's wake, increasing the rotation speed of the upstream turbine reduces the amount of energy that can be extracted downstream. As such, most algorithms seek to find the set of controls that maximize power extraction for the wind farm as a whole [4].

We begin by analyzing the three wind models most commonly used for axial-induction-factor-based control in Chapter 2. In Chapter 3 is a brief discussion on previous wind farm results. This provides a background on solving the wind farm problem using non-convex optimization techniques.

There exist many ways to handle optimization problems. Chapter 4 provides some background in using coordinate ascent algorithms and gradient methods under the constraint of a non-convex problem. Combining these ideas with Bellman's 'principle of optimality' [5] leads to the derivation of our control algorithm. This synthesis is also covered in Chapter 5. Finally, Chapter 6 provide simulations comparing wind power extraction for various turbine setups and control schemes.

CHAPTER 2

WIND FARM MODELS

The conversion of wind energy into power involves two processes: the first is extracting the kinetic energy from wind and its transformation into mechanical energy at the rotor axis, and second is the processing it into useful energy [6]. In the primary process, wind turbines extract the energy from wind, leading to a reduction in the wind speed behind the rotor and disrupt the air flow, which is known as the wake effect of the wind turbine. Thus, the downstream wind turbines receive a modified wind inflow both in terms of mean velocity and turbulence, producing less energy as a result.

2.1 Park (Jensen) Wake Model

The Park wake model (otherwise known as the Jensen wake model) is an empirically based linear expanding wake model. The model was first described by N. O. Jensen in 1984. The model is a single wake model that describes the turbine wake in terms of an initial loss and a decay rate. It is represented by a uniform velocity profile at any one location downstream. In 2-D models, this is characterized by an x-coordinate within a wake profile. Due to the simplified constant velocity profile, the model behaves poorly when used to make wake predictions near the turbine, as it oversimplifies many of the fluid behaviors that occur. Nevertheless, due to the simplicity of the model, this wake model is used to design and compare axial-induction-based control algorithms. The typical one-turbine model is shown in Figure 2.1: The wake size is dependent on a constant k (known as a roughness coefficient). This roughness coefficient depends on many factors of the wind, including density, temperature and humidity. The larger the k, the wider out the wake expands. In practice, this k varies based on the distance between the turbines with a typical value being $k = 0.075$ over land and $k = 0.04$ for offshore. It has been shown that the lower decay factor results in slower growth of the wake width. This decay is explicitly shown in Figure 2.2.

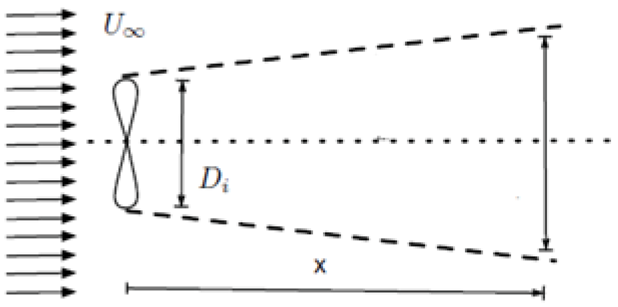

Figure 2.1: One-turbine model where D_i is the diameter of the turbine, U_∞ is the uniform infinite wind, and x is the horizontal displacement from the turbine. Adapted from [7].

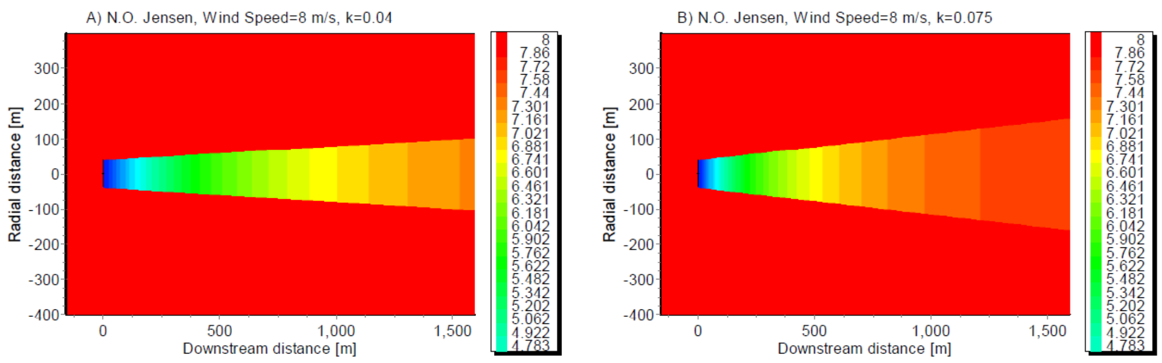

Figure 2.2: Simulations demonstrating wake degradation with identical settings but different k values [8].

The Park model is derived by the defining equations from an actuator disk, when the equation for power is modeled as

$$P(\lambda, \beta) = \frac{1}{2}\rho A_{rotor} v^3 C_p(\lambda, \beta) \tag{2.1}$$

where ρ is the air density, A_{rotor} is the area swept by the rotor, v is the velocity of the wind traveling through the disk and $C_p(\lambda, \beta)$ is a power coefficient that is determined based on the blade angle to the wind (β) and the tip speed ratio (λ) defined as

$$C_p(\lambda, \beta) = \frac{1}{2}(\frac{116}{\lambda} - 0.4\beta - 5)e^{\frac{-21}{\lambda}} \tag{2.2}$$

The Park model, parameterizes $C_p(\lambda, \beta)$ into a single variable known as the axial induction factor. This has an explicit meaning when dealing with wind turbines, as it is defined as the fractional decrease in wind velocity between the external wind and the wind turbine itself.

We consider a wind farm consisting of n wind turbines. For simplicity in developing the model, a common assumption is to have all wind turbines oriented orthogonal to the direction of wind travel.

Given uniform wind with magnitude, U_∞, the Park wake model states that each turbine affects wind velocity downstream by a diminishing multiplicative factor

$$V_i = U_\infty(1 - \Delta V_i), \tag{2.3}$$

with

$$\Delta V_i = 2u_i \left(\frac{D}{D + 2k\Delta d}\right)^2 \tag{2.4}$$

where Δd is the horizontal displacement from the turbine in question, k is the roughness coefficient, D is the diameter of the wave propagation cone, and u_i is the axial induction factor for turbine i.

Adapting (2.1) leads to the power equation:

$$P_i(u_i) = C_i u_i (1 - u_i)^2 V_i^3 \tag{2.5}$$

where the coefficient C_i combines A_{rotor} and ρ for a turbine i. At a known location in the intersection of wakes $1, \ldots, i$, the aggregate wind velocity is determined by:

$$V_i = U_\infty \left(1 - \sqrt{\sum_{i=1}^{k} (p_{ij}\Delta V_i)^2}\right) \tag{2.6}$$

where p_{ij} is the proportion of turbine j lying in the i'th wake, measured in the cross-sectional area between the velocity cone of the two turbines. A visual representation of this is seen in Figure 2.3.

Additionally, from actuator turbine dynamics it has been shown that in the absence of any interactions between turbines, there exists an upper bound on the amount of energy that can be extracted. This upper bound is to set $u_i = 1/3$ for all i, which approximates the optimal solution if the turbines are located sufficiently far apart for wake effects to be negligible.

It is important to note that since the eventual goal is to solve for the net power extraction of the whole system, the wind farm becomes an optimization problem in which we are attempting to solve for the value of u_i for each turbine that maximizes power.

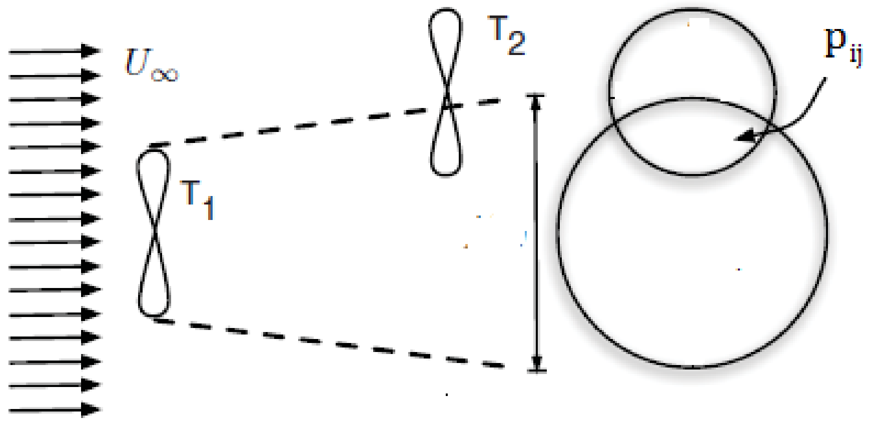

Figure 2.3: Two-turbine illustration showing the term p_{ij}.

Explicitly written, for the Park wake model have the following bounded optimization problem:

$$\max_{u_i \ldots u_n} P_{total}(u_i \ldots u_n) = \max_{u_i \ldots u_n} \sum_{i=1}^{n} C_i u_i (1-u_i)^2 V_i^3 \tag{2.7}$$

$$s.t.\ 0 \le u_i \le \frac{1}{3} \tag{2.8}$$

This problem is difficult to solve for a global optimum solution. In fact, attempting to solve for each u_i by exhaustively searching and optimizing over the full range of axial induction factors even for small numbers of turbines, becomes unreasonable.

For the majority of this work, the focus will be on solving for the optimal solutions of the optimization problem provided by (2.7) and (2.8). Once an algorithm is developed for this problem, it can be extended to other wind farm models.

For most works analyzing axial-induction-based control, variants of the Park model are often used. These still take advantage of the simplistic nature of combining wake effects and require less computational time.

2.1.1 Near-Field Approximation

Previous work has focused on exploring the near-field model between turbines in which it is assumed that each turbine only affects adjacent turbines in the array. This model allows for an optimal solution to be obtained using the Bellman equation, rather than a model that more accurately represents wind and turbine dynamics.

The near-field equations that dictate downstream wind wake effects for a single turbine are identical, but when solving for the aggregate wind velocity using (2.6) we instead have $p_{ij} = 0$ for $i \neq j - 1$.

We demonstrate the differences using a simple example: three turbines in an array as shown in Figure 2.4.

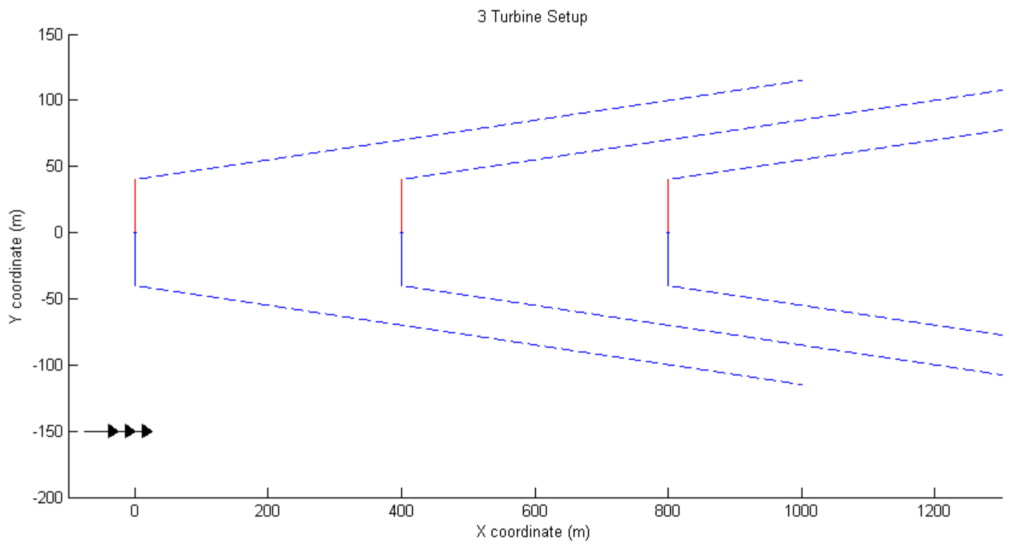

Figure 2.4: Simple three-turbine wind farm. Each turbine has a diameter of 80 meters, and they are spaced 400 meters apart. Turbines are numbered from upstream to downstream, i.e. since the wind is traveling in the direction of the positive x-axis, they are indexed from left to right.

Our optimization problem for this simple example becomes:

$$\max_{u_1,u_2,u_3} P_{total}(u_1,u_2,u_3) = \max_{u_i \dots u_n} \sum_{i=1}^{3} C u_i (1-u_i)^2 V_i^3 \tag{2.9}$$

In this case, the velocities across the three turbines in the array can be written explicitly using the Park wake model given by (2.9):

$$V_1(u) = U_\infty \tag{2.10}$$

$$V_2(u) = U_\infty(1 - 2u_1 p_{12}) \tag{2.11}$$

$$V_3(u) = U_\infty(1 - 2\sqrt{(u_1 p_{13})^2 + (u_2 p_{23})^2}) \tag{2.12}$$

For the near-field, in which we disregard the interaction between turbines 1 and 3, the equations become:

$$V_1(u) = U_\infty \tag{2.13}$$

$$V_2(u) = U_\infty(1 - 2u_1 p_{12}) \tag{2.14}$$

$$V_3(u) = U_\infty(1 - 2u_2 p_{23}) \tag{2.15}$$

The solution to the set of axial induction factors to maximize the power differ because of the cross-turbine terms found in $V_3(u)$.

2.1.2 High-Fidelity Extensions

A high-fidelity simulator designed for wind plant studies is known as *Simulator fOr Wind Farm Applications* (SOWFA). SOWFA provides a solver that has been used in numerous prior wind control studies. Vigorous simulations have demonstrated that results provided by SOWFA differ from the Park wake model. As such, the goal has been to extend the model to further increase the validity to real-world applications.

To better accommodate the effects of partial wake overlap, the Park wake model is expanded by adding a correction factor η in the power term of each turbine thereby adjusting (2.7) to become

$$\max_{u_i \dots u_n} P_{total}(u_i \dots u_n) = \max_{u_i \dots u_n} \sum_{i=1}^{n} C_i u_i (1 - u_i)^2 V_i^3 \eta \tag{2.16}$$

$$s.t.\ 0 \le u_i \le \frac{1}{3} \tag{2.17}$$

where $\eta = 0.8051$ has been derived to match real-world data from the National Renewable Energy Laboratory [9].

Another improvement to the Park model is to separate the wake into three zones: an inner, middle and outer wake. This changes (2.4) to become

$$\Delta V_i = 2u_i \left(\frac{D}{D + 2m_{U,q} k_i \Delta d} \right)^2 \tag{2.18}$$

where $m_{U,q}$ are scaling factors that define velocity breakdown in each zone. A wake adjustment term k_i is introduced for each turbine, derived from the axial-induction settings of the turbines that are upstream from it.

2.2 G. C. Larsen Wake Model

This model was first described by Larsen in 1988 [8]. Originally derived from the Prandtl turbulent boundary layer equations, it results in closed-form solutions for the width of the wake and the mean velocity profile in the wake. The final form of the equation used to determine the velocity difference at the radial position r at the position x downstream is

$$U_\infty - V_i = -\frac{U_\infty}{9}(C_t A_{rotor} x^{-2})^{\frac{1}{3}} \left\{ r^{\frac{3}{2}} (3c_1^2 C_t A_{rotor} x)^{-\frac{1}{2}} - \left(\frac{35}{2\pi}\right)^{\frac{3}{10}} (3c_1^2)^{-\frac{1}{5}} \right\}^2 \tag{2.19}$$

where U_∞ is the free stream velocity, V_i is the downstream velocity of the wake, A_{rotor} the rotor area, C is the coefficient of thrust, and the constant c_1 is known as the mixing length. c_1 is correlated to the k discussed earlier. Figure 2.5 simulates a single wake using this model.

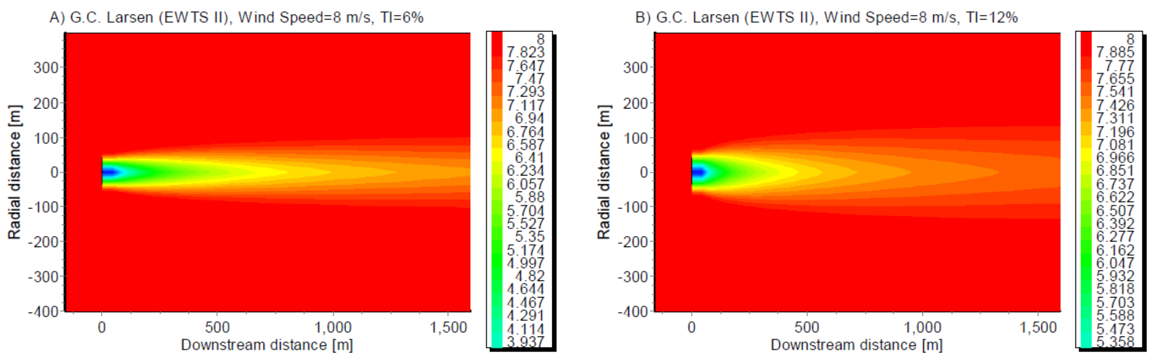

Figure 2.5: Simulation demonstrating the G. C. Larsen wake model with identical settings but different k values [8].

Due to the use of a power curve based on a single wind speed, the non-uniform velocity profiles calculated by the Eddy and Larsen wake models must be averaged in order to predict power. Summation of multiple wakes is often calculated using the velocity deficit sum of squares. As the number of turbines grows, this rapidly becomes computationally intractable, as it often involves integrating over the entire area of the turbine and averaging out multiple nonlinear effects. For this reason, this model is often used when the number of turbines is small.

2.3 Ainslie Eddy Viscosity Wake Model

As opposed to the previous two models based on the kinematics of the wind, a more complex model is based on the thin shear layer approximation of the Navier-Stokes equation. This model is derived on the field and wake turbulence models. The most commonly used 2-D eddy viscosity model was derived by Ainslie in 1988. The approximation assumes an axisymmetric, stationary, fully turbulent wake with zero circumferential velocity and negligible pressure gradients outside the wake region. By writing the differential equation in cyclindrical coordinates, aligned with the rotation of the turbine, the model is straightforward to use to numerically simulate the wake. This model is simulated in Figure 2.6.

$$\frac{1}{r}\frac{\partial}{\partial r}(ru_r) + \frac{1}{r}\frac{\partial u_\phi}{\partial \phi} + \frac{\partial u_z}{\partial z} = 0 \tag{2.20}$$

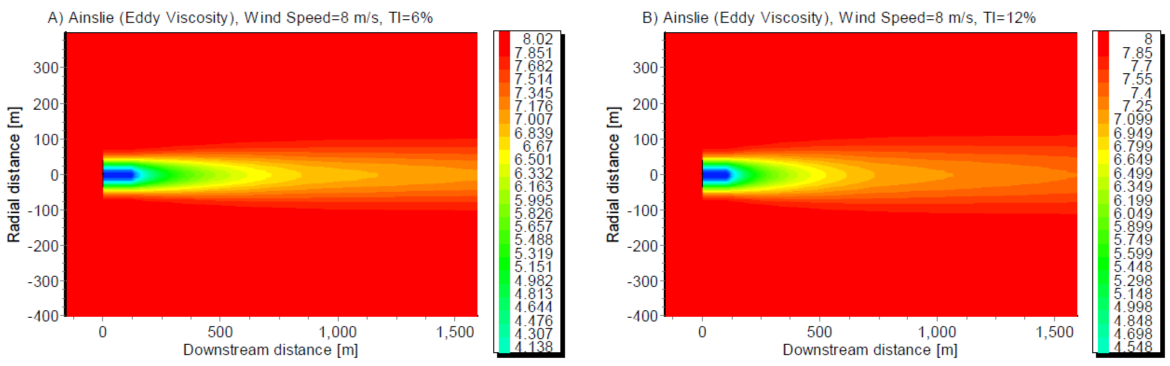

Figure 2.6: Simulation demonstrating the Ainslie Eddy viscosity wake model with identical settings but different k values [8].

The eddy viscosity model disregards effects that are necessary for the velocity in the wake to recover far downstream. As such, this model is primarily used when turbines are expected to be close to each other. Almost all analyses using this model use a near-field approximation in which any wake effects beyond the nearest neighbor are disregarded.

CHAPTER 3

PRIOR WIND FARM ANALYSIS RESULTS

Most prior work on analysis of wind farms is focused either on what is referred to as a near-field simplification or on a single-turbine model.

3.1 Near-Field Approximation Results

In this methodology, one of the wake models (typically the Park model) is taken and used to find the wind degradation only of adjacent turbines [10]. All other interactions are ignored. Upstream turbine axial induction factors affect all turbines downstream in a cascading manner, rather than directly. Most near-field approximation methods use a configuration derived from dynamic programming [11].

Dynamic programming is based on Bellman′s principle of optimality. He argues that the optimization problem can be solved by recursively solving Bellman′s equations to find time consistent policy functions. For a system of the form

$$x_{t+1} = f(x_t, u_t) \tag{3.1}$$

To solve this problem using Bellman′s method for control problems is to go backward in time. This can also be done spatially rather than temporally, if the terminal cost is known. For each x_t we find the x_{k+1} to have the smallest cost (a combination of the running cost and terminal cost). This is known as the ‘cost-to-go.’ Upon termination, an optimal sequence of x_t would be obtained, known from the principle of optimality. This states that an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy for all substates resulting from the first choice.

Two different methods were used to solve for optimal axial induction factors in the near field. For the first approach, the framework for the derivation was to explicitly use the near-field approach as a multi-stage optimal control problem. This allows the

optimization problem at each stage of the recursion is similar to optimizing the power $P_i(u_i)$ of a single turbine plus an additional term that comes from the optimal value function modeling the downstream system. Using this method the optimal axial induction factors for the near-field problem are given by the recursions:
For $i = N, N-1, \ldots, 1$

$$u_i := \frac{1}{2 + (1 - 6Q_{i+1})^{-1/2}} \tag{3.2}$$

$$Q_i := (u_i)(1-u_i)^2 + (1-2u_i)^3 Q_{i+1} \tag{3.3}$$

with the boundary condition $Q_{N+1} = 0$.
One approach is taken when the problem is converted from a temporal control problem to a spacial near-field interaction model. Rather than model the problem with controls, prior work [10] derives the dynamic programming equations that dictate the principle of optimality. Their derivation for the the optimal axial induction factors for the near-field problem are given by the following recursions:
For $i = N, N-1, \ldots, 1$

$$u_i := \frac{1}{3}\left(\frac{2 - 3\phi_{i+1}p_{ij}^2 - \sqrt{1 - 12\phi_{i+1}p_{ij}^2 + 9\phi_{i+1}p_{ij} + 3\phi_{i+1}p_{ij}^3}}{1 - \phi_{i+1}p_{ij}^3}\right) \tag{3.4}$$

$$\phi_i := (u_i)(1-u_i)^2 + (1-p_{ij}u_i)^3\phi_{i+1} \tag{3.5}$$

with the boundary condition $\phi_{N+1} = 0$ and $p_{ij} = 0$ for for $i \neq j-1$.

These solutions allow for the values of the axial induction factors to be solved rapidly for strings. Each term is found recursively, i.e. only depending on the prior turbine, and the proof guarantees optimality of the solution. The trade-off is that taking the near-field approximation as a wind farm model in order to analyze axial induction based control leads to solutions that vary wildly from using the far-wake version of the Park wake model. A clear example of this comes in Chapter 6, for the seven-turbine string.

3.2 Far-Field Approximation Results

The far-field model, in which the effect of turbines on all adjacent downwind turbines are included, is traditionally not examined. As the number of turbines increases, the size of the resulting matrix that tracks the coefficients between pairs of turbines grows combinatorically. In order to handle this, prior work [7] developed a model-free approach focusing on the design of local turbine control policies. With both simulations and through proofs, their algorithm convergences to a solution that improves on a greedy policy. However, this is not feasible in practice, as it cannot guarantee convergence in finite time.

CHAPTER 4

PROPOSED ALGORITHM

4.1 Background on Optimization Methods

4.1.1 General Framework of Coordinate Ascent

Coordinate ascent algorithms are derivative-free optimization methods [12]. To find a local minimum of a function $f(x)$, one searches along one coordinate direction at the current point in each iteration. Different coordinate directions are evaluated throughout the procedure. The motivation comes from trying to solve the maximization problem $\max_{x \in \mathbb{R}^n} f(x)$ where f is smooth and concave. (The f is continuously differentiable and the gradient is Lipschitz continuous.)

When trying to solve optimization problems in this manner, coordinate ascent algorithms have certain trade-offs to their simplicity in use. For some problems, the dimension is very large so it becomes computationally expensive to calculate gradients at each timestep. Thus, these derivative-free methods that involve maximizing each coordinate independently can converge to optimal solutions under certain conditions.

General pseudocode for a coordinate ascent algorithm is given by

```
Initialize $x^{(0)}$
    for $t = 1, 2, \ldots$
        Pick one coordinate $i$ from $1, 2, \ldots n$
        $x_i^{(t+1)} = {}_{x_i \in \mathbb{R}} f(x_i^{(t)}, \omega_{-i}^{(t)})$
    end
```

where $\omega_{-i}^{(t)}$ are fixed values that represent all other coordinates besides $x_i^{(t)}$.

There are many ways to choose $\omega_{-i}^{(t)}$ and several possible methods to choose which indices will be used.

Updating Coordinates at Each Iteration

$$\omega_{-i}^{t} = (x_1^{(t+1)}, \dots, x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, \dots, x_n^{(t)})$$

Gauss-Seidel Update This method uses the most updated value for each coordinate less than index i. Other values are from the previous iteration. An advantage of this method is the potential to converge faster, since recent updates are used. This method of coordinate update naturally fits well when applied to the Park wake model, as updating each variable is akin to updating turbines sequentially.

$$\omega_{-i}^{t} = (x_1^{(t)}, \dots, x_{i-1}^{(t)}, x_{i+1}^{(t)}, \dots, x_n^{(t)})$$

Jacobi Update The coordinates are updated with the previous solution simultaneously, i.e. we do not care about intermediary iterates until we complete all coordinates. The Jacobi update has a unique advantage: since only previous iterates are used, which are always known, it can be run in parallel for each coordinate. It is feasible for large configurations of turbines to cluster them into smaller groups and implementing a two-step method. After the turbines are grouped, we run a Gauss-Seidel update on the small groups. Once those values are returned, we update the wind farm as a whole using the Jacobi update. This allows for extensions of our algorithm to larger configurations of turbines.

Updating the Order of Index of Each Iteration

There are several ways to update the order of which index during each iteration.

- **Cyclic Order**: Index chosen is $1 \to 2 \to 3 \to \dots \to n$ at each iteration
- **Random Sampling**: Randomly select some i (this selection can be uniform or not)
- **Gauss−Southwell**: At each iteration, pick i so that $i =_{i \leq j \leq n} |\nabla_j f(x^{(t)})|$
- **Random Permutation Communication**: Cyclic order on a permuted index
- **Double Sweep**: Index chosen is $1 \to 2 \to \dots \to n$ then $(n-1) \to (n-2) \to \dots \to 1$
- **Almost Cyclic**: Each coordinate is picked at least once every $B \geq n$ successive iterations

Depending on the configuration of turbines, any different method of updating the indices become prevalent. For example, for strings of turbines it is simplest to implement a reverse cyclic order.

Coordinate maximization offers certain useful properties. To begin, the function values are nondecreasing: $f(x^{(0)}) \leq f(x^{(1)}) \leq \ldots$. Additionally, if f is strictly concave and smooth, it converges to a global maximum (optimal solution [13]). However, if f is only locally concave, it may converge to a local maximum if the starting point is near the local maximum.

Additionally, knowledge about differentiability of the function f allows for further exploitation of this algorithm. By using gradient methods to update each coordinate, one can improve the rate of convergence. These are known as coordinate gradient ascent methods [14].

Gradient Methods

There exist many gradient methods designed to solve optimization problems. A short list of some of the most common examples includes:

- Steepest gradient ascent
- Nesterov′s accelerated gradient ascent
- Coordinate ascent
- Frank-Wolfe method
- Subgradient methods
- Primal-dual methods
- Stochastic and incremental gradient methods

A brief discussion of some of these methods is included next.

Steepest Gradient Ascent

Gradient ascent (otherwise known as steepest ascent) is a first-order iterative optimization algorithm. To find a local maximum of a function using gradient ascent, the function follows a series of points in the direction of the gradient of the function at the current point. Given a starting point $x_0 \in dom f$, and step-size $\gamma < 0$, it works as follows,

$$x_{t+1} = x_t - \gamma \nabla f(x_t), t = 0, 1, 2, \dots \tag{4.1}$$

until a stopping criterion is satisfied.

Gradient descent direction is easier to calculate, and performing a line search in that direction is a more reliable progress toward an optimum. However, for some optimization problems, gradient descent is slow near the optimal point. Its rate of convergence is inferior to many other methods.

Newton Method

Newton's method is applied to the derivative ∇f of a twice-differentiable function f.

$$x_{t+1} = x_t - [\nabla^2 f(x_t)]^{-1} \nabla f(x_t), t = 0, 1, 2, \dots \tag{4.2}$$

Newton's method is relatively expensive in that you need to calculate the Hessian on sequential iterations. For well-behaved functions, Newton's method can be computationally less intense. There also exist methods that only slightly update the prior iteration's Hessian. These are known as quasi-Newton methods.

Stochastic Gradient Ascent

In stochastic gradient ascent we iteratively update our weight parameters in the direction of the gradient of the reward function until we have reached a maximum. Unlike traditional gradient ascent, only a subset of the entire dataset is used to compute the gradient at each iteration. Instead, at each iteration we randomly select a single data point from our dataset and move in the direction of the gradient with respect to that data point. This is only an approximation of the true gradient but it can be proven that this algorithm will eventually converge.

4.2 Algorithm

Suppose we have N turbines in an array. Since the rearmost turbines' control values are fixed if no other turbines are downstream, we set $u_N = u^\star = \frac{1}{3}$. We then consider the next turbine and search over possible values of u_{N-1} to find an optimal solution for the two turbine pair $\{N-1, N\}$. This solution is optimal for the two turbine case, since we are solving a concave function of a single variable. However, since there are likely other turbines in the string, this solution will only approximate the optimal solution for the entire array. We continue appending turbines to the front of the array and solving for optimal solutions assuming that the prior axial induction factors are fixed. This initial set of axial induction factors provides a reasonable estimate of the true solution, as downstream turbine effects are observed, but is not a global solution as upstream turbine effects are disregarded.

Following this initialization of axial induction factors, the goal is to search for improved locally optimal solutions that exist near those values. Since downstream effects were taken into account during initialization, we are likely to converge to global optimal solutions. In order to guarantee convergence, we apply a traditional coordinate ascent method on our power function $P_{total}(u_i...u_n)$ for the entire system, using the initial axial induction factors derived from initialization.

In a similar order to the initialization, we begin at the rearmost turbine and locally optimize that axial induction value assuming the remainder of the array is fixed. This function is concave so we either return the same value or find a new axial induction factor that increases the value of $P_{total}(u_i...u_n)$. Thus, at each iteration, we have a monotonically nondecreasing function in $P_{total}(u_i...u_n)$. Since the function is bounded by above, i.e. the energy extraction of a single wind farm is finite, we are able to guarantee convergence of our algorithm.

This procedure to determine axial induction factors is for a fixed wind direction and magnitude. Under varying wind conditions, the overlap between turbine wakes would need to be updated. Simulations demonstrate that our algorithm is fairly efficient, and thus can rapidly adapt to wind changes. A changing wind direction redefines which turbines are downstream and the cross-sectional overlap of the wakes, and a varying wind velocity influences the magnitude of that effect.

The mathematical representation of our algorithm is shown in Algorithm 1. In simulations, C_i is not an input, but directly calculated given a configuration of turbines, wind velocity and direction.

Theorem 1. *Given a continuously differentiable function $P_{total}(u_i, \dots u_n)$ over the closed and bounded set defined by the optimization problem in Equations (2.7) and (2.8), the nondecreasing sequence $P^j_{total}(u_i...u_N)$ at iteration j provided by ICyCA-WF converges to a local maximum solution.*

Algorithm 1: Initialized Cyclic Coordinated Ascent for Wind Farms (ICyCA-WF)

```
Data: $C_i, U_\infty$
Result: $u_i$ for all turbines
Step 1: Assign for rear turbines: $u_N^* = \frac{1}{3}$
Step 2: For all other turbines define $u_i = 0$
Step 3: Initialization
for $\forall$ turbines $t_i \notin$ rear turbines sequentially from $t_{N-1}$ to $t_1$ do
    Solve
    max $P_i(u_i) = \sum_{j=i}^{N} C_j u_j (1-u_j)^2 V_j^3$
    s.t. $0 < u_j \leq \frac{1}{3}$
    and $u_{j+1} \dots u_N$ fixed
end
Step 4: Update
for $\forall$ turbines $t_i \notin$ rear turbines sequentially from $t_{N-1}$ to $t_1$ do
    Solve
    max $P_{total}(u_i ... u_n) = \sum_{j=1}^{N} C_j u_j (1-u_j)^2 V_j^3$
    s.t. $0 < u_j \leq \frac{1}{3}$
    and $u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_N$ fixed from prior iterations.
end
Step 5: Update $u_i$ if and only if $P_{total}(u_i ... u_n)$ increases.
Step 6: Repeat Steps 4 and 5 until convergence.
```

To prove the theorem, we require the following two lemmas.

Lemma 1. *Under the assumptions of Theorem 1, the sequence $P_{total}^j(u_i ... u_n)$ provided by ICyCA-WF converges.*

Proof. Since the sequence $P_{total}^j(u_i ... u_n)$ is bounded from above (by Betz's law [15]), there exists a value M such that $|P_{total}(u_i ... u_N)| \leq M$ over our compact set of u_i. Further, since our sequence is nondecreasing, u_i are in a compact set and $P_{total}^j(u_i ... u_N)$ is continuous in u_i, there exists a maximum $P \leq M$, such that $P_{total}^j(u_i ... u_N) \to P$ as $j \to \infty$. Thus the sequence converges. [16] ■

Lemma 2. *Under the assumptions of Theorem 1, ICyCA-WF terminates at a local maximum.*

Proof. Given the properties of $P_{total}(u_i, \dots u_n)$ combined with Lemma 1, it suffices to show that the algorithm converges to a limit point that is a stationary point.

Denote $z_j^k = (u_1^k, \dots, u_j^k, u_{j+1}^{k+1}, \dots, u_N^{k+1})$, and let $\{u^k\}$ represent the sequence of axial induction factors generated by applying coordinated ascent. We see that

$$P_{total}(u^k) \leq P_{total}(z_1^k) \leq P_{total}(z_2^k) \dots \leq P_{total}(z_{N-1}^k) \leq P_{total}(u^{k+1}) \tag{4.3}$$

Let $\bar{u} = (\bar{u}_1, \ldots, \bar{u}_N)$ be a limit point of the sequence $\{u^k\}$. Equation (5.1) implies that the sequence of $\{P_{total}(u^k)\}$ converges to $\{P_{total}(\bar{u})\}$. Let $\{u^{k_j} | j = 0, 1, \ldots\}$ be a subsequence of $\{u^k\}$ that converges to $\bar{u}$. From the properties of coordinated ascent and (5.1), we have $P_{total}(u^{k_{j+1}}) \geq P_{total}(z_1^{k_j}) \geq P_{total}(u_1, u_2^{k_j}, \ldots u_N^{k_j})$.

Taking the limit as $j \to \infty$, we obtain

$$P_{total}(\bar{u}) \geq P_{total}(u_1, \bar{u_2}, \ldots \bar{u_N}) \tag{4.4}$$

The idea of the remainder of the proof is to show that $z_1^{k_j}$ converges to $\bar{u}$ as $j \to \infty$. If this is true, by symmetry we get that each $z_i^{k_j}$ converges to $\bar{u}$ as $j \to \infty$ $\forall i$, thus guaranteeing (5.2), and a unique maximum. Since our set of variables is compact, we note $z_1^{k_{j+1}}$ remains in our compact set $\mathcal{U}_1$. Thus a limit point, $\bar{\xi}$, is a minimizer of $P_{total}(u_1, \bar{u_2}, \ldots \bar{u_n})$. It is concave in each variable when all others are held fixed, so $\bar{\xi} = \bar{u_1}$. Thus we see that $z_1^{k_j}$ converges to $\bar{u}$. This argument holds for each variable, thus we see the point is stationary. A stationary point under our assumptions of a concave, nondecreasing function is a local maximum. ■

Simulations were implemented to compare the ICyCA-WF algorithm to other methods of solving the wind farm problem.

CHAPTER 5

SIMULATIONS

We can now compare the ICyCa-WF algorithm to other techniques via simulations. The three major properties that we will use to compare the methods are:

- Axial Induction Factor Structure
- Total Power Extracted
- Runtime

We can compare each method to what we solve using an exhaustive search. By taking the brute-force method to solve the problem (explicitly testing every possible value) we can observe the structure that should exist for the optimal set of axial induction factors. In general, the form of the solution is one that has the most upstream turbine that has a higher value (implying a faster rotation and greater power extraction) because it has no upstream effects. The central turbines are known to have similar values, and the rearmost turbine is set to $u_N = \frac{1}{3}$.

After solving for the different values, we can then compare the power extracted by plugging them into the equations defining the model and observing the runtime of the program. In theory, the runtime will be proportional to the theoretical computation complexity. The use of exhaustive search to solve for the optimal values of the axial induction factors requires fully exploring the $\mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_N$ control space. Given some level of precision where $\mathcal{U}_1$ is divided into k equally spaced partitions for each of the N turbines, to fully search the state space and identify the near-optimal values requires $\mathcal{O}(k^N)$ operations.

5.1 Seven-Turbine Array

We consider a string of seven turbines illustrated by Figure 6.1. This provides sufficient complexity for exposing some of the intricacies of wake interactions but also is simple enough that we can use exhaustive search over the $\mathcal{U}_1 \times ... \times \mathcal{U}_7$ range of axial induction factors to compare directly to the axial induction factors obtained from our algorithm for each turbine.

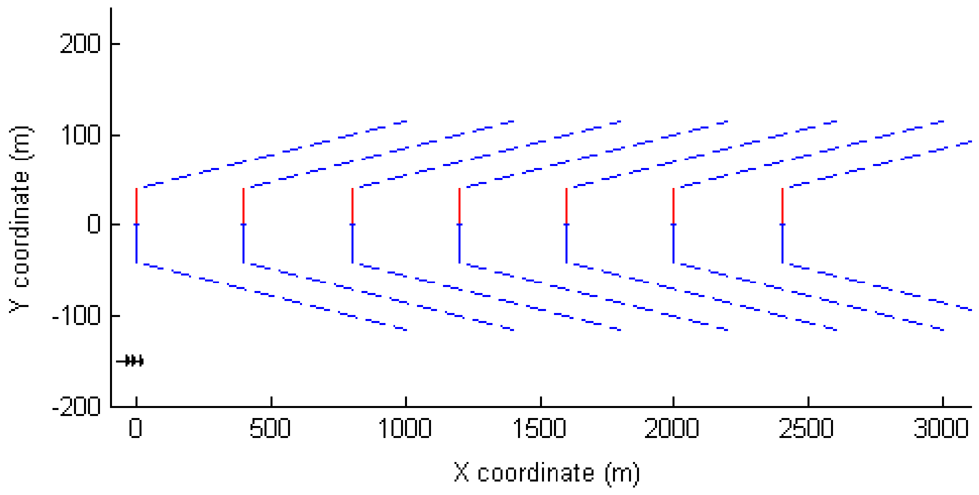

Figure 5.1: Seven-turbine array used for simulations.

For the seven-turbine array, a near-optimal solution for the axial induction settings, obtained by exhaustive search, is given by

$u_{opt} =$

$$\begin{bmatrix} 0.2260 & 0.1910 & 0.1940 & 0.1960 & 0.1990 & 0.2110 & 0.3333 \end{bmatrix}$$

All axial induction factors were initialized at $u_i = 0$, and the search space was gridded to resolution $\frac{1}{1000}$. The rearmost turbine is known a priori. Regardless of initialization, an exhaustive search has a fixed runtime in order to test every combination.

Using our heuristic algorithm, and iterating until termination, we determine the following axial induction factors for the seven-turbine array

$u_{array} =$

$$\begin{bmatrix} 0.2265 & 0.1856 & 0.1874 & 0.1925 & 0.1950 & 0.2056 & 0.3333 \end{bmatrix}$$

This set of axial induction factors provides a power ratio of $\frac{P_{ICyCA-WF}}{P^*} = 0.9944$ of the total power possible. P^* was obtained by an exhaustive search, which had a runtime of approximately 40 hours; our algorithm terminated in under 5 seconds.

Simulations were run in MATLAB on an Intel i5-3210M CPU at 2.5 GHz. We note that our approach yields a set of axial induction factors whose structure is similar to that of the exhaustive search. Prior works [10] focused exclusively on the near-field approximation lead to the following set of axial induction factors for the same array

$u_{near-field} =$

$$\begin{bmatrix} 0.0667 & 0.0769 & 0.0909 & 0.1111 & 0.1429 & 0.2000 & 0.3333 \end{bmatrix}$$

A graphical representation of the various results can be seen in Figure 6.2. We observe how the near-field approximation varies in result as more turbines are included in a string. Simulations and comparisons have also been completed for three-turbine strings

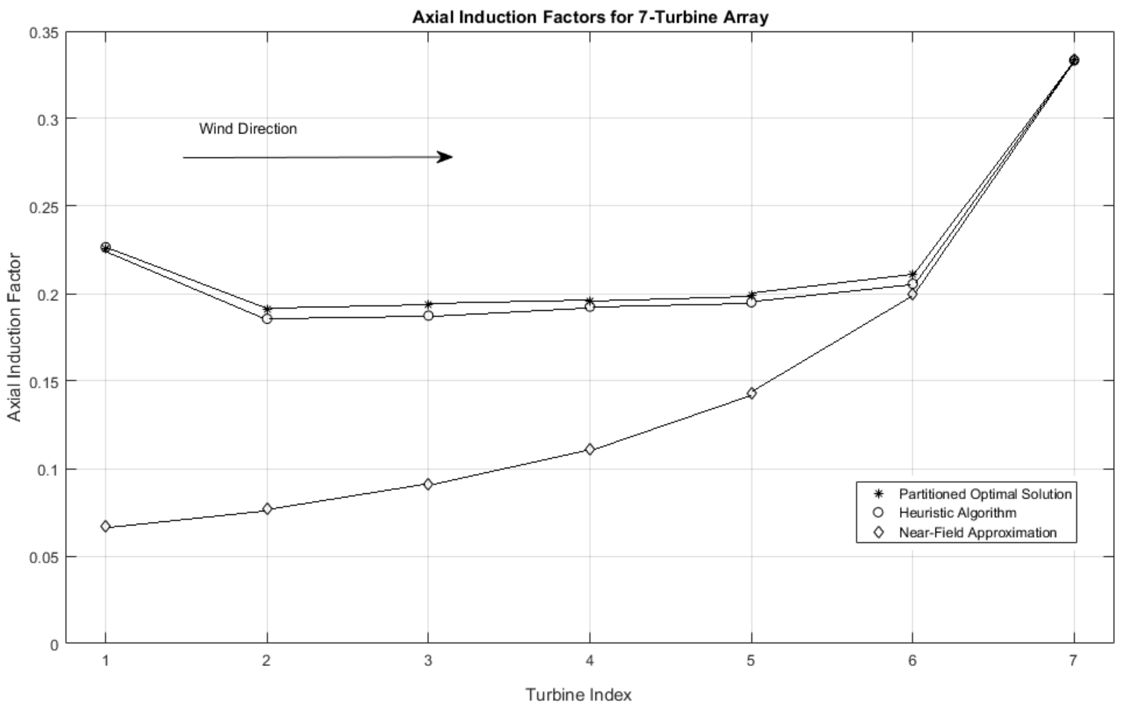

Figure 5.2: Graphical representation of axial induction factors for seven-turbine array from Figure 6.1 for different methods. The optimal solution is found via an exhaustive search.

with similar results. In addition, three-turbine arrays in which not all turbines have the same y-coordinate (which only influences p_{ij}) have also yielded promising results.

5.2 Three-Turbine Configurations

To further demonstrate the efficiency of our algorithm, we consider three configurations of three turbines: a string (Figure 2.4), an off-center string (Figure 6.3) and a tree (Figure 6.4).

In order to compare our algorithm, we use three additional metrics: *Greedy* (in which all u_i are set to $\frac{1}{3}$), *Locally Selfish* (where turbines upstream are maximized with no regard to downstream turbines), and *Exhaustive Search* (where the control space is discretized and all possible values are explored). The exhaustive search result is known to be an approximately optimal solution for (2.7).

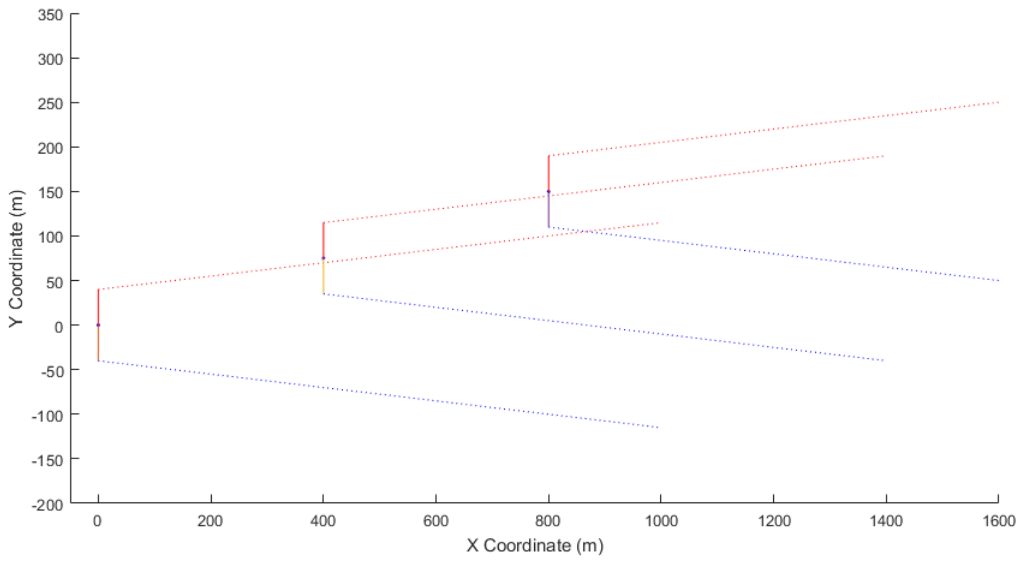

Figure 5.3: Three-turbine off-center string configuration.

5.2.1 Three-Turbine String

The simple example given in Chapter 2 is an excellent example to demonstrate the issues with greedy and locally selfish metrics for axial induction factor assignments. The results of the three metrics, along with our algorithm and the near-field solution are given in Table 6.1.

Table 6.1 Axial Induction Factors for Three-Turbine String

Algorithm	Axial Induction Factor $[u_1, u_2, u_3]$
$u_{Greedy} =$	$[\ 0.3333\quad 0.3333\quad 0.3333\ ]$
$u_{Selfish} =$	$[\ 0.3333\quad 0.2085\quad 0.1694\ ]$
$u_{Near-Field} =$	$[\ 0.1429\quad 0.2000\quad 0.3333\ ]$
$u_{Exhaustive} =$	$[\ 0.2320\quad 0.2080\quad 0.3333\ ]$
$u_{ICyCA-WF} =$	$[\ 0.2320\quad 0.2080\quad 0.3333\ ]$

We can also compute and compare total power extracted as shown in in Table 6.2.

Table 6.2 Power Extracted and Runtimes for Three-Turbine String

Algorithm	Total Power Extracted	Runtime
Greedy	3,260 KW	0.0041 sec
Selfish	3,699 KW	0.0334 sec
Near-Field	2,610 KW	0.0271sec
Exhaustive	3,772 KW	10.8244 sec
ICyCA-WF	3,768 KW	0.0343 sec

Clearly both the greedy and selfish solutions deviate in structure from the optimal solution, and the fact that the near-field case ignores the interaction between turbines 1 and 3 leads to a suboptimal solution. The slight discrepancy in resulting power values between our algorithm and the exhaustive search solution is attributed to small rounding errors that propagate in the power equation. As the number of turbines increases, exhaustive search becomes infeasible.

5.2.2 Three-Turbine Tree

The results for a tree-like configuration are given in Table 6.3.

Table 6.3 Axial Induction Factors Obtained for Three-Turbine Tree

Algorithm	Axial Induction Factor $[u_1, u_2, u_3]$
$u_{Greedy} =$	$[\ 0.3333\quad 0.3333\quad 0.3333\ ]$
$u_{Selfish} =$	$[\ 0.3333\quad 0.2717\quad 0.2717\ ]$
$u_{Exhaustive} =$	$[\ 0.2360\quad 0.3333\quad 0.3333\ ]$
$u_{ICyCA-WF} =$	$[\ 0.2360\quad 0.3333\quad 0.3333\ ]$

Table 6.4 Power Extracted and Runtimes for Three-Turbine Tree

Algorithm	Total Power Extracted	Runtime
Greedy	4,522 KW	0.0062 sec
Selfish	4,606 KW	0.0327 sec
Exhaustive	4,657 KW	11.0646 sec
ICyCA-WF	4,652 KW	0.0131 sec

Again we note convergence to the solution of the exhaustive case, but with much faster termination times as seen in Table 6.4. For this case, ICyCA-WF converges faster than the selfish algorithm, as the algorithm detects the two rearmost turbines and thus assigns the axial induction factors immediately to $\frac{1}{3}$.

5.2.3 Three-Turbine Off-Center String

We now consider the off-center string and note the final axial induction factors in Table 6.5.

Table 6.5 Axial Induction Factors Obtained for Three-Turbine Off-Center String

Algorithm	Axial Induction Factor $[u_1, u_2, u_3]$
$u_{Greedy} =$	$[\ 0.3333 \quad 0.3333 \quad 0.3333\ ]$
$u_{Selfish} =$	$[\ 0.3333 \quad 0.2818 \quad 0.2818\ ]$
$u_{Exhaustive} =$	$[\ 0.2920 \quad 0.2800 \quad 0.3333\ ]$
$u_{ICyCA-WF} =$	$[\ 0.2910 \quad 0.2810 \quad 0.3333\ ]$

We can compare total power extracted and runtimes in Table 6.6.

Table 6.6 Power Extracted and Runtimes for Three-Turbine Off-Center String

Algorithm	Total Power Extracted	Runtime
Greedy	4,673 KW	0.008 sec
Selfish	4,712 KW	0.0269 sec
Exhaustive	4,728 KW	12.6473 sec
ICyCA-WF	4,727 KW	0.0223 sec

5.3 Six-Turbine Tree

To demonstrate the resiliency of our algorithm, we consider the six-turbine tree in Figure 6.4.

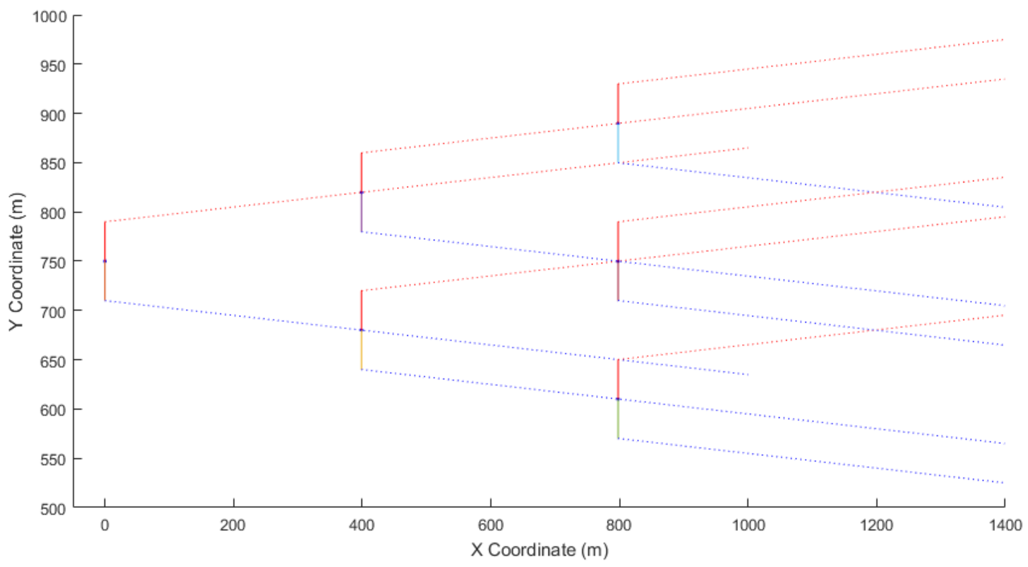

Figure 5.4: Six turbines arranged in a tree formation.

We immediately note how the exhaustive search algorithm terminates at basically the same values. The differences come primarily from rounding errors.

$u_{Exhaustive} =$

$$\begin{bmatrix} 0.2170 & 0.2430 & 0.2440 & 0.3333 & 0.3333 & 0.3333 \end{bmatrix}$$

$u_{ICyCA-WF} =$

$$\begin{bmatrix} 0.2160 & 0.2440 & 0.2440 & 0.3330 & 0.3330 & 0.3330 \end{bmatrix}$$

We compare total power extracted and runtimes for all four methods. Comparing these results to the prior ones we see as the number of turbines increases the runtime grows exponentially.

Table 6.7 Power Extracted and Runtimes for Six-Turbine Tree

Algorithm	Total Power Extracted	Runtime
Greedy	7,851 KW	0.011 sec
Selfish	8,268 KW	0.262 sec
Exhaustive	8,656 KW	313.39 sec
ICyCA-WF	8,643 KW	0.4201 sec

REFERENCES

[1] A. W. E. Association, “Wind industry annual market report 2015,” 2015. [Online]. Available: http://www.awea.org/2015-market-reports

[2] J. S. Gonzlez, M. B. Payan, and J. R. Santos, “Optimal control of wind turbines for minimizing overall wake effect losses in offshore wind farms,” EuroCon, 2013.

[3] A. Bonanni, T. Banyai, B. Conan, J. VanBeeck, H. Deconinck, and C. Lacor, “Wind farm optimization based on cfd model of single wind turbine wake,” Wind Energy, 2015.

[4] M. A. Ahmad, S. Azuma, T., and Sugie, “A model-free approach for maximizing power production of wind farm using multi-resolution simultaneous perturbation stochastic approximation,” Wind Turbines, 2014.

[5] R. E. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton University Press, 1957.

[6] A. C. Aranake, V. K. Lakshminarayan, and K. Duraisamy, “Assessment of low-order theories for analysis and design of shrouded wind turbines using cfd,” in *Journal of Physics: Conference Series*, vol. 524, 2014, p. 524.

[7] J. Marden, S. D. Ruben, and L. Pao, “A model-free approach to wind farm control using game theoretic methods,” 2013, pp. 1207–1214.

[8] D. R. VanLuvanee, “Investigation of observed and modeled wake effects at horns rev using windpro,” 2012. [Online]. Available: http://www.mek.dtu.dk/ /media/Institutter/Mekanik/Sektioner/FVM/uddannelse/eksamensprojekt/mastertheses

[9] J. Annoni, P. Gebraad, A. Scholbrock, P. Fleming, and J. W. van Wingerden, “Analysis of axial-induction-based wind plant control using an engineering and a high-order wind plant model,” *Wind Energy*, vol. 19, pp. 1135–1150, 2016.

[10] E. Bitar and P. Seiler, “Coordinated control of a wind turbine array for power maximization,” in *American Controls Conference*, Washington DC, 2013.

[11] M. Rotea, “Dynamic programming framework for wind power maximization,” in *The International Federation of Automatic Control*, Cape Town, South Africa, Aug. 2014.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization.* New York, NY: Cambridge University Press, 2004.

[13] S. Adlakha, R. Johari, G. Weintraub, and A. Goldsmith, "Oblivious equilibrium: an approximation to large population dynamic games with concave utility," in *Proceedings of the Workshop on Game Theory for Networks*, Istanbul, Turkey, 2009, pp. 68–69.

[14] Y. Nesterov, "Efficiency of coordinate descent methods on hugescale optimization problems," in *SIAM Journal on Optimization*, Jan. 2010, pp. 341–362.

[15] A. Betz, *Introduction to the Theory of Flow Machines.* London: Pergamon Press, 1966.

[16] D. P. Bertsekas, *Nonlinear Programming.* Cambridge, MA: Athena Scientific, 1999.