Wind Farm Layout Optimization using Population Distributed Genetic Algorithms

Helene Tangen

NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Science
Faculty of Information Technology, Mathematics and
Electrical Engineering
Trondheim, December 2015

Problem Description

This master thesis is an investigation of how different genetic algorithm models perform when used to solve the wind farm layout optimization problem. It is investigated if population distributed genetic algorithms can obtain better results than traditional non-population distributed genetic algorithms.

- Assignment given: 20 January 2016

- Supervisor: Professor Keith Downing

Abstract

Wind turbine technology is a promising source of renewable energy. However, the potential of wind turbine technology can not be utilized if the wind farm layout is not efficient. The challenge with wind farm layout optimization is that finding the optimal layout in the huge search space of different layouts is hard, if not impossible, to do analytically. Genetic algorithm techniques have been applied in many fields to solve none-linear optimization problems, and has shown promising results within wind farm layout optimization. Even so, not much effort has been put into investigating whether different genetic algorithm models can improve the results.

In this project, it is investigated if population distributed genetic algorithms are able to solve the wind farm layout optimization problem better than traditional non-population distributed genetic algorithms. This project is a contribution to a wind farm layout optimization contest launched by the Genetic and Evolutionary Computation Conference (GECCO) 2016.

An advanced genetic algorithm containing four different genetic algorithm models was implemented for this project. One of the genetic algorithm models was a traditional genetic algorithm, and three were population distributed genetic algorithms. The program was connected to a wind farm simulator provided by GECCO 2016 so that the performance of the different genetic algorithm models could be investigated on realistic wind scenarios.

The results showed that the traditional non-population distributed genetic algorithm consistently obtained better results than the three population distributed genetic algorithms. While two of the population distributed algorithms showed promising results, the third seemed unfit for solving the wind farm layout optimization problem. It is concluded that when providing the different genetic algorithm models the same amount of computational resources, the population distributed genetic algorithms come in short.

Sammendrag

(This is a Norwegian translation of the abstract.)

Vindmølleteknologi er en lovende kilde til fornybar energi. Likevel kan ikke potensialet til vindmøller utnyttes hvis ikke posisjoneringen av vindmøller i vindmølleparken er optimalisert. Utfordringen med vindmølleposisjonering er at løsningrommet - antall mulige vindmølleposisjoneringer - er enormt, og det er ekstremt vanskelig, om ikke umulig, å finne den optimale posisjoneringen analytisk. Genetiske algoritmer har vist seg å være svært nyttige når de har blitt brukt til å løse ikke-lineære optimaliseringproblemer, og de har vist lovende resultater brukt i vindmølleposisjonering. Det er likevel lite forskning på hvorvidt populasjonsdistribuerte genetiske algoritmer kan forbedre resultater enn ikke-populasjonsdistribuerte algoritmer for vindmølleposisjonering.

I dette prosjektet har det blitt undersøkt om populasjonsdistribuerte genetiske algoritmer kan løse vindmølleposisjoneringsproblemet bedre enn den tradisjonelle genetiske algoritmen. Prosjektet er et bidrag til en konkurranse for vindmølleposisjonering lansert av konferansen GECCO (the Genetic and Evolutionary Computation Conference) 2016.

En avansert genetisk algoritme bestående av fire forskjellige modeller var implementert for dette prosjektet. En av de genetiske algoritmene var en tradisjonell genetisk algoritme, mens de tre andre var populasjonsdistribuerte genetiske algoritmer. Programmer ble koblet sammen med GECCO 2016's vindfarmsimulator slik at de forskjellige genetiske algoritmene kunne bli observert og sammenlignet på forskjellige vindsenarioer.

Resultatene viste at den tradisjonelle genetiske algoritmer konsekvent fant bedre vindmølleposisjoneringer enn de tre populasjonsdistribuerte modellene. To av de populasjonsdistribuerte algoritmene gav lovende resultater, mens den tredje viste seg å være uegnet til å finne tilfredstillende vindmølleposisjoner. Resultatene ledet til konklusjonen at populasjonsdistribuerte algoritmer ikke klarer å levere like gode resultater som den tradisjonelle genetiske algoritmen når hver modell er tildelt de samme resursene.

1

Preface

This project is a master thesis written at the Department of Computer and Information Science, Norwegian University of Science and Technology.

I wish to thank my supervisor Professor Keith Downing at the Department of Computer and Information Science for his invaluable guidance and support throughout this project.

Contents

Chapter 1

Introduction

This thesis is a contribution to the third edition of the wind farm layout optimization contest launched by the Genetic and Evolutionary Computation Conference (GECCO) 2016. The contest involves optimizing the number of turbines and turbine positioning in a wind farm with goal of producing maximum power to minimum cost using genetic algorithms. A wind farm simulator is provided by GECCO 2016, the focus of this thesis will therefore be on improving the genetic algorithm that will be used to search for the optimal wind farm layout by investigating the state of the art within wind farm layout optimization and genetic algorithms and experiment with variations of the genetic algorithm.

This chapter provides an introduction to this thesis. Section 1.1 contains the background and motivation behind the thesis. Section 1.2 introduces the goal and research questions that will be investigated. In section 1.3 a brief overview of the research method is given. Contributions is given in section 6.3, and an overview of the thesis structure is given in section 1.5.

1.1 Motivation and Background

Transitioning from non-renewable energy sources to renewable energy sources is one of the largest, if not the larges political challenge of today. Renewable energy is less polluting than non-renewable energy and should therefore be preferred. However, renewable energy sources make up only 21% of the world's energy sources as of 2011 [EIA, 2015]. Wind turbine technology is

a promising source of renewable energy. Wind turbine technology advances have led to wind turbines able to produce more energy to lower costs. Despite these improvements, wind turbines still produce less energy than predicted because of the wake effect; reduction in wind speed caused by turbines placed in front of other turbines [Samorani, 2013]. For wind energy to become a bigger player in the world's energy sources, sophisticated methods for wind turbine placement in wind farms need to be developed so that each turbine produces as much energy as possible.

Wind turbine positioning is hard to optimize analytically. Fortunately, a wide variety of local search methods and bio-inspired methods have shown promising results, with genetic algorithms being the most popular method. As more advanced approaches to evaluate layouts has been developed, and more realistic constraints are introduced, more sophisticated genetic algorithms are required. To come up with more advanced genetic algorithms for solving the wind farm layout optimization problem, the annual Genetic and Evolutionary Computation Conference (GECCO) 2016, launched a competition where different contestants will provide their own implementation of a genetic algorithm [IRIT, 2015]. The goal of the competition is to bring more realistic problems to algorithm developers, and to create an open source library useful beyond the scope of the competition. Wind parameters and evaluation mechanisms are provided by GECCO 2016, therefore, the focus of this thesis will be on optimizing the genetic algorithm. Still, some knowledge of wind turbines, wind farms, and wind- and power models are crucial to understanding this thesis and will therefore be introduced in chapter 2.

Greedy heuristics, simulated annealing search, ant colony algorithms, particle swarm optimization and genetic algorithms have all been used in solving the wind farm layout optimization problem, and these will all be reviewed in chapter 3. Turbine positioning has been improved by genetic algorithms for more than 20 years, each approach bringing something new to the field such as a new type of genetic algorithm, a more realistic environment, or combinations of genetic algorithms and other approaches. Many researchers have solved the wind farm layout optimization problem by implementing a population distributed genetic algorithms called the Island model, and it has shown promising results. However, as far as the author knows, no attempt has been made in implementing any of the other existing population distributed models. This fact is the main motivation behind this thesis, and has inspired the

author to investigate the effect different population distributed algorithms can have on the wind farm layout optimization problem. Chapter 2 contains a review of the population distributed genetic algorithm investigated in this thesis.

1.2 Goal and Research Questions

This section states the goal statement and research questions that will be investigated in this thesis.

Goal statement

> *The project goal is to investigate the advantages of using population distributed genetic algorithms in optimizing wind farm layout, i.e. solving the wind farm layout optimization problem.* [Samorani, 2013]

The performance of population distributed genetic algorithms will be studied and compared to the performance of a simple genetic algorithm (non population distributed genetic algorithm) in order to answer the first research question. In addition, each of the population distributed genetic algorithms will be compared against the others with the goal of answering the second research question.

Research question 1

> *Can distributed genetic algorithms improve the quality of the solution to the wind farm layout optimization problem as compared to simple genetic algorithm.*

Research question 2

> *Which distributed genetic algorithm works best for the wind farm layout optimization problem? What properties are essential for its success?*

Both research questions will be answered by testing the different distributed genetic algorithms in a wind farm simulator provided by GECCO 2016. The

genetic algorithms will be implemented from scratch in order to give the author control of the environment and the possibility of adding functionality fast and easy. Each genetic algorithm will be tested on different wind scenarios which are also provided by the contest. Research question 1 will be answered by implementing 3 different types of population distributed genetic algorithms and compare their results with a simple genetic algorithm on different wind scenarios. Research question 2 will be tested the same way, and the results of each of the population distributed algorithms will be compared.

1.3 Research Method

In order to answer the research questions, a genetic algorithm was implemented for this thesis. Even though a simple genetic algorithm were provided by GECCO 2016 along with a wind farm evaluator and wind scenarios, the choice was made to implement the algorithm from scratch so that the population distributed genetic algorithms could be added easily, and so that more functionality could be added to the program.

To evaluate the different genetic algorithms, scenarios provided by GECCO 2016 were used. The scenarios provided contained wind parameters from different angles, terrain descriptions, and wind turbine parameters. First experiments were run to find parameter settings for the genetic algorithm suitable for the wind farm layout optimization problem, second the different genetic algorithms were run with those settings. The genetic algorithms were compared on four different wind scenarios and the results for each scenario is presented, a long with overall performance on all scenarios for each genetic algorithm. Finally, the results from the different simulations were compared and analyzed in order to to answer the research questions.

1.4 Contributions

The contribution to the field of wind farm layout optimization using genetic algorithms provided by this thesis is a comparison of different population distributed genetic algorithms and their effect on solving the problem. As far as the reader knows, no attempt has been made in comparing population distributed genetic algorithms on the wind farm layout optimization problem,

therefore this thesis can contribute with clearance as of which, if any, is the best choice for the given problem.

1.5   Thesis Structure

The thesis is divided into six chapters. Chapter 2 contains an introduction to the wind farm layout optimization problem, a description of the genetic algorithm, and a description of each of the distributed genetic algorithms that will be implemented. Chapter 3 is a survey of the state of the art within wind farm layout optimization using genetic algorithms and other approaches. The chapter also contains a discussion/conclusion of related work. Chapter 4 contains the methodology used to implement and test the genetic algorithms. In chapter 5 the results are presented and discussed. Finally, chapter 6 contains a conclusion.

Chapter 2

Background

This chapter contains the background for this thesis. In section 2.1 the wind farm layout optimization problem is defined, and challenges of wind farm construction are presented. The first part of section 2.2 gives an introduction to genetic algorithms, this sub section can be skipped if the reader is already familiar with genetic algorithms. The second part of section 2.2 explains the 3 distributed genetic algorithms that will be investigated in this thesis.

2.1   Wind Farm Layout Optimization

This section aims to give the reader an understanding of the wind farm layout optimization problem. It is divided into two parts. Part one formally defines the wind farm layout optimization problem. Part two gives a brief introduction to wind farm construction with the goal of providing the reader with an understanding of its complexity and an introduction the the main challenges of wind farm layout design.

2.1.1   The Wind Farm Layout Optimization Problem

An overview of the wind farm layout optimization problem is presented by Samorani [2013]. Grouping of wind turbines in a wind farm decreases installation- and maintenance costs. However, positioning of wind turbines in a farm also introduces new challenges. The power produced by wind turbines is largely dependent on wind speed, therefore it is important that the wind speed that hits a wind turbine is as large as possible. The main chal-

lenge for wind farms is that a wind turbine positioned in front of another will cause a wake of turbulence, meaning that the wind speed that hits the second wind turbine will be decreased. This effect is called *wake effect*, and will be explained in the subsequent sub section. Since the goal is to produce as much power as possible it is very important to position the wind turbines so that the wake effect is minimal. Samorani stated the wind farm layout optimization problem like this "The wind farm layout optimization problem consists of finding the turbine positioning (wind farm layout) that maximizes the expected power production". In this thesis, the problem formulation will be extended to include both cost constraints and optimization of the number of wind turbines, not just their positions. A formal definition is given below

> *"The wind farm layout optimization problem consists of finding the number of turbines and turbine positioning (wind farm) that maximizes the expected power production while minimizing costs."*

A typical solution to the wind farm layout optimization problem is given in figure 2.1. The rectangular grid represents the wind farm. Black cells represents positions containing wind turbines, and white cells represents empty positions. The wind farm layout in the figure is the optimal layout for a simple wind scenario where the wind is blowing at one direction with constant speed. When the wind is blowing from different directions with different wind speed the turbines will be more scattered around in the grid.

2.1.2 Challenges of wind farm construction

Samorani [2013] gives an overview of the main challenges of wind farm construction. First, a suitable site has to be found, meaning a site with good wind conditions. Wind power conditions are partitioned into 7 different wind power classes. With today's technology, sites with power class 4 or higher are considered suitable for hosting a wind farm. Even though the wind farm has the required wind conditions, it might not be suitable for hosting a wind farm because it might be far from the electronic grid, so that connecting it to it would be too costly, or it could require costly road work because current roads are not able to handle the transportation trucks. Second, land owners has to be contacted and convinced that hosting a wind farm on their land is a good idea. Land owners usually gets a percentage of the wind farm profit.
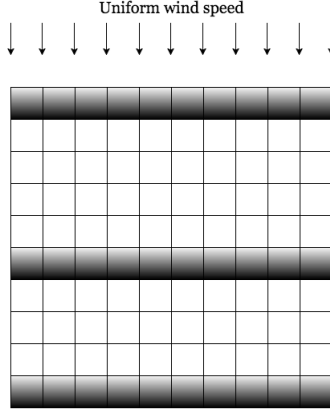
Figure 2.1: Optimal layout for a simplified wind scenario [Grady et al., 2005]. This figure is provided as an example of a wind farm layout.

This phase of contract negotiation usually takes a few months. At the same time, wind distribution need to be measured as accurately as possible. This step is extremely important, since the layout of the farm is optimized based on the measured wind distribution. Getting enough data to capture the wind distribution can take a few months if wind conditions are stable all year long, but if the wind conditions vary extensively over the year this step can take a few years.

An equally important step is to decide on which turbines to buy for the wind farm. A trade-off exists between power and cost since larger wind turbines are usually more expensive than smaller turbines, but they generally also produce more power. Realistic estimation of maintenance cost is also crucial in deciding on turbine type. In Samorani [2013] the number of wind turbines are also decided in this step, but in this project, deciding the number of turbines is included in the wind farm layout optimization problem and will therefore be part of the next step.

Only after the site is found, wind turbine type is decided on, and wind distribution is measured, can the layout optimization begin. Layout optimization faces different challenges. At some locations, the terrain is a great challenge because it contains areas which are unsuitable for wind turbines. These areas has to be dealt with by the layout optimization algorithm so that turbines

are only positioned in legal areas. There are also constraints on how close turbines can be positioned, according to Şişbot et al. [2010], the minimum spacing rule states that the minimum distance between turbines is 8D in prevailing wind direction, and 2D in cross wind direction, where D is the rotor diameter. Still, the greatest challenge of wind farm layout optimization is the wake effect. As mentioned above, the wake effect is the effect of reduced wind speed in the wake behind a wind turbine. Samorani explains the wake effect using the Jensen wake model [Jensen, 1983]. As will be apparent after chapter 3, most research in wind farm layout optimization use the Jensen model because it is quite accurate and simple. The Jensen model will be explained below in order to give a brief understanding of how the wake effect is calculated.
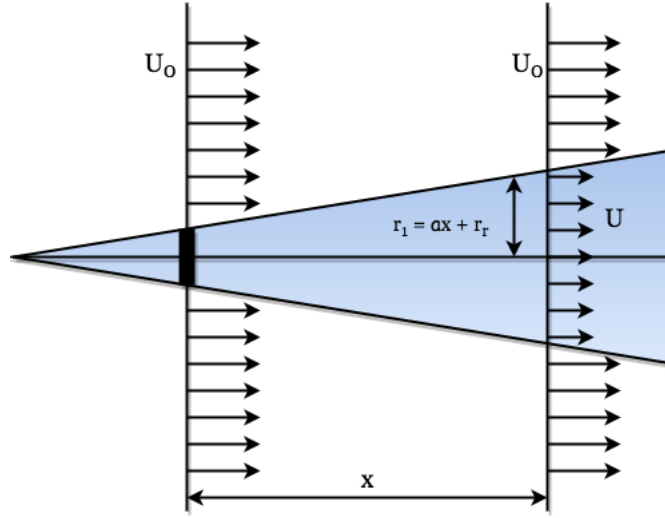
Figure 2.2: The wake effect [Samorani, 2013].

In figure 2.2 the small, black rectangle represents a wind turbine, and the blue area behind it illustrates the area that is affected by the turbulence created by the wind turbine. In the figure, the wind is blowing from left to right with uniform wind speed of U_0. As the wind hits the wind turbine it creates a wake of turbulence behind it so that the wind speed at distance x behind the wind turbine is $U < U_0$. The area behind the wind turbine that is affected by the wake at distance x has the radius $r_1 = \alpha x + r_r$ where r_r is

the rotor radius and α is the entrainment constant, a constant that decides how fast the wake expands. For a detailed, mathematical explanation of the Jensen model and other wake models see references [Jensen, 1983] and [Liang and Fang, 2014].

In summary, construction of a wind farm is a complicated and time consuming process. In order to only start the layout optimization process consecutive important decisions has to be made. The layout optimization is dependent on turbine cost, terrain parameters, wind conditions and turbine positioning. Finding the optimal layout is a non-linear, complex problem that only sophisticated algorithms can solve.

2.2 Genetic Algorithms

This section provides the background needed to understand the genetic algorithms implemented for this thesis. The first sub section gives a step-by-step introduction to simple genetic algorithms invented by Holland [1992]. If not otherwise stated this section is based on the books Holland [1992] and Goldberg [2005]. If the reader is familiar with genetic algorithms he or she can skip this sub section. The second sub section introduces the four genetic algorithms tested in this theses: The Master-Slave model, the Island model, the Cellular model and the Pool model [Gong et al., 2015]. The first model is a simple genetic algorithm implemented in parallel, the other three are population distributed genetic algorithms.

2.2.1 Simple Genetic Algorithms (SGAs)

Genetic algorithm are probabilistic search algorithms inspired by evolution. Figure 2.3 shows the five steps that constitute the genetic algorithm. The genetic algorithm operates on a population of individuals each representing a solution to a problem. The first step is therefore to generate the initial population, which usually consists of randomly generated individuals. The second step of the genetic algorithm is evaluation. The child population is evaluated based on some predefined fitness function (objective function) - a measure of the goodness of the solution. Note that the terms fitness function and objective function will be used interchangeably in this thesis. The third-

and forth step, adult selection and parent selection respectively, is the process of deciding which individuals from the child population are allowed to grow up into adults, and which are chosen to be parents for the next generation of individuals. These steps are naturally guided by the fitness of the individuals. The fifth and last step is called recombination. In this step, the genes of the parent population are recombined and altered in order to generate a new, hopefully better, child population. After the new child population is generated the process starts again and continues until some predefined stop condition is reached.
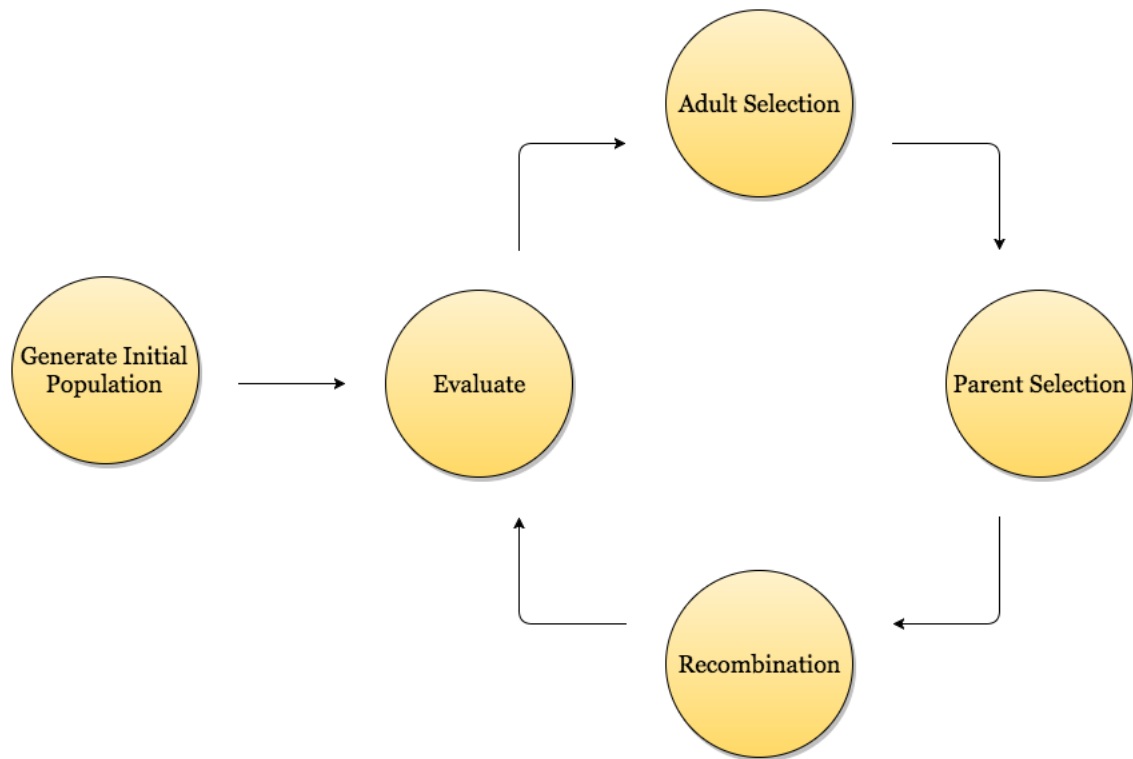

Figure 2.3: Overview of the five steps that constitute the genetic algorithm.

Inspired by survival of the fittest, the population evolves into a population of better solutions to the given problem. Two properties are crucial for the utilization and success of the genetic algorithm: (1) One have to know how the measure the fitness of the individuals (goodness of the solutions), and

(2) one have to find a way to represent individuals so that genetic operations can be performed on them. Examples of representation, fitness calculation, selection and genetic operations will be given below. Note that numerous different selection schemes and genetic operations exists and that the section below only gives one example of each in order to provide understanding of how a typical genetic algorithm works. How these steps are actually implemented in this thesis is explained in chapter 4.

Representation

In genetics, an organism's hereditary information is called its genotype, and its observable properties its phenotype. For example, the hereditary information in your genes (genotype) are responsible for your eye color (phenotype). The genetic search algorithm usually works on genotypes represented as bit strings. Goldberg [2005] explained this with a simple example. Let's say the objective function that we want to find the optimal solution for is x^2 for $x \in \{0, 31\}$. Then we can generate genotypes for the random solutions using bit strings of size 5, each representing a decimal value (phenotype) between 0 and 31. Figure 2.4 displays the genotype and phenotype for four randomly generated individuals. Here, the phenotypes are just the genotypes on decimal form, but in other problems the phenotype could be everything from eye color to a wind farm layout.

Genotype   Phenotype

0	1	0	1	1	11
1	0	0	1	0	18
0	0	1	1	0	6
0	0	1	0	1	5

Figure 2.4: Genotypes and phenotypes for four individuals where the phenotype is the decimal value of the genotype (binary number).

Selection

As mentioned above, selection is a two-step process: Adult selection and parent selection. Adult selection is the decision of choosing which children are allowed to grow up and enter the adult pool and thereby become potential parents for the next generation. The simplest form of adult selection is called *full generational replacement* and it consists of simply replacing the previous adult pool with the new child population. This adult selection scheme is one out of three adult selection mechanism implemented for this thesis. These will all be explained in detail in chapter 4.

Parent selection is the process of selecting which individuals from the adult population that will be the parents of the next child population. The simplest form of parent selection is to simply select the fittest individuals. Unfortunately, this selection strategy often leads to premature convergence of non optimal solutions. It is important to prioritize exploration, at least in the beginning of the search, otherwise, parts of the search space that could have lead to the optimal solution are cut off too soon. To cope with this problem *controlled elitist selection* schemes are preferred. A very popular selection strategy is *tournament selection* [Razali and Geraghty, 2011]. In tournament selection, groups of n individuals are randomly drawn from the population and the best (fittest) individual from the group is chosen as the tournament winner, and is therefore selected. Figure 2.5 illustrates how tournament selection works. In the example, n is equal to 3, therefore the three individuals with fitness 9, 4 and 6 are randomly drawn from the population. Since in this thesis, the best individuals are those with lowest fitness, the individual with fitness 4 wins the tournament and is chosen for reproduction.

By varying the value of n you can control how much exploration your algorithm should do. If n is equal to the population size only the best individuals are selected, and if n is equal to 1 the search is completely random. This means that low values of n leads to more exploration of the search space, something that can potentially lead the algorithm to better solutions. Higher values of n on the other hand will often lead to fast convergence to suboptimal solutions. These properties make it desirable to vary the value of n during the genetic search so that exploration is prioritized at the beginning of the search, while exploitation (making use of seemingly good solutions) is prioritized at the end. Tournament selection is one out of two parent se-
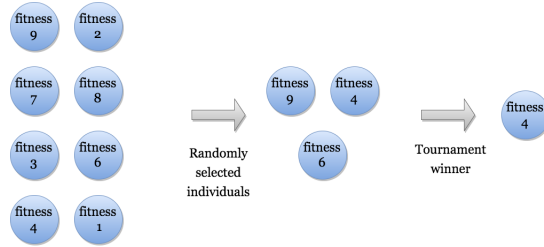
Figure 2.5: Tournament selection. A group of three individuals are randomly drawn from the pool of all individuals. The best individual in the group, the one with fitness 4, is selected for reproduction [Razali and Geraghty, 2011].

lection methods implemented in this thesis, these will also be explained in detail in chapter 4

Crossover

Crossover means combining genes of parent solutions to produce child solutions. One of the most common crossover schemes is called *uniform crossover*. For each gene of the child solution there is a 50% chance the gene will be copied from the first parent and a 50% chance that the gene will be copied from the second parent. Figure 2.6 shows how uniform crossover works. As can be seen, the first gene is taken from parent 1, the second gene from parent 2, the third and forth gene from parent 1 and so on. Uniform crossover is one out of three crossover methods implemented for this thesis.

Mutation

In biology, mutation is defined as permanent alteration in the DNA sequence that makes up a gene. When the genetic algorithm works on genotypes of bit strings the process consists of simply flipping bits. Mutation is usually implemented by having a given probability of each value in the genotype being flipped as shown in figure 2.7.

Mutation is important because without mutation a population can converge to a population of individuals where each genotype has the same value at a given position. Since every individual has the same value in their genotype,
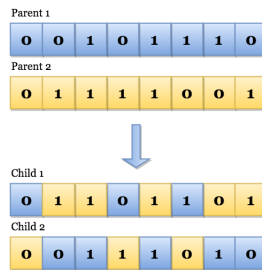
Figure 2.6: Uniform crossover. Two child genotypes are created by combining the genotypes of two parents. Each gene is drawn from one of the parents with equal probability.
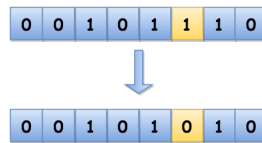

Figure 2.7: Mutation of a single bit. The bit in position 6 at the upper bit string has the value 1 before the mutation, while after mutation the value is flipped to 0.

reproduction will never be able to make a new individual that does not also have the same value at the same position. With mutation however, there is always a probability of the value being flipped, mutation is therefore crucial for maintaining diversity in the population; keeping it from becoming sterile.

Even though mutation is important, the probability of mutation needs to be kept low. If the mutation rate is very high, the genotype of a new individual will almost be a random bit string. Remember that a new individual is made by recombination of two individuals with high fitness in the previous population, if mutation changes the new individual heavily, it will not inherit the good features of its parents and the whole point of evolutionary search will be gone.

2.2.2 Population Distributed Genetic Algorithms

On of the main challenges of simple genetic algorithms is keeping diversity in the population long enough so that the population does not converge to a sub-optimal solution. By distributing the population, the population is able to explore different solution paths, even some that do not look that good at first, and consequently, probably find better, more sophisticated solutions. The goal of this thesis is to investigate and compare the performance of population distributed genetic algorithms with that of simple genetic algorithms. It will be distinguished between population distributed genetic algorithms and distributed genetic algorithms where processing is distributed among different processors, meaning that parallelism is utilized in the implementation. The first model introduced in this section, the master-slave model, is not population distributed. It is a simple genetic algorithm, implemented using parallelism in order to run fast, and it will be implemented so that the performance of a simple genetic algorithm (non-population distributed) can be compared to a population distributed algorithm. This section introduces 4 different distributed algorithms presented by Gong et al. [2015]. These will all be implemented and tested in this thesis.

Master-Slave Model

As mentioned above, the master-slave model is not a population distributed genetic algorithm, but a simple genetic algorithm where the main operations of the algorithm are distributed between different processors. It will be implemented in this thesis for two reasons: (1) distributing tasks between different processors gives a faster-running algorithm, (2) results obtained will be the same as results obtained by a simple genetic algorithm, and can therefore be used to compare against population distributed algorithms. The master-slave model is displayed in figure 2.8.

When the master-slave model is used, the main loop is taken care of by the master process, however the most expensive operation in the genetic algorithm, calculation of fitness, is distributed to different slave processes. Each slave simply calculate the fitness of the individuals received from the master, and return the calculated fitness to the master.
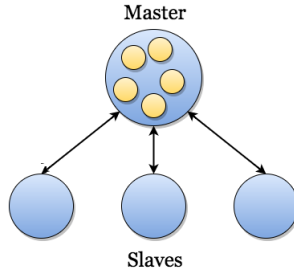
Figure 2.8: Master-slave model. The master process distributes the population to different slave processes, which calculate the fitness of each individual and return the results to the master process [Gong et al., 2015].

The Island Model

In the Island model, the population is divided into sub populations that are distributed onto different Islands. By letting each population evolve separately, different islands can explore different solutions. Figure 2.9 displays a population divided into four sub-populations.

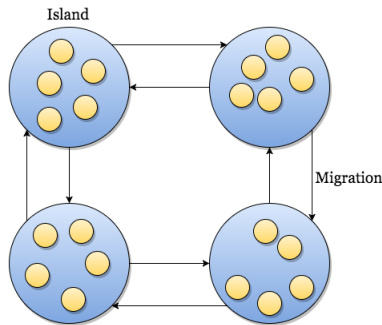Figure 2.9: An Island model using a ring topology with four demes (Islands) of size five. [Gong et al., 2015]

According to Huang [2007], six parameters must be specified when using the Island model. First of all, one need to decide on the number of demes (Islands). Second, the deme size needs to be specified; the number of individuals on each island. In figure 2.9 the deme size is five, and four demes are

used. Third, the topology must be specified; the allowed routes to migrate from one population to another. Numerous topologies can be used. In figure 2.9 the arrows represents legal migration routes. Since the topology forms a circle it is called a ring topology. The forth and fifth parameters listed by Huang are migration rate and migration interval, meaning the number of individuals that migrate from one population to another and the number of generations between each migration respectively. These parameters are very important since they largely affect the time the population gets to explore different solutions before the best solutions from some of the demes takes over the population. Sixth, the policy for emigrant selection, and how to replace existing individuals with new migrants needs to be specified.

The parameters listed above must be given careful thought when implementing the Island model, but as Gong explains, they are not the only ones. If the Island model is implemented in parallel one also have to decide if the migration is synchronous or asynchronous. Synchronous migration means that all migration is performed at the same time; at a specific generation. Asynchronous migration on the other hand, can be performed whenever one of the parallel processes are ready. Additionally, it has to be decided if the Island model is homogeneous or heterogeneous. By a homogeneous Island model, Gong et al. means an Island model where each sub population use the same selection strategy, genetic operations and fitness function, while as an heterogeneous Island model can implement different settings for different sub populations.

The Cellular Model

Figure 2.10 displays the cellular model from Gong et al. [2015]. In the cellular model the population is distributed in a grid of cells where each cell holds one individual. Each individual can only "see" the individuals of its neighborhood (as decided by the given neighborhood topology) and can only be compared with, and mate with individuals in its neighborhood.

The takeover time is defined as the time it takes for one individual to propagate to the whole population. The neighborhood topology largely affects the takeover time. In figure 2.10 the neighborhood topology is defined as only the individuals to the left, right, over and under the given individual. Since the topology includes a small number of individuals, the takeover time
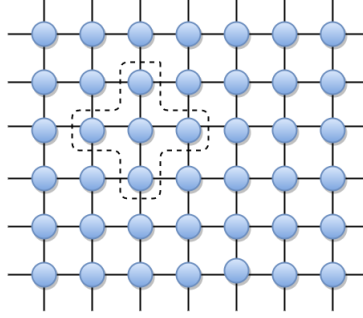
Figure 2.10: Cellular model where the neighborhood topology consist of the cells to the left, right, over and under the given cell [Gong et al., 2015].

will be long, meaning that exploration is prioritized. If the topology consists of a larger number of cells the takeover time will, off course, be much shorter.

The cellular model can also be implemented in parallel, ideally with one processor for each cell. As in the Island model, updating of the cells can be both synchronous and asynchronous.

Pool Model

Another population distributed model is called the pool model. In this model the population is put in a shared global pool of n individuals, where it can be accessed by different processors. Each processor draws a population from random positions in the pool, however it has allocated its own positions for which it can return individuals to the pool. This process is demonstrated in figure 2.11.

A processor p_1 is responsible for k positions in the pool. This is indicated by the coloring scheme in figure 2.11, where the processor and the positions it is responsible for has the same color. The processor draws a population of individuals $i_1, i_2, ..., i_k$ from random positions in the pool and performs genetic operations and fitness calculations on them. Next it writes each individual back to its corresponding position $1, 2, ..., k$ in the the positions it is responsible for, given that their fitness is higher then the fitness of the individual currently occupying the position.
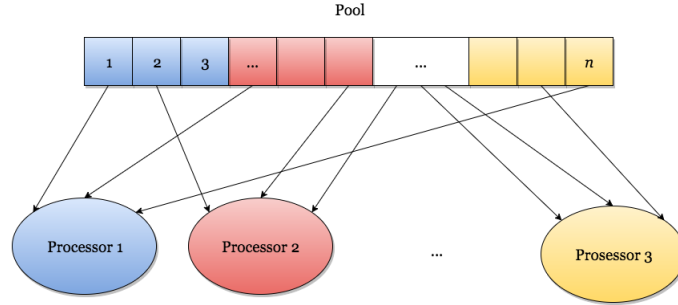
Figure 2.11: The pool model. Each processor has its own positions in the pool which it returns individuals to. The red processor is responsible for the red positions in the pool, and so on. Processors draw individuals from random positions in the pool, as indicated by the arrows, but can only write them back to its own positions, given that their fitness is higher than the fitness of the individual currently occupying the position [Gong et al., 2015].

In summary, this chapter has introduced the wind farm layout optimization problem and explained its complexity. A simple genetic algorithms has been introduced, with examples of selection-, crossover- and mutations methods. Last, the 4 distributed models that will be implemented has been presented. The master-slave model is a simple genetic algorithm implemented using parallelism. The Island model, the cellular model, and the pool model are population distributed parallel algorithms that will be compared against the simple genetic algorithm, and each other, in order to answer the research questions.

2.3 Structured Literature Review

In order to get a deep understanding of genetic algorithms the books Holland [1992] and Goldberg [2005] was read. By reading the initial work of Holland, the inventor of the genetic algorithm, and the in depth explanation and analysis presented by Goldberg, the author got the proper understanding of genetic algorithms needed to write this thesis.

After an extensive literature search, the author was able to come up with the goal and research questions that will be investigated in this thesis. The

literature search was performed in the following way: First, Google Scholar was used to search for articles with the search words *genetic algorithms* and *wind farm layout optimization*. A few of the first hits were read by the author. After reading these initial papers the author's understanding of the domain and domain vocabulary increased so new searches were made with the search words *genetic algorithm*, *evolutionary algorithm*, *genetic computation*, *wind farm layout optimization* and a few more. The resulting papers were collected by the author and review based on the following measurements: (a) Number of citations, (b) similarity of the problem to the problem investigated in this thesis, (c) whether the article brings something new to the field, and (d) publishing date. Articles published in resent years were preferred, a long with a few initial papers because they laid the foundation for the field. Twelve articles summarized in table **??** were selected to give the reader an extensive understanding of wind farm layout optimization using genetic algorithms. Additionally, four articles about wind farm layout optimization using other artificial intelligence algorithms were chosen to be presented in order to give the reader a brief overview over other approaches that has been used in solving the given problem.

In summary, this chapter has introduced the wind farm layout optimization problem and the challenges that comes with wind farm construction. The simple genetic algorithm has been explained in detail, and each of the 3 population distributed genetic algorithms have been introduce. The next chapter will give the reader an overview over the current research within wind farm layout optimization using genetic algorithms and a few other techniques.

Chapter 3

Related Work

Wind farm layout optimization has been studied extensively over the last 20 years and the goal of this section is to provide the reader with an overview of the research. This section is divided into three parts; Section 3.1 gives an extensive overview of wind farm layout optimization using the genetic algorithm, since genetic algorithms are the main focus of this thesis. Section 3.2 gives a short review of other optimization approaches, and section 3.3 contains a summary/discussion of related work.

3.1 Wind Farm Layout Optimization using Genetic Algorithms

Mosetti et al. [1994] were the first to successfully demonstrate the utilization of the genetic algorithm in solving the wind farm layout optimization problem. Although their work was made for illustrative purposes only, it laid the foundation for a number of more extensive studies of wind farm layout optimization using genetic algorithms.

In order to properly model a wind farm one have to specify a wake model, a power curve and a cost function. To model the wind decay, Mosetti et al. used a wake model similar to the one developed by Jensen [1983]. Power generated by each turbine i was modeled as a cubic function of the wind speed u and site roughness z_0, and summed up to get the total power produced by the farm in one year as shown in equation 3.1. This power model is called the Betz power model [Albring, 1967]. Cost was modeled as a simple function of

the number of turbines N_t, assuming the cost/year of a single turbine is 1, and that a maximum cost reduction of $\frac{1}{3}$ can be obtained for each turbine if a large number of machines are installed, as shown in equation 3.2

$$Power_{total} = \sum_{i}^{N_t} z_0 u_i^3, \tag{3.1}$$

$$cost_{total} = N_t \left(\frac{2}{3} + \frac{1}{3} e^{-0.00174 N_t^2} \right). \tag{3.2}$$

With the goal of producing a great amount of power at low cost, the objective function to be minimized was formulated as a combination of equation 3.1 and 3.2

$$Objective = \frac{1}{P_{total}} w_1 + \frac{cost_{total}}{P_{total}} w_2 \tag{3.3}$$

where w_1 and w_2 are weights. In the current study, w_1 was kept small so that the focus would be on lowest cost per energy produced.

Mosetti et al. divided the wind farm terrain into a 10×10 quadratic grid where a wind turbine could be installed in the middle of each cell. The optimization problem would then be to position turbines in cells in a way that maximize power production and minimize cost. With this representation, an individual of the genetic search could be represented as a binary string of length 100, where each index represents a cell in the grid, so that a value of 1 means that a wind turbine is installed in the corresponding cell, and a value of 0 means that there is no wind turbine in the corresponding cell. Figure 3.1 illustrates how an individual represents a wind farm for a wind farm partitioned into 100 cells. The genetic algorithm used was a simple, single-population genetic algorithm. The crossover operation was performed at random locations with probability $0.6 < P_c < 0.9$ and mutation was performed with probability $0.01 < P_m < 0.1$.

The model was tested using a single turbine type on three different wind scenarios; (a) single wind direction, (b) multiple wind directions with constant intensity, and (c) multiple wind directions and intensities. For each scenario, the results were compared to random configurations of 50 turbines.

| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | .... | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

1   Wind Farm Genotype   n
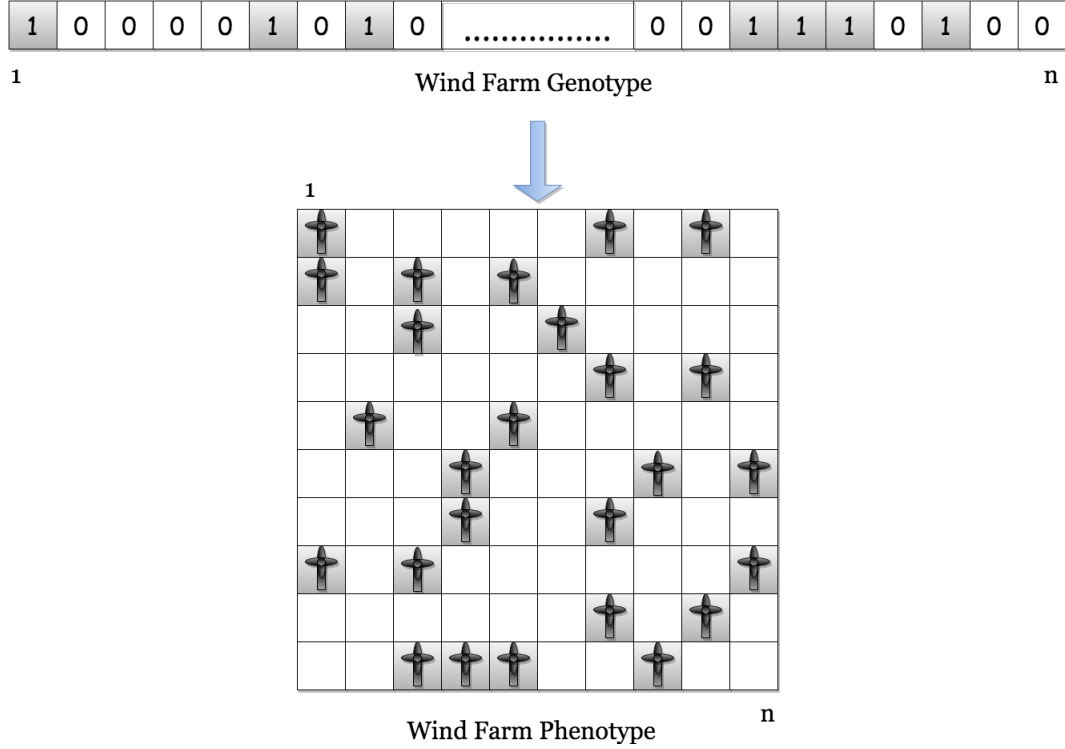

Wind Farm Phenotype

Figure 3.1: An example of how the wind farm is represented in the genetic search from Mosetti et al. [1994]. An individual is represented as a bit-string of size 100, where each cell can either contain the value 1 or 0, representing a position containing a turbine and a position not containing a turbine, respectively.

In scenario (a), the efficiency of the random configuration was 0.50, while the efficiency of the optimized solution was 0.95. In (b), the efficiency was increased from 0.35 in the random configuration to 0.88 in the optimized configuration. And, in (c) the efficiency was increased from 0.34 to 0.84. For each scenario the number of wind turbines was decreased drastically in the optimized version. Table 3.1 summarizes the results obtained.

As discussed in the publication, different simplifying assumptions were made in the model, such as the cost function, only one turbine type and the layout

Table 3.1: Optimized configurations compared against random configurations for each of the three scenarios (a) single wind direction, (b) multiple wind directions with constant intensity and (c) multiple wind directions and intensities[Mosetti et al., 1994].

Scenario	Configuration	Efficiency	P_{tot}(kWyear)	cost/kWyear	Number of turbines
(a)	Random	0.50	13025	2.57×10^{-3}	50
	Optimized	0.95	12375	1.57×10^{-3}	25
(b)	Random	0.35	9117	3.68×10^{-3}	50
	Optimized	0.88	8711	1.84×10^{-3}	19
(c)	Random	0.34	4767	7.04×10^{-3}	50
	Optimized	0.84	3695	3.61×10^{-3}	15

model. The results are also only compared against random configurations, not configurations optimized by other optimization approaches, and no attempt has been made to optimize the software. However, the purpose of this initial paper was to demonstrate the applicability of genetic algorithms on the wind farm layout optimization problem, and it has certainly laid the ground work for a number of studies performed over the last 20 years.

Grady et al. [2005] picked up where Mosetti et al. [1994] left of. They recognized that while the results of Mosetti et al. beats random configurations they were not close to beat configurations made by simple empirical placement schemes. In their study, they wanted to show that by implementing a population distributed genetic algorithm, the effectiveness of the algorithm could also be compared to optimal configurations. As Mosetti et al., they used the Jensen wake decay model, as well as the same cost- and power function, shown in equation 3.2 and 3.1 respectively. However, the objective function was changed into the following

$$Objective = \frac{cost}{P_{tot}}. \tag{3.4}$$

The same three scenarios as Mosetti et al. were considered. However, the number of individuals was increased from 200 to 600, and run for 3000 generations instead of 400. The population distributed model used was an Island model where the individuals were divided into 20 sub-populations. Sadly, not many implementation details were shared. On the first scenario, Grady et al. recognized that with uniform wind distribution the optimal solution could be obtained empirically by optimizing one single row of the layout, and

copy it to the rest. As opposed to Mosetti et al., their results was identical to the optimal solution. In scenarios (b) and (c) however, the optimal solutions could not be obtained empirically, and therefore the results are just compared against those of Mosetti et al. The results for each scenario is displayed in table 3.2. The first thing to notice is the difference in number of turbines. Grady et al. ends up with more turbines in each case, approximately doubling the number of turbines in scenario (b) and (c). The explanation behind this observation is in the objective function. Objective function 3.3 prioritizes low cost and hence prioritizes a lower turbine count. For each scenario the fitness of Mosetti et al. is higher than the fitness obtained by Grady et al. With exception of the first scenario, the efficiency is also higher in Mosetti et al. These results make sense since fewer turbines leads to less wake effect and decreased efficiency. However, in each case, the total power production is largely increased in the current study, which also makes sense based on the turbine count.

Table 3.2: Current results compared against the results from Grady et al. for each of the three scenarios [Grady et al., 2005].

Scenario	Parameter	Mosetti et al.	Grady et al.
(a)	Fitness	0.0016197	0.0015436
	Total power (kW year)	12 352	14 310
	Efficiency (%)	91.645	92.015
	Number of turbines	26	30
(b)	Fitness	0.0017371	0.0015666
	Total power (kW year)	9244.7	17220
	Efficiency (%)	93.859	85.174
	Number of turbines	19	39
(c)	Fitness	0.00099405	0.00080314
	Total power (kW year)	13 460	32 038
	Efficiency (%)	94.62	86.619
	Number of turbines	15	39

In summary, Grady et al. were able to show that by implementing a population distributed genetic algorihtm optimal solution for scenario (a) can be

obtained. They also showed that the power production obtained in Mosetti et al. could be increased by optimizing parameter values and using a more sophisticated implementation of the genetic algorithm. However, they make no attempt to compare their solutions for scenario (b) and (c) to solutions obtained using other optimization techniques. For a similar study where individuals are implemented as matrices in MATLAB see [Emami and Noghreh, 2010].

Zhao et al. [2006] presented a very interesting study, where the electrical system of an off shore wind farm on Burko Bank in Liverpool Bay was optimized using a genetic algorithm. Although this is a study of cable clustering design, with fixed wind turbine count and positions, it is very interesting because it is compared against actual results obtained by the Burbo project team. To get an understanding of the optimization problem four example clustering designs are presented in figure 3.2.
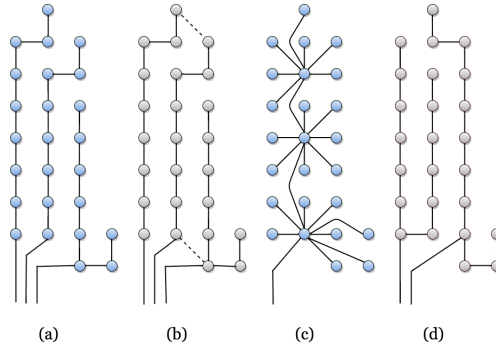

Figure 3.2: Four example clustering designs [Zhao et al., 2006].

In their publication, Zhao et al. present an extensive study of different genetic algorithm techniques to find out which performs best on this type of optimization problem. Premature convergence is discovered as the main problem of the genetic algorithm and to deal with this, different techniques are presented such as a diversity check, and a crowding technique called restricted tournament selection. For implementation details see reference [Zhao et al., 2006]. Different genetic algorithm designs were tested, and the results showed that the final design obtained was equal to the design obtain by the Burbo

31

project team! This shows that optimization using sophisticated genetic algorithm implementations can find the same solution as current optimization techniques for optimization of electrical systems.

Huang [2007] presented a study showing that a population distributed genetic algorithm performs better than a simple genetic algorithm. Huang uses a more realistic objective function than the previous studies, taking into account the selling price of electric energy, as well as cost and energy production. The distributed genetic algorithm used the Island model, with 600 individuals divided among 20 demes, with the ring-topology shown in figure 3.3.
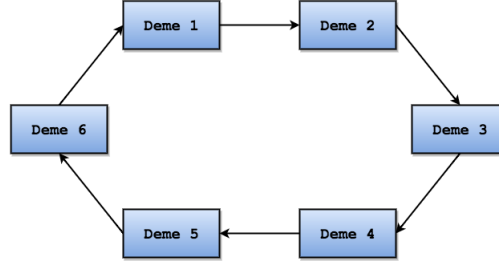

Figure 3.3: Ring topology example with 6 demes [Huang, 2007].

The simulation was run for 2500 generations with the migration strategy that 3.3% of the individuals with highest fitness was selected as migrants, to replace the individuals with lowest fitness in the new population every 20th generation. The distributed algorithm was tested using the same three scenarios as Mosetti et al. [1994] and Grady et al. [2005], and compared against a simple genetic algorithm. In case (a) the population distributed genetic algorithm was able to come up with the optimal solution (presented by Grady et al.), while as the simple genetic algorithm was not. For each of the three scenarios the population distributed algorithm ended up with higher fitness value, more power produced, lower CPU time and fewer generations. In case (a) the turbine count was equal for both algorithms, resulting in higher efficiency for the distributed algorithm. In scenario (b) and (c) the population distributed algorithm produced solutions with one more turbine than the simple algorithm, resulting in slightly lower efficiency.

All studies presented above have used binary encoding in their wind farm representation, but Mora et al. [2007] presented a study where the binary encoding was replaced by an integer encoding. In their approach, an individual was represented as a set of (x, y)-coordinates representing turbine positions. In addition to optimize turbine position, Mora et al. also wanted to optimize both turbine type and height. In order to do this, their individuals was represented by a matrix where the first row contained the x-coordinates of the turbines, the second row the y-coordinates, the third row turbine type and the forth row turbine height as shown in figure 3.4. Note that with this type of encoding, different individuals can have different lengths, depending on the number of turbines in each solution.

	Turbine 1	Turbine 2		Turbine k
X-Coordinate	X_1	X_2	...	X_k
Y-Coordinate	Y_1	Y_2	...	Y_k
Turbine type	T_1	T_2	...	T_k
Turbine height	H_1	H_2	...	H_k

Figure 3.4: Representation of an individual of length k (layout with k turbines), where the first row represents x-coordinates of the turbines, the second row y-coordinates, the third row turbine type, and the forth row turbine height [Mora et al., 2007].

Five crossover methods, and a masked mutation method were presented for the new type of encoding, see Mora et al. [2007] for implementation details. To model the wind speed, the Weibull distribution was used, a more realistic wind speed model than the one used in the previous studies. The Weibull distribution will be explained in more detail in chapter 4, since it is also used in the simulator for this thesis. Three different case studies are performed. The first one is a search for the optimal solution when the number of turbines are decided beforehand. The second is a search for the optimal positioning, type and height of turbines within a given budget. And the third one a search for the optimal solution with no such constraints. The results are only briefly discussed, however, this paper marks the shift from binary encoding to integer encoding, and from simple wind models to the Weibull distribution.

33

Wan et al. [2009] criticized the simple power-, and wind distribution model presented by references [Grady et al., 2005, Mosetti et al., 1994]. Rather, they used the Weibull distribution as [Mora et al., 2007] to model the wind, and they introduced a novel power model

$$\int_{u_{in}}^{u_{out}} P(u)f(u)du, \tag{3.5}$$

where u_{in} is the cut-in wind speed of the turbine, and u_{out} is the cut-out wind speed of the turbine. $P(u)$ is the power output for the wind speed u and $f(u)$ is the probability density of the wind speed u. The genetic algorithm was similar to that of Grady et al., and results show that the produced power increases.

One of the most complete studies of the wind farm layout optimization problem was done by Kusiak and Song [2010]. The study is based on six assumptions, which according to the authors are realistic and industrial-accepted. The study assume a fixed, predetermined turbine count, small variations of surface roughness, turbines with equal power curves, wind speed following the Weibull distribution, that wind speed at different locations with same direction share the same Weibull distribution, and last, it assumes that any two turbines must be separated with at least four rotor diameters. A multi-objective function was used to calculate the fitness of the solutions. It consisted of one objective function to maximize expected energy produced, and one to minimize the constraint violations. Kusiak et al. critiques Mosetti et al. and Grady et al. for not basing their wind energy calculation on the power curve function and not thoroughly discussing wind direction. Their work include an extensive model of wind energy based on a discretization of the expected power production for each wind direction. Their algorithm was tested on real wind data, and compared against an upper bound on power production (power produced without wake effect), and their results show that less than 2% of power is lost due to wake effects when 6 turbines are positioned in the wind farm.

Assumptions such as cost models only dependent on the number of turbines are unrealistic, and needs to be refined in order to model the wind farm layout optimization problem in an realistic way. González et al. [2010] introduced a cost model based on the net present value, which takes into account

wind speed, wind distribution, the number-, type-, rated power- and tower height of turbines, loss to due wake effects, auxiliary costs, road infrastructure, buildings, substation, electrical framework and financial aspects such as return on investment. For example, In order to more accurately model civil cost they present an greedy search which connects wind turbines to auxiliary roads or other turbines dependent on their position. Figure 3.5 shows how the greedy algorithm works. First, the distance between each turbine and every road is calculated. Since turbine A is closest to road 1, they are connected. Second, the distance between the remaining turbines and the roads and turbines already connected are compared, and turbine C is connected to road 2. At last, turbine B needs to be connected. Since it is closer to turbine A, than turbine C or any of the roads, it is connected to turbine A.
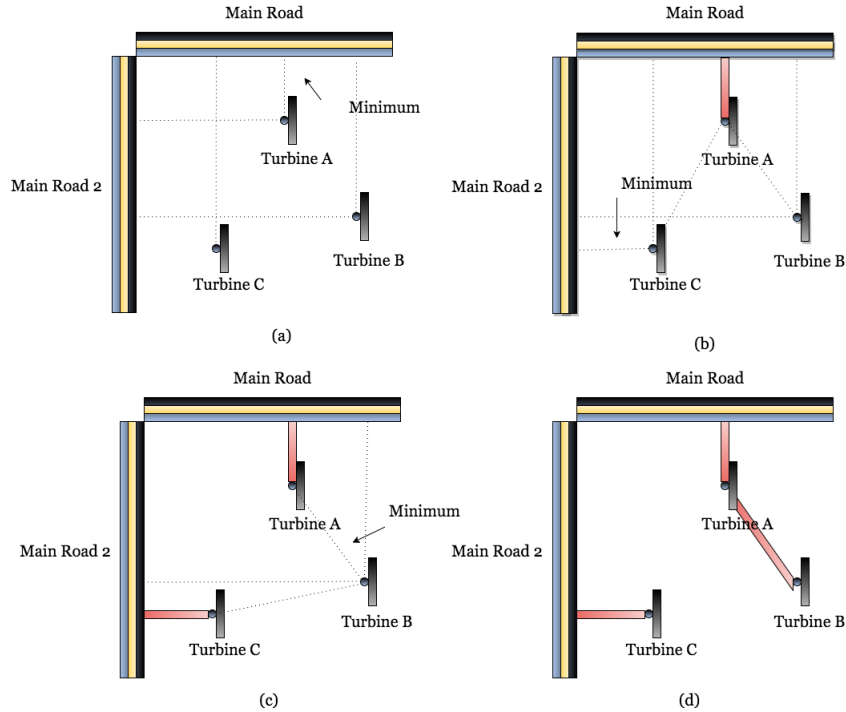

Figure 3.5: Greedy algorithm to estimate civil cost. [González et al., 2010].

Other new features includes a local search used when the genetic algorithm cannot find a better individual, and a genetic algorithm that can manage

forbidden areas and that gives penalties for turbines positioned in undesirable terrain. Individuals are represented the same way as in reference [Mora et al., 2007] displayed in figure 3.4. Results are compared against Grady, and shows higher produced energy. The authors also include three case studies showing how their algorithm can handle restrictions such as roads crossing, forbidden zones, undesirable zones and maximum investment cost.

Şişbot et al. [2010] published a case study of wind turbine placement on a wind farm at the Island Gökçeada, at the north east of the Aegean Sea. A distributed genetic algorithm was used, but unlike Huang, the individuals were evaluated based on multiple objective functions; one that measures the total cost (installation and operational), and one that measure total power production. Şişbot et al. argue that in an environment with changing demands, the use of a multi-objective function gives the decision-makers the opportunity to evaluate the different designs based on cost and power production separately, without ill-informed, randomly generated weights. The selection process used is a controlled, elitist process, meaning that not only the fittest, but also some individuals that can spread diversity to the population are selected for reproduction. The genetic algorithm returns a set of Pareto optimal solutions; a set of solutions that are not dominated by any other solution in the set. Stated more formally, solution $\mathbf{y}$ is said to dominate solution $\mathbf{x}$ if

$$\forall i : f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \text{ and } \exists j : f_j(\mathbf{x}) < f_j(\mathbf{y}) \tag{3.6}$$

where f_i is objective function [Murata et al., 2001]. Other interesting features of this study is the introduction of constraints on wind turbine positions and constraints on the cost, meaning that individuals with wind turbines outside the area of the island, and individuals with costs larger than the budget are removed from the population. Even though constraints on individuals are not in accordance with the nature genetic algorithms, they can be necessary when the algorithm is applied to a real problem. Another feature introduced in this paper is rectangular cells. The argument behind this decision is that the safe distance between wind turbines is dependent on the direction of the turbine. The minimum distance between turbines in prevailing wind is 8D, while the minimum distance between turbines in the crosswind is 2D. In spite of this attempt to make the wind scenario more realistic, it is critiqued because it operates with a constant wind direction

and constant speed, using the average wind direction and speed measured at the Island [Samorani, 2013]. Results are not compared against previous studies, and the argument behind this decision is that it is hard to compare a Pareto-optimal set of solution to one of the previous solutions.

Another very interesting solution to the wind farm layout optimization problem was proposed by Saavedra-Morena et al. [2011]. Four novel improvements were included in their model. First, a shape model was introduced to model the terrain shape. By introducing this model, the simplification of a square grid was lost, and every terrain shape could be implemented. Second, an orography model was used to model the wind speed on different heights. Using this technique, the wind model is much more realistic because it takes into account that wind speed differs at different heights. Third, they introduce a new cost model, which takes into account installation cost, connection between turbines, road construction and net benefit from the produced energy. The fourth, and maybe most exiting improvement presented was that a greedy heuristic was used to decide the initial positioning of the turbines for some of the individuals. This improvement was requested by Mosetti et al. in 1994, but not included by anyone until now. The greedy heuristic works by placing turbines one by one in the position with maximal wind speed. First of all, the first turbine is positioned in the position wind maximum wind speed. Next, the wind speed is updated because of the reduction in wind speed caused by the wake effect of the first turbine. Third, the second turbine is positioned in the position with maximal wind speed. This process continues until N wind turbines have been placed. Clearly, the resulting layout is largely influenced by the positioning of the first turbine, and leads to a sub-optimal solution on its own. However, it is much better than a random solution, and as it turns out a good starting point for the genetic algorithm. Results for 15 different orographys for the same terrain shape were provided, showing the objective function values obtained by the greedy heuristic, a simple genetic algorithm with random starting positions, and the seeded genetic algorithm (the genetic algorithm with starting positions provided by the greedy heuristic). In each case, the genetic algorithm with random starting positions beats the results obtained by the greedy heuristic, but more importantly, in each case, the seeded genetic algorithm beats the results of the simple genetic algorithm.

Both Mora et al. [2007] and González et al. [2010] used the genetic algorithm

to optimize the height of the turbines, as well as other parameters by representing individuals as shown in 3.4. Another approach to optimize turbine height was presented by Chen et al. [2013]. They state that normally, the same turbine type can be bought with several different heights, and that it therefore makes sense to use different height turbines. To optimize turbine position and height, they used two nested genetic algorithms. The first one was used to optimize turbine positioning, while the second one was used to decide between two turbine heights. For each generation of the first genetic algorithm, the second one was run for 50 generations to optimized turbine height. Binary encoding was used for individual representation in both genetic algorithms as shown in figure 3.6. The first binary string represent turbine positioning in the environment, while the second binary string represents turbine height for each position that contains a turbine. Several case studies were performed in the paper, and results show that turbine layout with different turbine height, produce more energy than same-height turbines every time.

Gao et al. [2015] also implemented a population distributed genetic algorithm to solve the wind farm layout optimization problem. Unfortunately, the implementation details have not been published. Integer encoding was used to represent individuals, but unlike those presented earlier [González et al., 2010, Kusiak and Song, 2010, Mora et al., 2007, Saavedra-Morena et al., 2011], each individual have the same number of turbines. The algorithm is tested on the same three scenarios from Mosetti et al. and compared against other studies [González et al., 2010, Grady et al., 2005, Mosetti et al., 1994, Wan et al., 2009]. For the comparison to be valid they force their solutions to use the same number of turbines as obtained in the results of the previous studies. In each case their resulting layout is able to produce more energy, and has higher efficiency, however, it never achieves the highest fitness. In addition to this comparison, Gao et al. introduce an interesting hypothetical case study of wind turbine placement on an offshore farm located in the Hong Kong southeastern water. By using real wind data, collected over 20 years, they demonstrate that the distributed genetic algorithm can be applied to a real-world wind farm layout optimization problem. The resulting wind farm layout was estimated to be able to produce 9.1% of yearly electrical consumption in Hong Kong (2012).
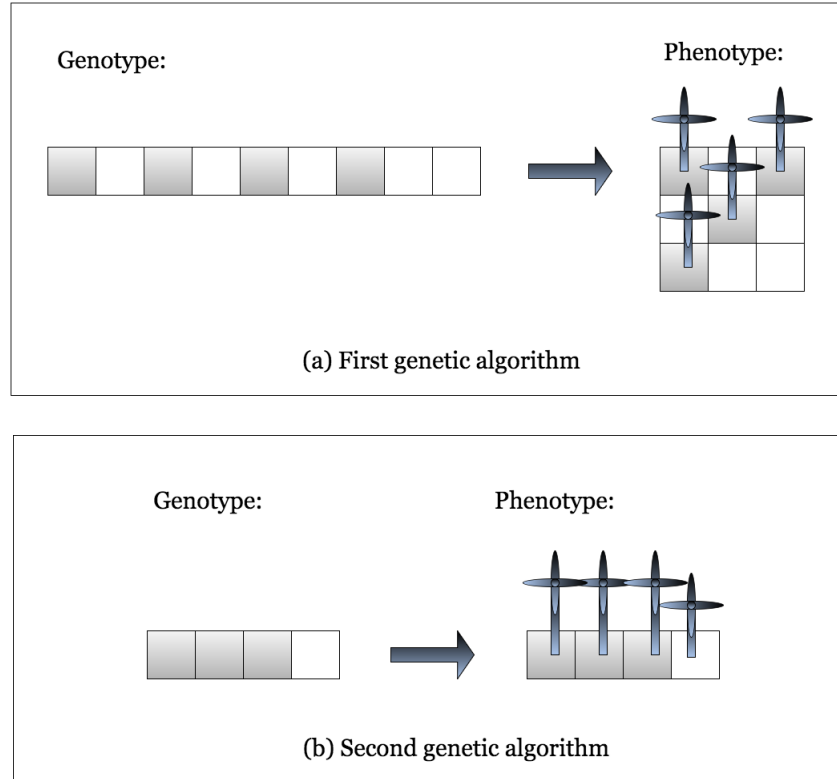
Figure 3.6: Representation for both genetic algorithms from [Chen et al., 2013]. (a) Binary string representing turbine positions, (b) binary string representing height of the given turbines positioned by the first genetic algorithm.

3.2 Wind Farm Layout Optimization Different Approaches

A greedy heuristic approach to the wind farm layout optimization problem was presented by Ozturk and Norman [2004]. The algorithm starts out with an initial solution, where a number of wind turbines are positioned in the wind farm. Next, the greedy algorithm tries to improve the layout by performing either an add operation, a remove operation or a move operation. The add operation works by randomly position one new turbine in the terrain a number of times at different locations, and observe the change in

the objective function value. The remove operation works by observing the change in objective function value when a turbine is removed, the process is repeated for all turbines. A move operation consist of moving each turbine 4 rotor diameters away from its current position in eight wind direction on at a time, and observe the change in the objective function value. The operation actually performed by the algorithm is the one that improves the objective function value the most. The greedy heuristic often converged to a local optimum, and the authors try to cope with this problem by performing randomly perturbations on a number of turbine positions if no improvements could be found using the add-, remove-, or move operation. Three approaches were investigated to find the initial position of the turbines; (1) randomly positioning, (2) packing the wind farm with as many turbines as possible, and (3) start with zero turbines. Preliminary testing showed that the second approach produced best results. The greedy algorithm was tested, and the results were compared to a feasible solution with the maximum number of turbines positioned, i.e. the initial layout before the algorithm was run. In 10 out of 12 case studies the algorithm improved the layout of the wind farm, with an average improvement of 4.3%.

Bilbao and Alba [2009] designed a simulated annealing algorithm to solve the wind farm layout optimization problem. The same wind parameters and representation as Grady et al. [2005] was used, and the algorithm was tested on the same three scenarios. The simulated annealing algorithm works as follows; first, an initial layout is obtained by randomly positioning a predefined number of turbines. Later, a random position that contains a turbine is chosen, and a new, randomly generated location is suggested. If the new position is better, the turbine is moved, but if the new position is not better, the turbine is moved with a certain probability which is regulated by a decreasing temperature parameter, in order to prevent the algorithm of converging to a local optimum. In case (a) from [Grady et al., 2005] the simulated annealing algorithm is able to find the same optimal solution, and that by using only $\approx 4\%$ of the execution time of Grady et al., and only $\approx 1\%$ of the time spent evaluating the solution. In case (b) and (c) the simulated annealing algorithm is able to find solutions with better fitness, higher power production, higher efficiency, and significantly lower execution- and evaluation times, showing that simulation annealing might be a good technique to search for the optimal wind farm layout, and it should definitively be tested in a more realistic environment.

Eroğlu and Seçkiner [2012] proposed an ant colony algorithm for the wind farm layout optimization problem; an algorithm that is inspired by how ants search for food, and show other ants food sources based on leaving a pheromone trail. The algorithm operates on a predetermined number of turbines, randomly positioned. The pheromone quantity of each turbine is decided by calculating the wake loss for the given turbine, resulting in a stronger pheromone trail for turbines with worse locations. Ants will follow the pheromone trail, therefore more ants will try to better the position of the worse turbines by moving them in random directions - the turbine is only moved if the new position is better than the current. Results are compared against [Kusiak and Song, 2010] and it is shown that the ant colony algorithm was able to position two more turbines; eight turbines in total, and that when the number of turbines is greater than two, the current algorithm produce more power, has less wake loss and higher efficiency.

Another technique to search for the best wind farm layout is swarm optimization, and Wan et al. [2012] demonstrates how a Gaussian particle swarm algorithm can solve the wind farm layout optimization problem. Swarm optimization, is an optimization technique inspired by fish schooling, insect swarms and bird flocking. The algorithm presented used an objective function that tries to maximize produced power, while minimizing constraint violations. The algorithm works as follows: First, N particles are placed in random (x, y)-positions. Second, the initial solution is evaluated and the results are saved. Third, the population best position z^g is saved, along with the current best position observed for each particle Z^p, which in the beginning will be the initial positions. An algorithm is presented, to decide which, out of two layouts, is the best. It works by first prioritizing layouts which violates less constraints, and second compare power produced by the two layouts. Forth, an updating scheme is run for a given number of iterations. It first checks if the local best position observed by that particle z^p is equal to the global best position z^g, and if so, uses an regeneration scheme that moves the particle to a random position. Otherwise, a new position is calculated for the particle based on the particles current position, its current best observed position and the global best observed position and two normalized random Gaussian numbers. If the new position is better than the previous one, the particle is moved. A differential evolution local scheme is also incorporated in each iteration to improve the algorithms local search ability. It basically

works by randomly picking three random particles as potential parents for a given particle, and combine these to generate an alternative new position, which is assign to the particle, if it is better than the current one. Results were compared against [Grady et al., 2005], and they showed that the power generated is higher using this algorithm. Their algorithm is also tested in a more realistic environment and compared against an empirical method as well as a simpler particle swarm algorithm. The results showed that the power generated was increased using the proposed algorithm.

3.3 Discussion Related Work

In order to provide an overview over the different publications that use genetic algorithms to solve the wind farm layout optimization problem the wake-, wind-, power-, and cost model, along with the objective function, type of genetic algorithm, representation of individuals, and novelties presented in each publication is presented in table **??**. Note that the table is simplified so that it fits on one page, but it still provides a good overview over the publications.

As can be seen in the table, each article utilized some variation of the Jensen model, developed by Jensen [1983], and later improved upon by Katic et al. [1986] and Frandsen et al. [2007], to model wind speed decay as a consequence of the wake effect. The same model will also be used in this thesis.

Even though not many improvements has been made to the wake model, the wind model has evolved a lot since Mosetti et al. [1994]. As presented before, the three wind scenarios developed by Mosetti et al.; (a) single wind direction, uniform intensity, (b) multiple wind directions, uniform intensity, and (c) multiple wind directions and intensities, are not very realistic (these scenarios are represented as "simple scenarios" in table **??**). The Weibull distribution, introduced by Mora et al. [2007], models the wind distribution way better, and has been adopted by everyone, except those who still wanted to compare their results against Mosetti et al. [1994] and Grady et al. [2005]. As mentioned before, the Weibull distribution will also be used to model wind distribution in this thesis, and will be described in more detail in chapter 4.

The majority of the publications presented uses the power model presented by Mosetti et al. [1994], however, Kusiak and Song [2010] present a more realistic, linear power model, which also will be used in this thesis and are explained in chapter 4.

The quality of the cost model has varied greatly in the different studies. The very unrealistic cost model that only takes turbine count into account has been adopted by many, as can be seen in table ?? represented as "simple". However, [Mora et al., 2007], [González et al., 2010], [Şişbot et al., 2010], [Saavedra-Morena et al., 2011], and [Chen et al., 2013] used more realist cost models, taking into account different parameters such as net present value, installation cost, maintenance work, civil work, interest rate and so on, these are represented as "complex" in the table. In the current thesis, a very complicated objective function is presented, one which takes into account turbine cost, substation cost, interest rate, operating costs, and turbine count as well as produced power. chapter 4 gives an detailed explanation of the objective function.

Other methods than the genetic algorithm also shows promising results in solving the wind farm layout optimization problem. Simulated annealing [Bilbao and Alba, 2009], ant-colony optimization [Eroğlu and Seçkiner, 2012] and swarm optimization [Wan et al., 2012] are popular algorithms within the artificial intelligence community, that should be optimized in order to solve the wind farm layout optimization problem. The greedy heuristic approach presented in [Ozturk and Norman, 2004] needs to be tested in a more realistic environment in order to find out if this method really could be used on turbine layout positioning.

In this thesis, the wake-, wind-, power-, and cost model, and objective function was provided by GECCO 2015, and therefore no attempt will be made in improving any of these. Since the simple genetic algorithm presented by [Mosetti et al., 1994], different approaches has been tested in order to improve the results. Already in 2005, it was shown that by using a population distributed genetic algorithm [Grady et al., 2005], and change a few parameters, the results from [Mosetti et al., 1994] was improved. In [Huang, 2007] the focus was on showing how population distributed genetic algorithms performs better than simple genetic algorithms on solving the wind farm layout optimization problem, and his results show that the population distributed

genetic algorithm is never beaten by the simple genetic algorithm. Both Grady et al., and Huang et al., uses the Island model when implementing their population distributed genetic algorithms. Even though these results indicate that population distributed genetic algorithms works better, the results of Grady et al., is beaten by the simple genetic algorithm of González et al. [2010] when a local search is used together with the genetic algorithm. Also, in [Saavedra-Morena et al., 2011] it is shown that a seeded simple genetic algorithm shows promising results, even though it is not compared against a population distributed genetic algorithm. Gao et al. [2015] also demonstrates how their population distributed genetic algorithm performs better than [González et al., 2010, Grady et al., 2005, Mosetti et al., 1994, Wan et al., 2009], but sadly they do not share many implementation details. These results clearly show that population distributed genetic algorithms can be an effective optimization technique for the wind farm layout optimization problem, and they are the motivation behind the goal statement and research questions. Together with the Island model, all the distributed algorithms presented in section 2.2.2 will be implemented and tested in this thesis.

In summary, this chapter has given an overview over them most important research within the field of wind farm layout optimization and genetic algorithms. It has been shown that not much effort have been put in deciding which genetic algorithm model is best suited for wind farm layout optimization. Additionally, it has been shown that other optimization methods can also be used to optimized wind turbine layout. The next chapter contains the implementation details of the genetic algorithm models implemented for this thesis.

Table 3.3: A simplified overview over the publications from section 3.1. In the column "Wind Model" the term "Simple scenarios" refer to the three wind scenarios developed by Mosetti et al. [1994]. The cost model and objective function columns are defined as "Simple" if it only takes into account the number of turbines. In the "GA" column, "SGA" stands for simple genetic algorithm, and "PDGA" stands for population distributed genetic algorithm.

Publication	Wake Model	Wind Model	Power Model	Cost Model	Objective	GA	Representation	Novelties
Mosetti et al. [1994]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Simple	Simple	SGA	Binary	Novel.
Grady et al. [2005]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Simple	Simple	PDGA	Binary	Population distributed.
Huang [2007]	Katic et al. [1986]	Simple scenarios	Betz [Albring, 1967]	Simple	Simple	PDGA	Binary	Compared DGA and SGA. Realistic objective.
Mora et al. [2007]	NA	Weibull distribution	NA	Complex	Complex	SGA	Integer	Weibull distribution, integer encoding.
Emami and Noghreh [2010]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Simple	Simple	SGA	Binary	Matrix representation of individuals.
Wan et al. [2009]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Simple	Simple	PDGA	Binary	New power function.
Kusiak and Song [2010]	Jensen [1983]	Weibull distribution	Linear	NA	Complex	SGA	Integer	Realistic environment.
González et al. [2010]	Frandsen et al. [2007]	Weibull	Betz [Albring, 1967]	Complex	Complex	SGA	Integer	Extensive cost model.
Şişbot et al. [2010]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Complex	Complex	PDGA	Binary	Multi-objective with pareto ranking.
Saavedra-Morena et al. [2011]	Frandsen et al. [2007]	Weibull distribution	Betz [Albring, 1967]	Complex	Complex	SGA	Integer	Seeded genetic algorithm.
Chen et al. [2013]	Jensen [1983]	Simple scenarios	Betz [Albring, 1967]	Complex	Simple	SGA	Binary	Nested genetic algorithms.
Gao et al. [2015]	Jensen [1983]	Real wind data	Betz [Albring, 1967]	Simple	Simple	PDGA	Integer	Hong Kong case study.

Chapter 4

Methodology

In this chapter, the simulator used to investigate the research questions is described. An overview of the system is presented in section 4.1. Section 4.2 includes implementation details, and design decisions made when implementing the genetic algorithm which is the foundation for all the population distributed genetic algorithms. Sections **??** contain implementation details of the population distributed genetic algorithms. The wind scenarios used to evaluate the different population distributed genetic algorithm are described in section 4.3, and the choice of implementing the genetic algorithm from scratch is defended in section 4.5.

4.1 System Architecture

The program is implemented in Java and the interactions between the different classes of the program are shown in figure 4.1. The GeneticAlgorithm class is extended by the three population distributed genetic algorithm classes: IslandModel, CellularModel and PoolModel. In addition, the GeneticAlgorithm class is also implemented as instances in all three population distributed algorithms. The main loop of the program is contained in the GeneticAlgorithm class. It uses instances of the classes WindScenario, WindFarmLayoutEvaluator, Population, AdultSelection, ParentSelection, Crossover and Mutation. AdultSelection, ParentSeleciton and Crossover are interfaces that needs to be implemented if new methods are to be added to the program. Mutation is a class containing four different mutation methods.
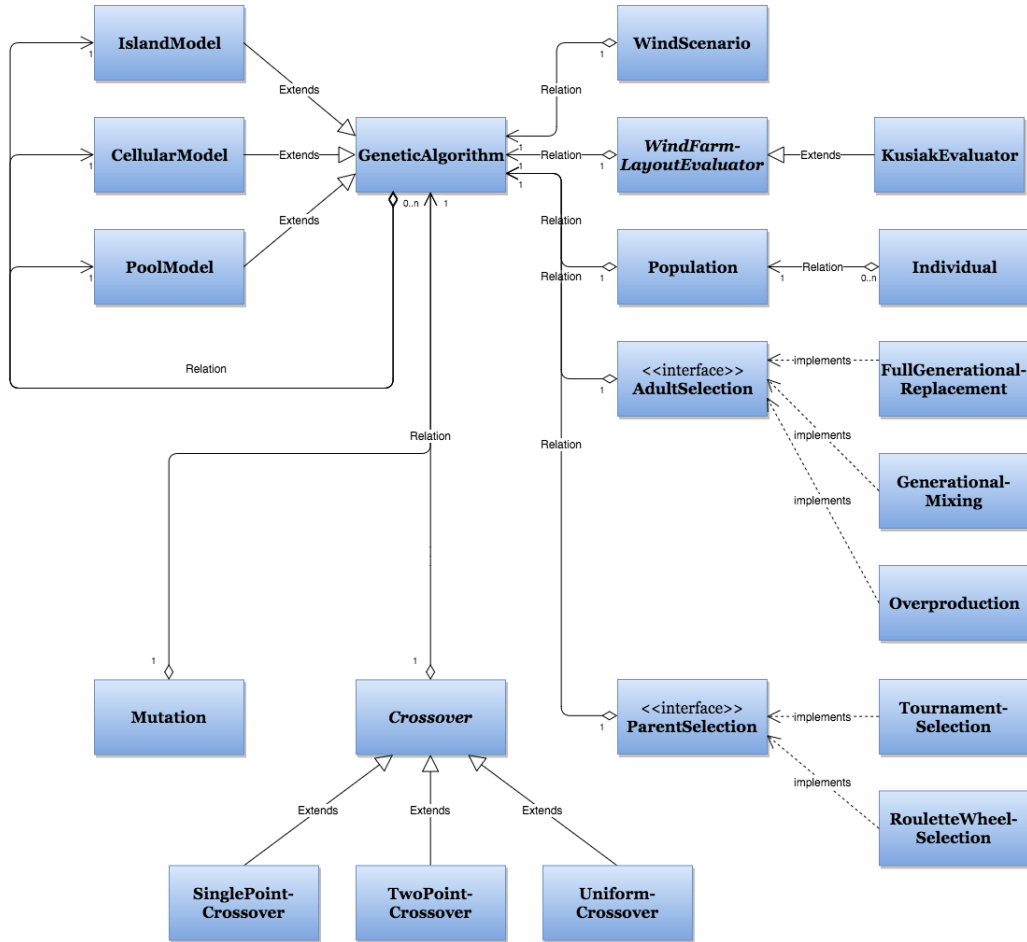
Figure 4.1: Class Diagram.

4.2 Genetic Algorithm

As mentioned in chapter 2, the genetic algorithm is a four step process: Adult selection, parent selection, recombination such as crossover and mutation, and fitness evaluation as shown in figure 4.2. Implementation details of each step is described in the subsequent sub sections. In addition, the wind-, wake-, and power model used in evaluating the genetic algorithm are described in the following subsections since these are crucial to understanding the fitness function.
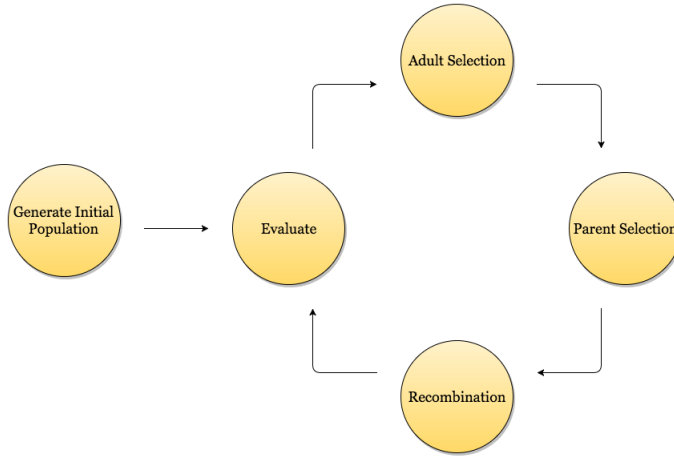
Figure 4.2: Genetic algorithm.

4.2.1 Representation

As in most of the studies presented in chapter 3, the individuals implemented for the genetic algorithm for this thesis are represented by binary strings. However, each position in the binary string can be directly mapped to a position in the terrain kept in an array called "grid". The purpose behind this design decision is that not all positions in the terrain, when dividing the terrain into a squared grid, are legal turbine positions because of the existence of obstacles. By implementing individuals this way, the genetic operations can be performed on individuals without having to check for illegal positions since this is already taken care of as soon as the scenario is read from file. Also, this design decision makes sure that no space is wasted keeping illegal positions in the binary representation. Figure 4.3 shows how a given terrain is represented using a binary string and the grid-array. The grid to the left in the figure represents the terrain. Red cells represent illegal positions while grey cells represents legal positions. An individual and the grid array are presented to the right. Since there are only 9 legal positions in the terrain the individual has length 9. The grid-array contains the (x,y)-positions of each legal positions in the terrain. The given individual has the value 1 in positions 2, 4 and 8 respectively (zero indexed), meaning that wind turbines are positioned in the (x,y)-positions in positions 2, 4 and 8 in the grid-array, meaning position (1,2), (2,0) and (3,2) in the terrain. Wind parameters and

obstacle positions are read into the program from the scenarios provided by GECCO 2016, these will be described later in this chapter.
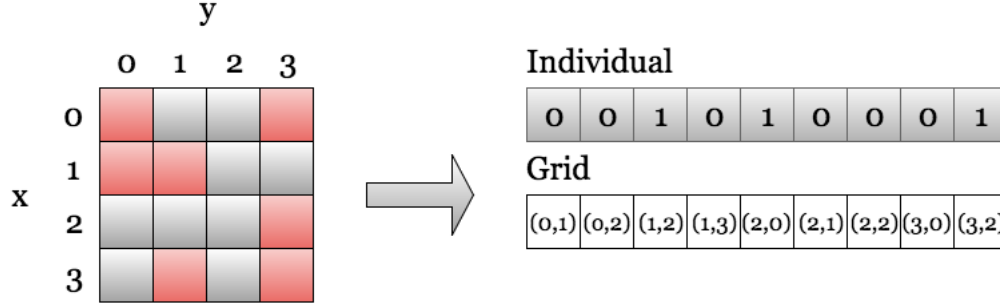

Figure 4.3: Individual representation. The grey squares represent legal turbine positions in the terrain, while the red represent illegal positions. Since there are only nine legal positions in this example, an individual is represented as a binary string of length 9. The grid array is shared between all the individuals and holds the (x, y)-coordinates for each legal position.

4.2.2 Adult Selection

Adult selection is the process of selecting which individuals that are allowed to step into the adult pool and thereby become potential parents for the next generation of individuals. Three adult selection mechanisms were implemented in this thesis: Full generational replacement, generational mixing, and overproduction. Each method was tested in order to decide which adult selection method was more suitable for solving the wind farm layout optimization problem.

Full generational replacement, the simplest adult selection mechanism, replaces the previous adult population with the newly generated child population. The second method, generational mixing, is illustrated in figure 4.4. As the name suggests, generational mixing mix the previous adult pool together with the new child pool and selects the best individuals from the mix to become the new adult population. As can be seen in the figure, the new adult pool consists of the best individuals from both the newly generated child pool and the previous adult pool. The two individuals with fitness 2

49

and 3 are selected from the child pool, and the two individuals with fitness 4 and 4 is selected from the previous adult pool. The new child population will therefore contain individuals with fitness 4, 2, 3 and 4, instead of fitness of 5, 6, 3 and 2, which would be the adult population if full generational replacement was used.
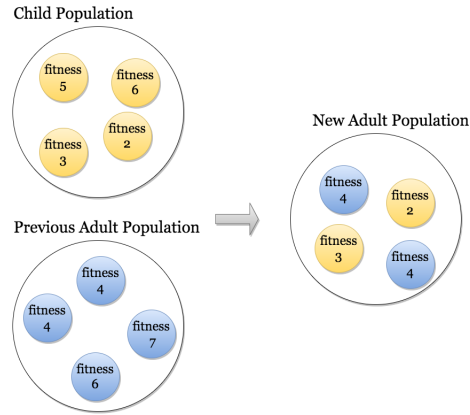

Figure 4.4: Generational mixing. The best individuals, from the pool consisting of individuals from the previous adult population and the new child population are selected as new the adult population.

Overproduction, the third adult selection mechanism, is illustrated in figure 4.5. The newly generated child population consist of twice as many individuals as there are room for in the adult population. Therefore, the children have to compete against each other for the spots in the adult pool.

4.2.3 Parent Selection

Parent selection is the process of deciding which adults become parents for the next child generation. When choosing parent selection method there are a few concerns that needs to be addressed. First, it is important that parents with good genes, i.e. lower fitness, gets their genes transferred to the next generation. However, it is also important to keep diversity in the population so that one does not end up with a sub-optimal solution; a local maximum. Two parent selection methods are implemented for the genetic algorithm for
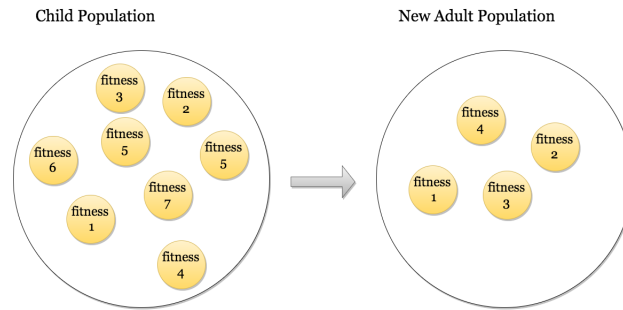
Figure 4.5: Overproduction. The newly generated child population consist of twice as many individuals as there are room for in the adult population. Therefore, only the fittest individuals from the large child population grow up into adults.

this thesis: Tournament selection and roulette wheel selection.

In tournament selection, a given number of individuals are drawn randomly from the population. The number of individuals drawn is decided by the variable *tournament size.* These individuals compete in a tournament for one spot in the parent pool. The individual with best, i.e. lowest fitness, is selected as a parent. These tournaments continue until the adult pool is full. Figure 4.6 shows how tournament selection works. As can be seen in the figure, three individuals are drawn randomly from the adult pool, meaning that the tournament size in this example is 3. The best individual, the individual with fitness 4 is the tournament winner and is allowed to enter the adult pool. In order to maintain diversity in the population there is a small probability that parents are selected randomly from the adult pool instead of with tournament selection. This probability is called *epsilon* and makes sure that a small percent of the parents that might not currently be best are not killed of right away. Different values of the *tournament size* variable needs to be tested in order to find the settings that allow the algorithm to explore different solutions, but that also prioritize the best solutions. In chapter 5, results obtained for different values of *tournament size* and *epsilon* are compared.

Roulette wheel selection assigns a probability of being chosen as parent to each individual proportional to its fitness. Individuals with better fitness will
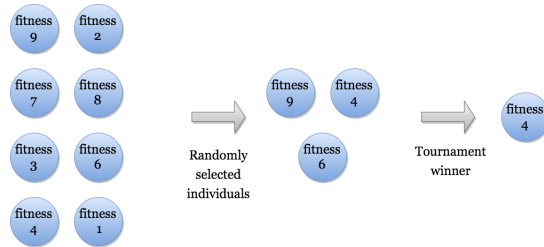
Figure 4.6: Tournament selection. Three individuals with fitness 9, 4 and 6 respectively, are selected randomly from the adult population. The individuals with best fitness, fitness 4, is the tournament winner and is selected into the parent pool.

therefore have a higher probability of beings selected into the parent pool. Figure 4.7 shows how roulette wheel selection works. The roulette wheel on the left shows the probability for each of the four individuals being selected. Since individual$_4$ has the best fitness, it has a larger probability of being selected than the others.
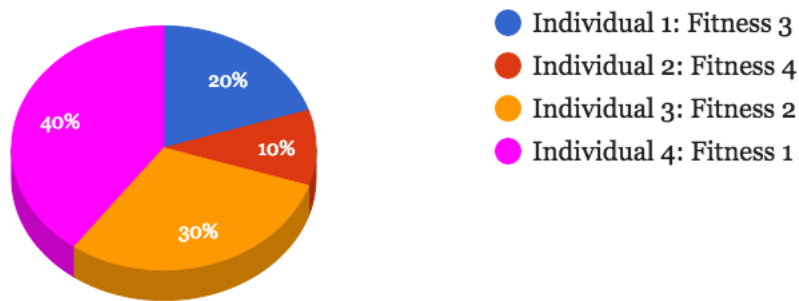
Figure 4.7: Roulette wheel selection. The roulette wheel is shown to the left, the four individuals to the right. Individual$_4$ has a four times better fitness than individual$_2$ and therefore has a four times larger probability of being selected.

4.2.4 Genetic Operations

This subsection gives an overview over the genetic operations used to produce the next child generation. Three crossover methods, elitism and four mutation methods are implemented and will be presented.

4.2.4.1 Crossover and Elitism

Crossover is the recombination method utilized by the genetic algorithm to perform sexual reproduction. A crossover operation produced two children by recombining genes of two parent individuals. The genetic algorithm implemented for this thesis has three crossover methods to chose from: Single point crossover, two point crossover and uniform crossover [Magalhaes-Mendes, 2013]. These are all presented in figure 4.8. As shown in figure 4.8a, single point crossover uses randomly generated position, called the crossover point, to perform recombination. All genes from the first parent prior to the crossover point are copied to the first child, and all genes after the crossover point are copied to the second child, and all genes from the second parent prior to the crossover point is copied to the second child, and all genes after the crossover point is copied to the first child. Two point crossover is similar to single point crossover, except that it uses to crossover points instead of one. This is shown in figure 4.8b. In uniform crossover, there is a fifty percent probability that each gene will be drawn from each parent as shown in figure 4.8c. This crossover method mix up the parent genes more that the two others, therefore more gene patterns will be lost using this method. Even though crossover is the main recombination method it is not used for creating every child individual. Some child individuals are copies of their parents with potential small mutation in their genes. The number of individuals created using crossover is decided by the crossover rate. Crossover rates are usually kept pretty high since the goal is to evolve new and better solutions, however test runs presented in chapter 5 do confirm that it is good to let a percentage of the children be produced without crossover.

A method used to make sure that the best individuals in the population does not get killed of by accident is called *elitism*. Elitism simply takes the best individual(s) from the parent population and copies it(them) directly into the new child population without performing any genetic operations on it(them). Elitism is commonly used with genetic algorithm and will be used
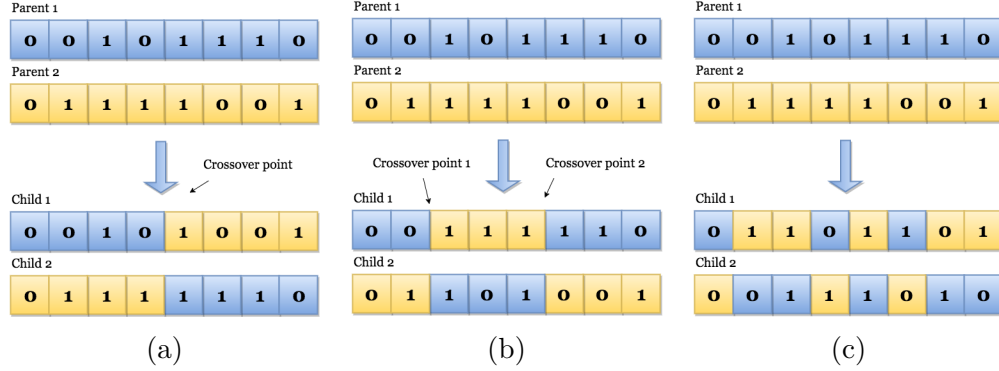
Figure 4.8: Crossover methods: (a) Single point crossover, (b) two point crossover, and (c) uniform crossover.

in this thesis.

4.2.4.2 Mutation

Although crossover is a powerful genetic operator that makes sure that child individuals to inherit genes from the best parent individuals, it is not enough by itself. Without any way to mutate genes, one can end up with a population where every individual has the same gene value in the same position, this means that no recombination method will be able to change the value of that gene. Mutation is the operation used by the genetic algorithm to make sure that the population do not get sterile. In nature, genes can be mutated in numerous ways. In this thesis, three mutation methods are implemented to mimic the mutations methods that can happen in genes [Soni and Kumar, 2014], these are called: Flip mutation, interchanging mutation, and inversion mutation. Flip mutation, shown in figure 4.9a, is the most common mutation method used in genetic algorithms. As it's name implies, flip mutation works by flipping the value of bits in the genotype. Usually, mutation rates are low since one do not want mutation to make to many changes to the good genes inherited from parent individuals, but simply introduce some diversity. Therefore, only a couple of bits are mutated in each genotype each generation. Interchange mutation and inversion mutation are rarely used in genetic algorithms, however they are included in this thesis to introduce even more diversity in the population and hopefully help the genetic search finding even better solutions. They both introduce more change than flip mutation, and

will be assigned rates even lower than the mutation rate for flip mutation. Figure 4.9b and 4.9c shows interchange- and inversion mutation respectively. Interchange mutation works by by picking two random genes and interchange their values, while inversion mutation works by picking two random positions and invert each gene between those positions.
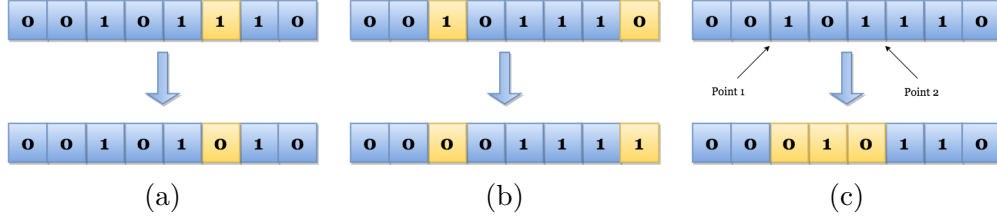

(a)   (b)   (c)

Figure 4.9: Mutation methods: (a) Single point crossover, (b) two point crossover, and (c) uniform crossover.

4.2.5   Wind-, Wake- and Power Model

The evaluation class uses the same wake-, wind- and power model as Kusiak and Song [2010]. The wake model used is the classical Jensen model [Jensen, 1983], which is used in almost every study of the wind farm layout optimization problem, as can be seen in table **??**.

Wind distribution is modeled using the Weibull distribution, a continuous probability distribution shown to model wind distribution quite well [Justus et al., 1978]. The probability density function is shown in equation 4.1

$$f(x; c, k) = \begin{cases} \frac{k}{c} \left(\frac{x}{c}\right)^{k-1} e^{-\left(\frac{x}{c}\right)^k} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{4.1}$$

where k is called the shape parameter and c is the scale parameter, and $k, c > 0$. In most of the wind scenarios provided by GECCO 2016, $k \approx 2$, this is shown empirically to be a good value for wind speed distribution [Justus et al., 1978]. On the other hand, the shape parameter vary for each wind direction. Figure 4.10 shows the Weibull distribution plotted for $k = 2$ and for different values of c.
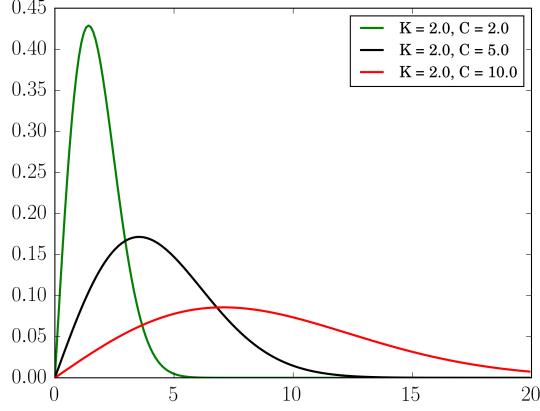
Figure 4.10: The Weibull distribution plotted for $k = 2$ for different values of the scale parameter c.

The wind scenarios used in this thesis are therefore a specification of the shape- and scale parameters for every wind direction, where wind direction is partitioned into 24 different directions. Twenty wind scenarios are provided by GECCO 2016, 10 which simply specify wind distribution parameters, and 10 that specify wind distribution parameters and locations of obstacles.

The power curve used is also the same as used in Kusiak and Song [2010], it is the linear function shown in equation 4.2,

$$
f(v) = \begin{cases} 0 & \text{if } v < v_{cut-in} \\ \lambda v + \eta & \text{if } v_{cut-in} \leq v \leq v_{rated} \\ P_{rated} & \text{if } v_{cut-out} > v > v_{rated}. \end{cases} \tag{4.2}
$$

Here λ is the slope parameter, v the wind speed, η the intercept parameter, P_{rated} is the fixed power output, and v_{cut-in} is the cut-in speed; the minimum speed for which the turbine produces power, and $v_{cut-out}$ is the cut-out speed; the maximum wind speed for which the turbine is kept on.

4.2.6 Fitness Function

The main task of the evaluation classes is to calculate the fitness of each individual based on the fitness function. The fitness function to be optimized is provided by GECCO 2016, and is displayed in equation 4.3.

$$fitness = \frac{\left(c_t \cdot n + c_s \cdot \left\lfloor \frac{n}{m} \right\rfloor\right)\left(\frac{2}{3} + \frac{1}{3} \cdot e^{-0.00174n^2}\right) + c_{OM} \cdot n}{\left(\frac{1-(1+r)^{-y}}{r}\right)} \cdot \frac{1}{8760 \cdot P} + \frac{0.1}{n} \quad (4.3)$$

Description and numerical values of all parameters given in equation 4.3 are displayed in table 4.1. As can be seen in this table, the values of n, the number of turbines, and P, farm energy output, are not given. This is because the number of turbines, together with the turbine positions, are the parameters to be optimized by the genetic algorithm. Farm energy output is the indirect parameter that we are trying to optimize. It is dependent on turbines count, position, wind scenario and so on, and is off course therefore not provided in table 4.1 either.

Table 4.1: Description and value of each parameter used in the objective function provided by GECCO 2015.

Parameter	Description	Value
c_t	Turbine cost (usd)	750 000
c_s	Substation cost (usd)	8 000 000
m	Turbines per substation	30
r	Interest rate	0.03
y	Farm lifetime (years)	20
c_{OM}	Yearly operating costs per turbine	20 000
n	Number of turbines	
m	Farm energy output	

Intuitively, the objective function can be divided into different parts. The first parenthesis in the nominator of the first fraction is the construction cost, while the second parenthesis is the economies of scale and the third

part of the nominator is yearly operating costs. The denominator represents the interests. The denominator of the second fraction describes yearly power output, while the number 0.1 in the nominator of the last fraction is a farm size coefficient.
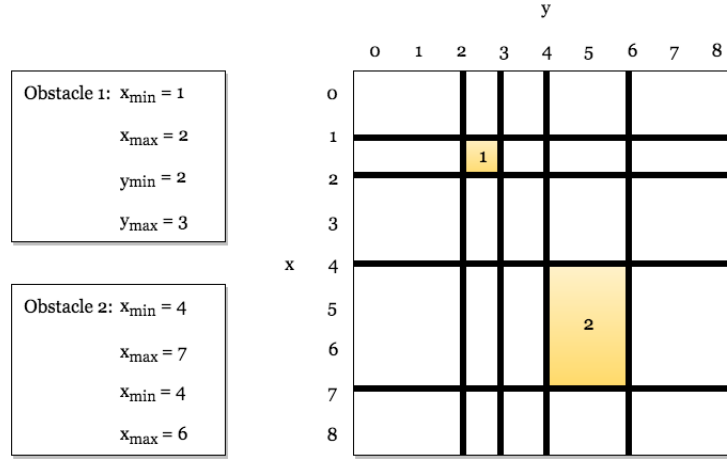
4.3   Scenarios

Figure 4.11:   Obstacles are described in the scenarios with its (x, y)-boundaries.

GECCO 2016 has provided the contestants with 20 different wind scenarios for which the genetic algorithm can be tested on. Each wind scenario contain shape and scale parameters, k and c respectively, for each wind directions where wind direction is partitioned into slots that cover an 15 degree angle each, so that there are 24 different wind directions. Unavailable areas are described in the scenario by the parameters x_{min}, x_{max}, y_{min} and y_{max} as shown in figure 4.11. In the figure, two obstacles are described giving their (x, y)-boundaries. The red lines gives the boundaries of obstacle 1 and the blue lines gives the boundaries of obstacle 2. The yellow area in the middle of the lines shows the illegal areas. Each wind scenario also contain values for the constants "width", "height", "number of turbines" and "wake free

energy". The scenarios are read from file at the beginning of each run of the genetic algorithm.

4.4 Population Distributed Genetic Algorithms: Implementation Details

Now that the reader is familiar with how individuals are represented, which selection mechanisms and genetic operators that are implemented, how fitness is calculated, and how the scenarios look, it is time to go into details of how the population distributed genetic algorithms are implemented. Although the population distributed genetic algorithms were introduced in chapter 2, the details of how each step in figure 4.2 are implemented for each model was not given. Therefore, this section contains a detailed description of how the population distributed genetic algorithms are implemented.

One decision that can be mentioned here, because it applies to all the models below, is whether the three population distributed genetic algorithms are heterogeneous or homogeneous. In an homogeneous environment, the same rules apply to the entire model. This means that the same adult- and parent selection mechanisms, the same genetic operators and the same fitness function is used for the entire model, it is not different for different Islands (Island model), threads (Pool model) or areas (Cellular model). Even thought it would be very interesting to implement heterogeneous models, it is not done in this thesis. The reason behind this decision is that it is best to let the same rules apply to every part of every model when different models are compared.

4.4.1 The Island Model

The Island model works as follows: A population of individuals are partitioned into sub populations which are distributed onto different Islands. An Island can be connected to one or more Islands and individuals on one Island can migrate to other Islands following one of the migration routes. Table 4.2 gives an description of the Island model parameters. For a number of generations, decided by the variable *migration interval*, the population on one Island follow the steps of the simple genetic algorithm from figure 4.2.

However, when the population has evolved for *migration interval* generations some individuals from one Island can migrate to another Island. These two steps continue until the total number of generations reaches some predefined value.

As mentioned in chapter 2 the Island model can be implemented with either synchronized or asynchronized migration. Synchronized migration means that migration is performed at the same generation for all Island, and asynchronized migration means that migration can be performed whenever an Island is ready. In this thesis, the Island model is implemented with synchronous migration in order to not over-complicate the migration function.

Table 4.2: Island model parameter description.

Parameter	Description
Deme size	Number of individuals on each Island (deme)
Deme count	Number of Islands (demes)
Migration rate	Number of individuals that migrate
Migration interval	Number of generations between migration.
Number of migrations	Total number of times migration is performed.
Topology	Circular

As mentioned, migration means that individuals migrates to another Island. In this thesis, migration from one Island to another means that the best individuals in Island i is copied onto Island j to replace the individuals with lowest fitness on Island j. In other words, migration does not mean that the best individuals in Island i leaves that Island, but that copies of them replace the individuals with worst fitness on Island j. The number of individuals that migrate is decided by the parameter *migration rate*.

As described in table 4.2 the topology implemented in this thesis is circular. This can be seen in figure 4.12 where the legal migration routes are indicated by arrows. Since individuals are only allowed to migration in one direction with this topology it will take time (many generations) for the individual on the upper-right Island to travel to the upper-left Island since they have to go through both Islands on the bottom. This means that different parts of the solution space can be explored on different Island before individuals from
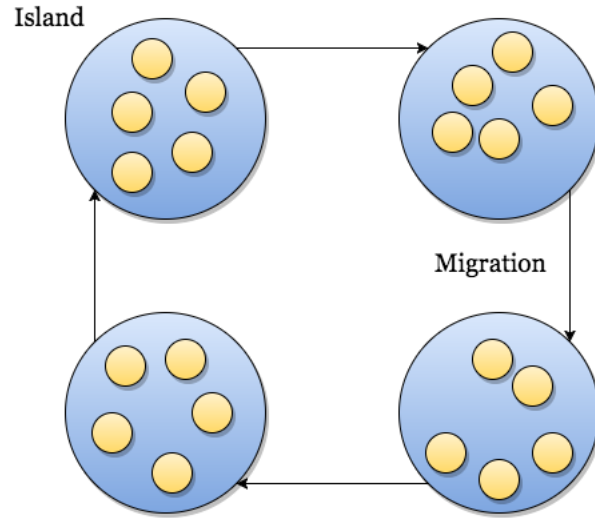
Figure 4.12: Island model topology.

one Island take over the entire population.

4.4.2   Pool Model

In the Pool model, the individuals are distributed in a resource pool. The resource pool is partitioned into equal size partitions so that one thread is responsible for one partition. Each thread selects x individuals from the entire population, where x is equal to the total population size divided by the total number of threads. Next, each thread generates a child pool by performing genetic operations on the selected individuals. After the new child pool is generated, each child individual is compared against the individual in the resource pool currently occupying its designated position. If the fitness of the new individuals is better than the fitness of the old individual the new child individual overwrites the old individuals. This is illustrated in figure 4.13. As can be seen in the figure thread i overwrites the individuals in positions 1, 3 and 4 in the partition it is responsible for.

The pool model is asynchronous by nature. The threads operate unaware of each other. Since each thread only writes to its own positions in the resource pool, it is not a problem that the threads work independently without any
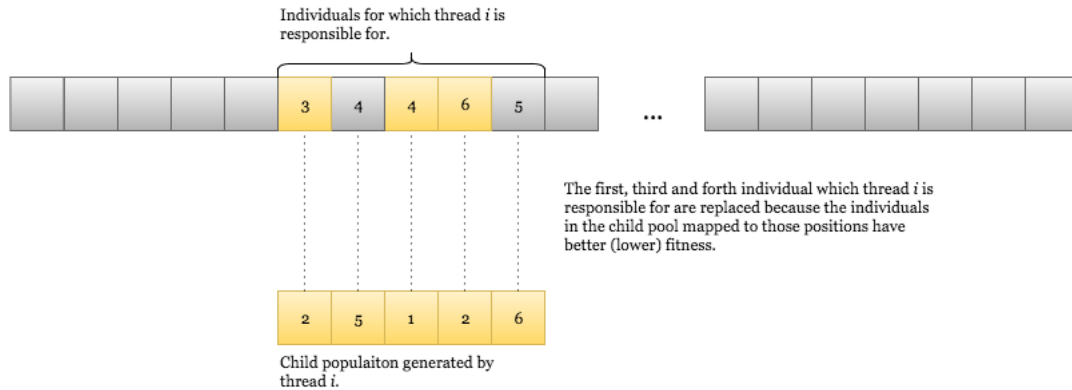
Figure 4.13: Pool model updating function. For each generation, each thread updates its region if the new individuals are better than the individuals currently occupying the position.

form of synchronization. By implementing the Pool model this way some parts of the resource pool might have better fitness that others because they have evolved for more generations than other parts. This property ensures that different parts of the search space is explored in different regions of the resource pool.

As mentioned above, parent individuals are selected from the entire resource pool. However, parents are not picked completely at random. For each parent selected a group of potential parents are selected at random from the resource pool, and these individuals compete for the spot in the parent pool by tournament selection. This guides the evolution in the right direction. Picking individuals at random might also work for this model since the new individuals are only written back if they are better than the previous individuals, however, this would make evolution extremely slow, and it would not make sense from an evolutionary point of view because the fittest individuals are not prioritized.

4.4.3 Cellular Model

The Cellular model distributes the population in a square grid. Each individual has its own cell which corresponds to an (x, y)-position. For each
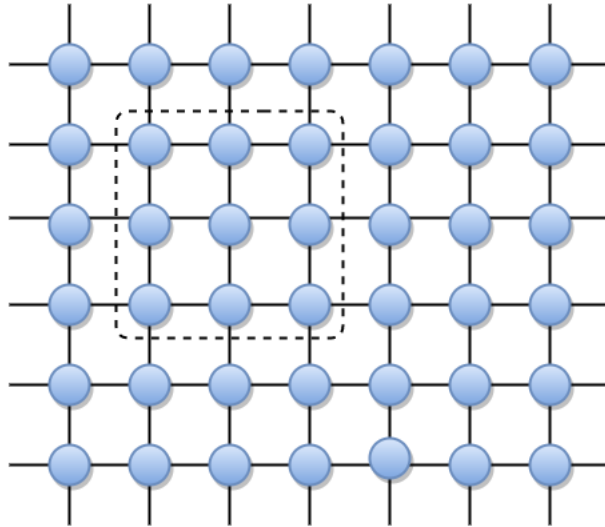
Figure 4.14: Cellular model topology. Only individuals within the marked square are allowed to become parents for the middle cell.

generation, each cell is updated. Each cell in the grid is updated by selecting parent individuals that are close to the given cell, perform genetic operations on those parents and write the newly generated individual back to the cell.

As mentioned in the background, the cells in the Cellular model could either be updated in a synchronous fashion or an asynchronous fashion. In this thesis the updating is synchronous, meaning that the cells are updated simultaneously.

The topology used in this thesis is a square topology, as shown in figure 4.14. Only individuals within the square can be selected as parents for the middle cell. Parent selection are performed by tournament selection within the square. By distributing individuals in a grid different parts of the grid will explore different regions of the search space.

4.5 Motivation behind Implementing the Genetic Algorithm from Scratch

Even though GECCO 2016 has provided each contestant with an API which could be used directly, or improved upon and used in the competition, the decision to implement the genetic algorithm from scratch was made. Even though this decision lead to more work for the author, several arguments supported the decision. First of all, even though the provided API was implemented in Java, it was not very object oriented nor was it modular, something that made it very difficult to add new features to the code without having to do major changes. In addition, the provided genetic algorithm was extremely simple with no room for making any decisions about which method to use for any of the steps in figure 4.2. Other problems such as hard-coded variables and too long methods made it very difficult to make changes and add new features.

The new genetic algorithm is implemented so that new selection methods can be added by simply implementing the AdultSelection- or ParentSelection interface, and new crossover methods can be added by extending the Crossover class. This makes the program much more flexible, if one wish to try something new. No variables are hard-coded in the new program, but taken as parameters to the GeneticAlgorithm constructor, leaving no room for making errors. The major improvement, and most important motivation behind the decisions of implementing the genetic algorithm from scratch is that it gives the author total control over every aspect of the code, and it was very important with modular code when the population distributed genetic algorithms were implemented.

In summary, this chapter contains the implementation detail of the genetic algorithm implemented for this thesis. Now that the reader is familiar with how the different genetic algorithm models are implemented it is time to take a closer look at the results obtained in this thesis.

Chapter 5

Results and Discussion

This chapter includes presentation and analysis of the results obtained in this thesis. Section 5.1 presents results from testing different parameter settings for the genetic algorithm, and briefly discusses each results. Section 5.2 presents the main results obtained when running the different population distributed genetic algorithms. Section 5.3 contains a discussion and comparison of the results obtained in section 5.2.

5.1 Parameter settings

Parameter settings are crucial for obtaining good results with the genetic algorithm. In order to find the right settings, simulations were run to find the best adult selection method, parent selection method, crossover method, crossover rate and mutation rate for the given problem. Even though it would take much shorter time and less effort to test these settings on a toy problem such as One Max, the decision was made to test them on the real problem. The reason behind this decision is that optimizing the parameters and selection mechanism on a toy problem is problematic. The wind farm layout optimization problem is extremely complex and hard to solve even with genetic algorithms because the relative positions of the wind turbines are extremely important, suggesting that the parameters and selection mechanisms that gives best results in wind farm layout optimization is might not the same that will be optimal for a simple toy problem. It is important to note that even though effort was made to find the right settings for the genetic algorithm, it is impossible to obtain the optimal ones. Just imagine

Table 5.1: Values that were kept fixed while one by one were optimized to find the correct settings for the find farm layout optimization problem.

Parameter	Value
Wind scenario	00.xml
Evaluator	KusiakLayoutEvaluator
Population size	100
Generations	100
Elitism	true
Flip mutation rate	0.01
Inversion mutation rate	0.0
Interchange mutation rate	0.0
Parent selection	Tournament selection
Tournament size	5
Epsilon	0.0
Crossover method	Uniform crossover
Crossover rate	0.9

trying the test every single value for the continuous parameter crossover rate, just that would be impossible! Therefore, the values that are tested for each parameter is based on the authors previous knowledge and experience with genetic algorithms. Should I say something about evaluation time in this section?

For each simulation, one of the parameters were tested while the others were kept fixed as shown in table 5.1. As can be seen in this table, other variables such as wind scenario, population size, number of generations, whether elitism should be used and mutation rate for interchange mutation and inversion mutation could also be tested. However, since evaluation of each farm takes half a day, even on an 8 core computer running in parallel, the time frame of this thesis prevented the author from investigation these variables.. These variables are set using the authors previous experience with genetic algorithms. Testing different parameter values on a single wind scenario might lead to values that are tailored for the given scenario and that are not as well suited for some of the others. However, there is no time to test each value for every single wind scenario, and this should not make that much of a difference. Population size is a value that influence the performance of the

66

genetic algorithm largely: Greater population size increases the probability of finding the global best solutions. The population size was set to 100 for two reasons. First of all, a population size of 100 is large enough so that many different solutions can be explored. Second, when the population size is kept at 100, the adult selection method *overproduction* will produce twice as many children, so that 200 individuals has to be evaluated for each generation something that will double the evaluation time, the step that already is the bottleneck of the algorithm. The number of generations was kept at 100 for each run. As mentioned above, evaluation is time consuming, and letting the algorithm run for more than 100 generations for each value for each parameter tested would not be possible within the time frame of this thesis. However, as can be seen on the graphs in the following sections, the fitness is starting to flat out after a 100 generations indicating that 100 generations are sufficient for the purpose of setting the parameter values. Elitism was set to true for every run without testing, meaning that the best individual of each generation will survive. This decision is based on the authors previous experience with genetic algorithm where experiments has shown that elitism leads to better results because the best individual is not lost due to coincidences. Different mutation methods were implemented for the genetic algorithm, but only the flip mutation rate was tested to find the best value. Usually, flip mutation is the only mutation method used with genetic algorithms and it is therefore also used as the main method for of mutation in this thesis. Inversion mutation and interchange mutation are implemented, but they will only happen at rare occasions to introduce more randomness to the algorithm. In the main simulations, they will be assigned extremely low probabilities so that their occurrence is so rare that they will introduce too much randomness, but hopefully introduce occasional jumps to different solutions so that the algorithm do not get completely stuck in a local optimal solution.

5.1.1   Adult Selection

Figure 5.1 shows the results of running the genetic algorithm with the three different adult selection methods. Figure 5.1a, 5.1b, and 5.1c shows the results for full generational replacement, generational mixing and overproduction respectively.

The results show that full generational replacement ends up with worse fitness

than both generational mixing and overproduction, and that overproduction is the method that is able to achieve the best fitness of the three. These results agrees with the authors previous experience. Note that with generational mixing, only the solutions in the new child population that are better than those in the previous child population are allowed to enter the adult pool. This means that for each generation the fitness of the new child population can never be worse than the fitness of the previous child population. If every new solution generated is worse than those from the previous generation, the previous generation is given another attempt to reproduce and the fitness from one generation to another is left unchanged. However, if one or more of the new solutions are better than those in the previous generation they will be allowed into the adult pool, and the new adult pool will have better fitness than the previous. Full generation replacement replace the entire population, even thought it will lead to worse average fitness than that of the previous generation. Regardless, full generational replacement is not destined to do worse than generational mixing, it could be the case that even thought the average fitness might go down from one generation to another, it might lead to better results in the end because picking the seemingly best solution as generational mixing does will not always lead to better fitness in the end. However, on this problem, the properties of generational mixing seems to be advantageous. Overproduction will, as full generational replacement, replace the entire previous adult pool with new individuals. However, with overproduction, the probability that the new generation will have better average fitness is much higher since it has twice as many children to choose from. This means that even though it is not guaranteed, it is likely that the new adult population will have higher fitness than the previous one. Still, as mentioned with generational mixing, taking greedy decisions like these might lead to sub optimal results at the end, and there is no guarantee that overproduction leads to better results than full generational replacement on every problem, but it does for the wind farm layout optimization problem. These observations will be discussed in depth in section 5.3.

5.1.2   Parent Selection

Figure 5.2 shows the results for running the genetic algorithm with parent selection methods roulette wheel is sub-figure 5.2a, and tournament selection in sub-figures 5.2b - 5.2f. Tournament selection is run with five different values for the variable *tournament size*. Clearly, roulette wheel do not work
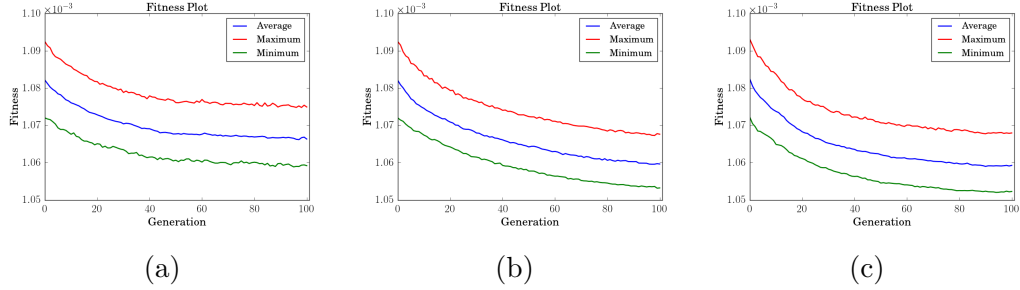
Figure 5.1: Adult selection methods: (a) Full generational replacement, (b) generational mixing and (c) overproduction. Each plot shows the averaged results obtained when running each scenario 10 times.

well for this problem. Roulette wheel assigns each individual a probability of being selected proportional to its fitness, since there is not a large different in fitness of the different individuals the selection becomes almost random. With tournament selection on the other hand, this problem is taken care of because it finds the best individuals out of those competing, event though there is not much different in their fitness.

As can be seen in the figure, the fitness gets better as the variable *tournament size* increases. However, a larger tournament size than 25 is not tested. The reason for this is that 25 is already a very large tournament size. With 25 % of the individuals competing in every tournament the population will not have much time to explore different solutions because the best individuals will take over the entire population extremely fast. As can be seen in the figure, a tournament size of 25 is not significantly better than a tournament size of 20, and therefore 20 is chosen as the final tournament size to slow down the take-over time.

As mentioned before, *epsilon* is the probability that the selected individual is selected from a random position in the adult pool instead of by tournament selection. Figure 5.3 shows the result for testing the values 5 %, 10 %, and 15 %. As can be seen in the figure, varying epsilon does not have a significant impact on the fitness on the wind farm layout optimization problem.
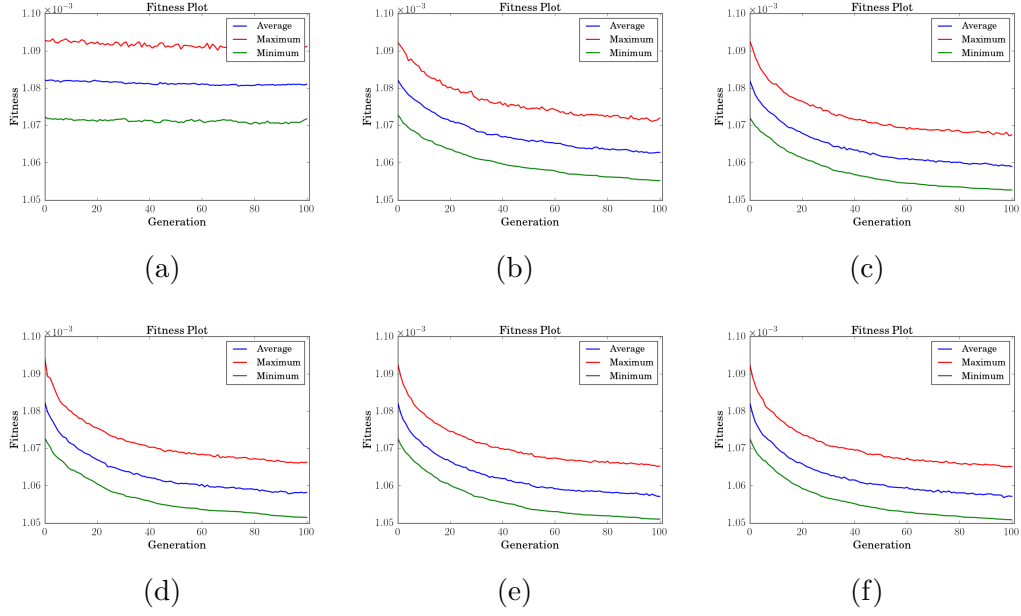
Figure 5.2: Parent selection methods: (a) Roulette wheel, (b) tournament selection, tournament size 5 (c) tournament selection, tournament size 10, (d) tournament selection, tournament size 15, (e) tournament selection, tournaments size 20, and (f) tournament selection, tournament size 25. Each result is an average over 10 runs.
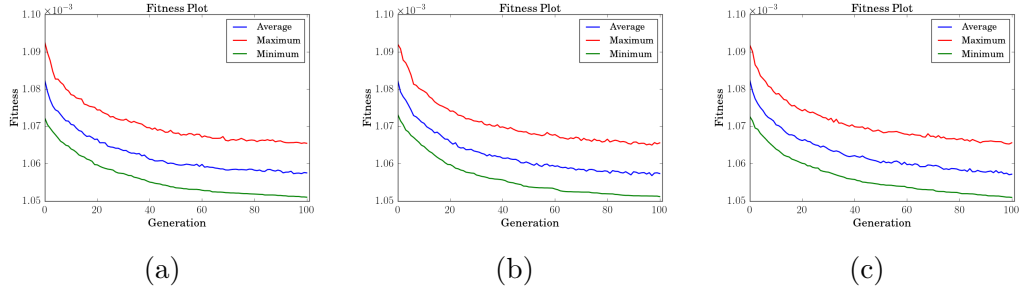

Figure 5.3: Different epsilon values: (a) Epsilon 0.05, (b) epsilon 0.10, and (c) epsilon 0.15. The tournament size was kept fixed at 20 since that was proven to give the best results. Each plot is an average over 10 runs.

5.1.3 Crossover Methods

Figure 5.4 show the results when running the genetic algorithm with single point crossover, two point crossover, and uniform crossover as can be seen in sub figures 5.4a, 5.4b, and 5.4c respectively. The experiment show that no crossover method is significantly better than the others in solving the wind farm layout optimization problem. This result is unexpected. Intuitively, one would expect uniform crossover to perform worse because it mixes up the relative positions of the wind turbines more than single point crossover and two point crossover. However, the results contradicts this hypothesis.
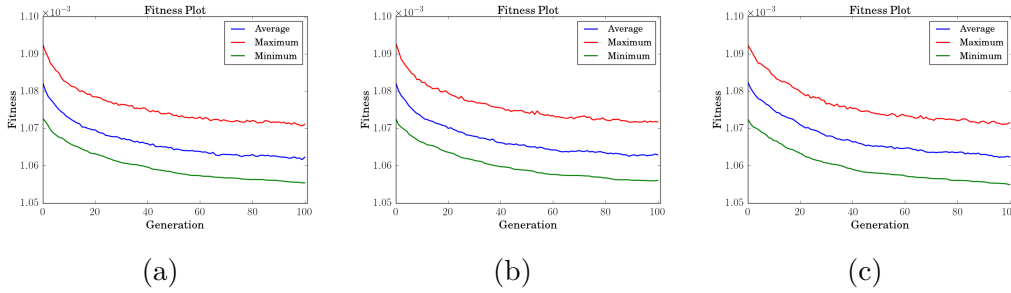
Figure 5.4: Crossover methods averaged over 10 runs: (a) Single point crossover, (b) two point crossover and (c) uniform crossover.

5.1.4 Crossover Rate

Figure 5.5 displays the results of running the genetic algorithm with different crossover rates. As can be seen in the figure, the crossover rate do not have a large impact on the fitness for this problem. A crossover rate below 0.4 gives ends up with slightly worst fitness than a crossover rate above 0.6. However, it is not obvious which crossover rate is best out of those presented in sub figures 5.5d, 5.5e, and 5.5f. The differences in the results obtained when running the algorithm with these crossover rates are not significant, and it can not be stated that one is better than the others based on these results. It can only be stated that a crossover rate above 0.6 should be used.
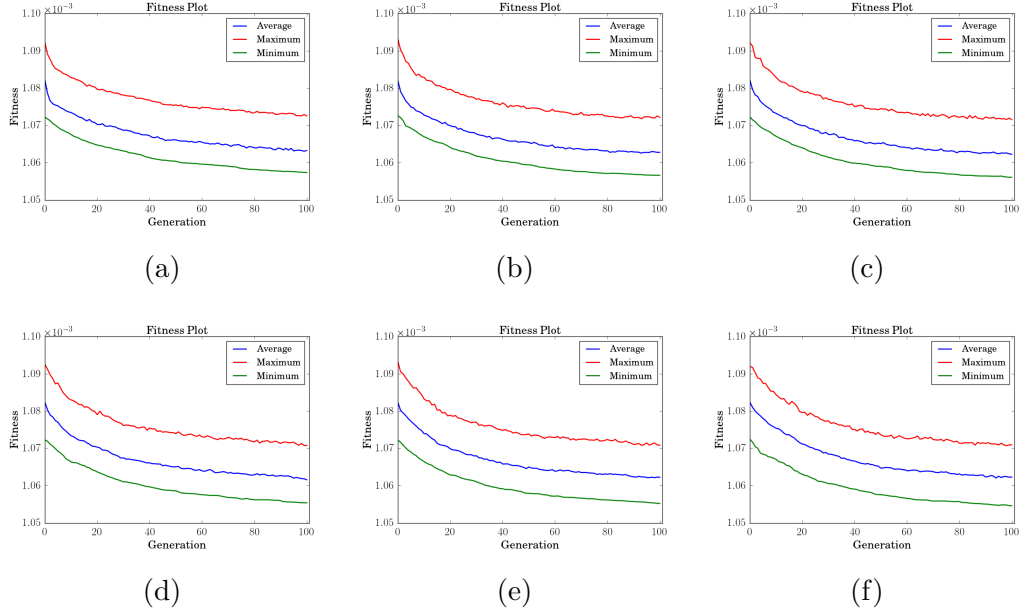
Figure 5.5: Crossover rates average over 10 runs: (a) Crossover rate 0.0, (b) crossover rate 0.2, (c) crossover rate 0.4, (d) crossover rate 0.6, (e) crossover rate 0.8, and (f) crossover rate 1.0.

5.1.5 Mutation Rate

In figure 5.6, the effect of varying the mutation rate can be seen. As the figure shows, if the mutation rate is very low as shown in sub-figure 5.6a, the population is clearly not able to find a good solution. Mutation means adding or removing a turbine. As the sub-figure show, with a mutation rate that is too low the genetic algorithm will not be able to add or remove enough turbines and therefore the population ends up in a local minima. On the other hand, when the mutation rate gets too high, as shown in sub-figure 5.6e, the population is able to explore many different solutions something that increases its chance of finding the global minima. However, since mutation is performed too frequently, it is not able to stay there even if it finds the global optimal solution. These results show that mutation rate largely impacts the fitness, and that a mutation rate of 0.001 clearly gives the best results.
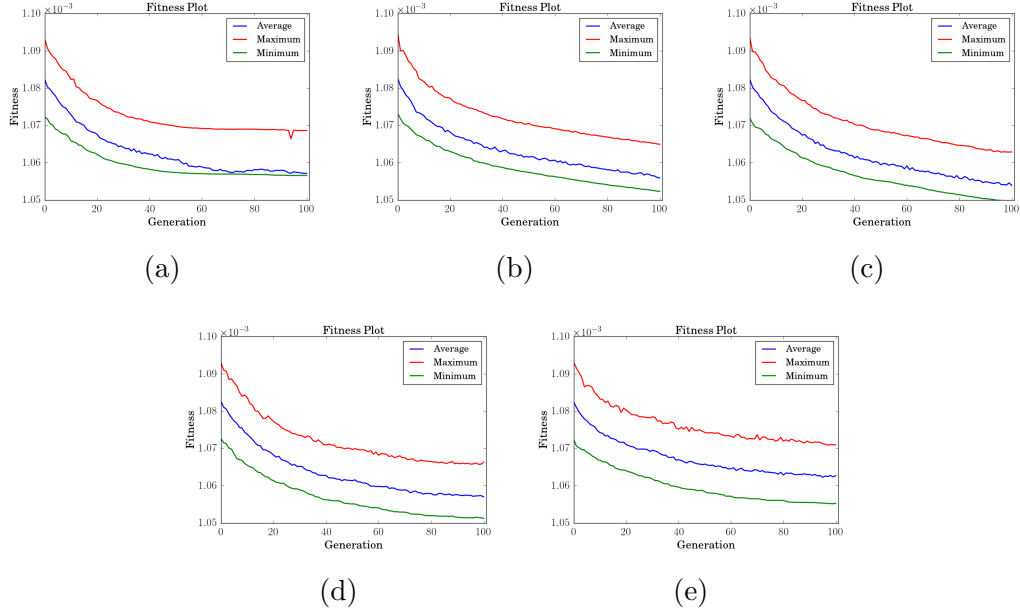
Figure 5.6: Mutation rate averaged over 10 runs: (a) Mutation rate 0.0001, (b) mutation rate 0.0005 and (c) mutation rate 0.001, (d) mutation rate 0.005 and (e) mutation rate 0.01s.

5.2 Results

5.2.1 Final Parameter Values

Table 5.2 provides an overview of the parameter values used when running the Master/Slave model. The crossover method, crossover rate, mutation rate, adult selection mechanism, parent selection mechanism and parent selection parameters are those that proved to work best for the Master/Slave model when solving the wind farm layout optimization problem as shown in section 5.1. The population size was set to 100 and the number of generations to 200. A population size of 100 is quite small, but since overproduction is the selected adult selection method a population size of 100 leads to evaluation of 200 individuals for each generation. Since evaluation is the bottle neck for wind farm layout optimization, a larger population size would not be possible to evaluate for 200 generations within the time limits of this thesis. A possibility could be to pick a larger population size and a smaller number

of generations, but, since the objective of this thesis is to explore population distributed genetic algorithms a large number of generations is crucial. Population distributed genetic algorithms need more time to find good solutions because they spend more time than simple genetic algorithms on exploring different parts of the search space.

Table 5.2: Parameter values used for the master slave model.

Parameter	Value
Population size	100
Generations	200
Crossover method	Single point crossover
Crossover rate	0.9
Elitism	True
Flip mutation rate	0.001
Inversion mutation rate	0.000001
Interchange mutation rate	0.000001
Adult selection mechanism	Overproduction
Parent selection mechanism	Tournament selection
Tournament size	20% of population size
Epsilon	0.1

Table 5.3 shows the parameters that are specific for the Island model. On each Island, the parameter values from table 5.2 are used and they are therefore not restated. The parameter *deme size*, the number of individuals on each Island, is set to 26 resulting in a total population size of 104, not 100 as in table 5.2. The reason behind this is that the implementation requires a population size of even numbers on each Island, this is not a problem since a few additional individuals will not be enough to impact the final results. The parameter *migration rate* was set to 2, meaning that only 7 % of the individuals on an Island are replaced by arriving individuals. This number was set to be low so that the new individuals would not be able to take over the new population too fast. The parameter *deme count* is set to 4 and the topology is circular as presented in figure 4.12 in chapter 4. In order to let the populations on the different Islands explore different parts of the search space 20 generations are run between each migration. Migration is performed 10 times so that the total number of generations becomes 200 as

for the Master/Slave model.

Table 5.3: Parameter values used for the Island model.

Parameter	Value
Deme size	26
Total population size	104
Deme count	4
Migration rate	2
Number of migrations	10
Migration interval	20
Topology	Circular (figure 4.12)

As can be seen in table 5.5, the Cellular model is run with population size of 225. At a first glance this might seem like an odd decision since the models above has a population size of about 100 individuals, but the reason is simple. For the Master/Slave model, and Island model overproduction is used as the adult selection technique because it was proven the best technique in section 5.1. For the Cellular model however, it does not make sense to use overproduction because of the way the grid is updated. Each new individual is generated to occupy a particular spot in the grid, because it has its own area for which its parents can be chosen from. Since each new individual is mapped to one position in the grid, overproduction does not make sense to use and it therefore not used for the Cellular model. Because of this, the population size is set to approximately 200 individuals, so that the Cellular model performs the same number of evaluations as the other models each generation, and the comparison becomes more fair than with a population size of 100. The reason why the population size is 225 and not 200 is because a quadratic grid is used to distribute the production and 225 is equal to 15^2. As explained in the methodology chapter, the topology used is a simple square containing nine individuals. The individual in the middle square can only be replaced by individuals generated by recombining individuals within the square grid. The decision to make the neighborhood this small is that it will give different solutions the opportunity to dominate different areas of the grid, giving the algorithm the time to explore.

Table 5.5 shows the parameter values used by the pool model that differs

Table 5.4: Parameter values used for the Cellular model.

Parameter	Value
Population size	225
Topology	Square (figure 4.14)

from those in table 5.2. As for the Cellular model, the population size is set to 200 so that 200 evaluations are performed for each of the 200 generations. As for the Cellular model, the individuals of the Pool model occupy specific positions. Child generation consist of producing one new individual for each position the the corresponding worker is responsible for. If overproduction was used, it would be no clear mapping between new child individual and old individual because twice as many individuals as position would be generated, this does not make sense for the Pool model. Because of this a population size of 200 individuals where used so that 200 evaluations are performed for each generation as for the other models.

Table 5.5: Parameter values used for the Pool model.

Parameter	Value
Population size	200
Number of workers	4

In summary, each algorithm run for 200 generations, and approximately 200 individuals are evaluated for each generation. This makes sure that every model gets a fair amount of processor time.

5.2.2 Performance Metrics

Each of the 4 genetic algorithm models are tested on 4 wind scenarios. The wind scenarios are provided by GECCO 2016 and can be seen in table 5.6. As can be seen, the first two scenarios are without obstacles, while the two last scenarios are with obstacles.

The performance of the 4 models on each of the four scenarios are measured using the performance metrics shown in table 5.7. The first performance

Table 5.6: Test scenarios provided by GECCO 2016.

Scenario Name	Obstacles
00.xml	No obstacles.
05.xml	No obstacles.
obs00.xml	Obstacles.
obs05.xml	Obstacles.

metric is the results of the fitness function presented in chapter 4. The efficiency is a measure of how much energy the given layout is able to produce out of the power the layout would have been able to produce had there been no wake effect, as shown in equation 5.1

$$\text{efficiency} = \frac{\text{yearly power output}}{\text{wake free yearly power output}}. \tag{5.1}$$

The performance metrics cost and power represent the cost and power part of the fitness function. Figure 5.7, shows which parts of the fitness function presented in chapter 4 that represents cost and which represents power. The last metric, number of turbines, is intuitively the total number of turbines in the layout.

$$fitness = \frac{\overbrace{\left(c_t \cdot n + c_s \cdot \lfloor \frac{n}{m} \rfloor\right)\left(\frac{2}{3} + \frac{1}{3} \cdot e^{-0.00174n^2}\right) + c_{OM} \cdot n}^{\text{Cost}}}{\left(\frac{1-(1+r)^{-y}}{r}\right)} \cdot \underbrace{\frac{1}{8760 \cdot P}}_{\text{Power}} + \frac{0.1}{n}$$

Figure 5.7: A screen shot of the fitness function presented in chapter 4 where the cost and power metrics are shown.

In chapter 2, one of the resulting layout from [Grady et al., 2005] was shown as an example layout. In this thesis, the resulting layout will be to large to include. Additionally, with the realistic wind scenarios used, the wind turbines will be scattered around in the grid in a way that would not make

Table 5.7: For each scenario, results are measured in these measurements.

Results measured	Description
Fitness	Fitness of the best individual
Efficiency	Amount of power produced out of maximum
Cost	Total cost (USD)
Power	Yearly power output (kWh)
Number of turbines	Total number of turbines

sense to look at. The performance metrics above on the other hand, gives a much more clear understanding of the performance of the models and are the metrics that will be used to present the results.

5.2.3 Results Scenario 00

The results when all the models were run on scenario 00.xml is shown in figure 5.8. The fitness plot is shown in sub-figure 5.8a. The Master/Slave model is able to get the best fitness, the Pool model comes in second, quite close to the Master/Slave model, the Island model comes in third, and the Cellular model comes in forth, clearly not able to keep up with the other models.

Sub-figure 5.8b shows the performance of the different models in terms of efficiency. Efficiency is a very interesting measure, since it shows the models' ability to live up to their potential. As will be discussed below, the Master/Slave model and Pool model end up with a different number of turbines. Interestingly, within their chosen number of turbines, both models are able to obtain almost the same efficiency, showing that they are both good at optimizing turbine positions.

As shown in sub-figures 5.8c and 5.8e, the plots for cost and number of turbines have identical shape. This makes sense off course since the cost increase and decrease with the number of turbines. The plots show that the Master/Slave model end up with a larger number of turbines than the other models, which all end up with the same number of turbines. The Master/Slave model, the Pool model and the Cellular model all decide on the number of turbines before having evolved for 20 generations, while the Island model is able to explore different solutions until around generation 50,

when it stabilized on the same solution as the Pool- and Cellular model. The Island model cost plot has a different shape than the other plots. While the other models slowly go from one solution to another, the Island model jumps from one part of the solution space to another. These observations make sense, because, these jumps occur at generation 20 and 40, the generations where the first and second migration takes place.

Between generation 20 and 40, the Island model explores a solution with the same number of turbines as the Master/Slave model, a solution that could be the global minima. However, the Island model is not able to stay in this minima, because it observes that jumping to a solution with fewer turbines seems more promising. If the migration interval had been longer than 20 generations, the Island model might have been able to optimize turbine positions in the global minima so that it would discover the true potential of the solution before leaving it for a sub-optimal solution.

The performance of the different models can be viewed in terms of power in sub-figure 5.8d. Power is also closely related to the number of turbines. Clearly a solution with more turbines leads to higher energy output. However, as the Island model power plot shows, solutions with high power production might not be prioritized because the cost becomes too high so that the total fitness becomes worse. After the models have stabilized on a particular number of turbines, the power is slowly increasing. This is because the turbines are slowly moved into positions with less wake loss.

In summary, the optimization can be viewed as consisting of two phases. The first phase is the search for the optimal number of turbines, while the second phase consist of moving turbines around in order to reduce wake loss. The Island model stays in phase one longer than the other models. Still, the Master/Slave model is the only model able to find what could be the global optimal solution in terms of the number of turbines.
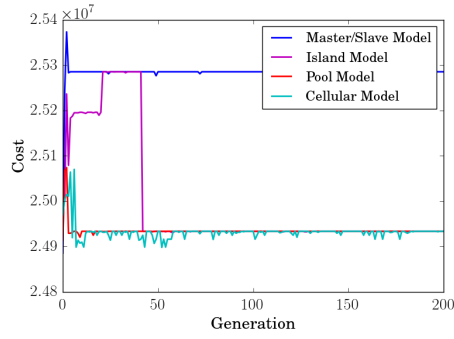
5.2.4   Results Scenario 05

The results for running the different models on scenario 05.xml is shown in figure 5.9. The fitness plot is shown in sub-figure 5.9a. As for scenario 00.xml, the Master/Slave model obtains the best fitness closely followed by the Pool model. The Island model comes in third again, and the Cellular
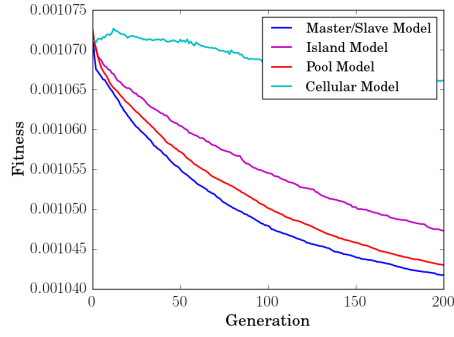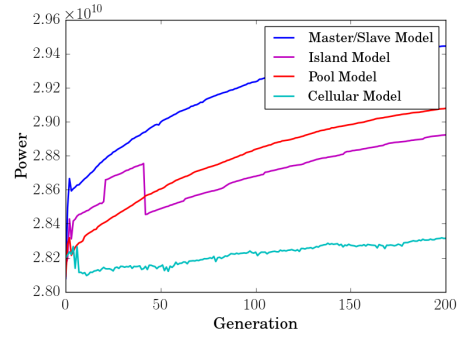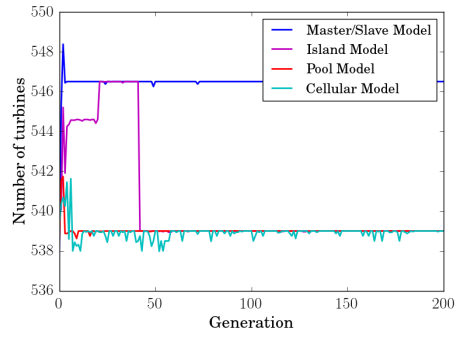
(a) Fitness
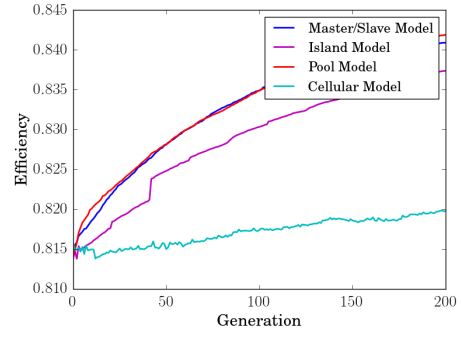
(b) Efficiency

(c) Cost

(d) Power

(e) Turbies

Figure 5.8: Scenario 00.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.

model is not able to keep up with the other models on this scenario either. After 200 generations the Cellular model has a fitness which each of the other models had beaten already at generation 10.

The efficiency plot is shown in sub-figure 5.9b. As before, the Master/Slave model and Pool model is able to find a solution with approximately the same efficiency. The Island model is a few steps behind, while the efficiency of the Cellular model is increasing extremely slowly.

Sub-figures 5.9c and 5.9e shows cost plot and number of turbines plot respectively. As can be seen, the Master/Slave model and Pool model does not explore different number of turbines, they both stabilize on a given number after less than 10 generations. The Cellular model explores for about 15-20 generations before it decide on the number of turbines. From the sub-figures it looks like the Island model explores for a little less than 50 generations, but if one takes a closer look, it actually does a small jump to a solution with fewer turbines at about generation 80. However, it is not able to stay in this solutions for longer than about 5 generations. As for scenario 00.xml, the Master/Slave model ends up with a higher number of turbines than the other 3 models. It seems like the Master/Slave model has found the global minima, or at least a better local minima than the other models, something that might explain why the Pool model is not able to get a fitness as good as the Master/Slave model.

Power is closely related to the number of turbines, and as can be seen in sub-figure 5.9d, the Master/Slave model finds the solution that is able to produce most energy. The Pool Model and Island model both end up with the same number of turbines, and as can be seen the amount of power that they are able to produce is similar. The Pool model produces a little more power because it is able to find better positions for the turbines as can be seen since it beats the Pool model in fitness and efficiency.

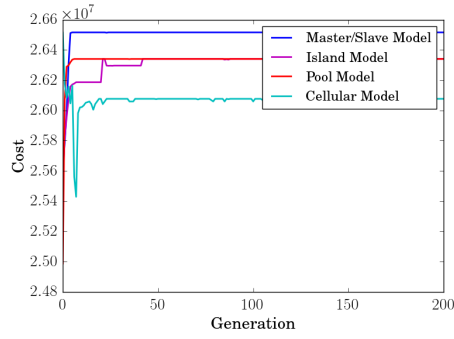### 5.2.5   Results Scenario obs00.xml

Figure 5.10 shows the results obtained when running the model on scenario obs00.xml. As for scenairo 00.xml and 05.xml, the Master/Slave model obtains the best fitness, the Pool model the second best, the Island model third
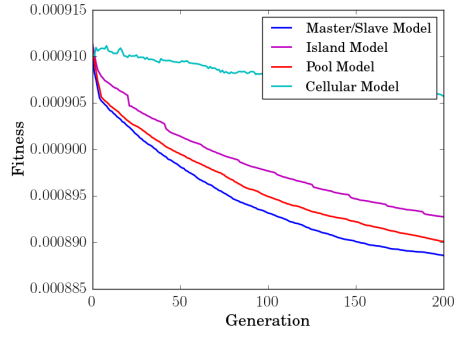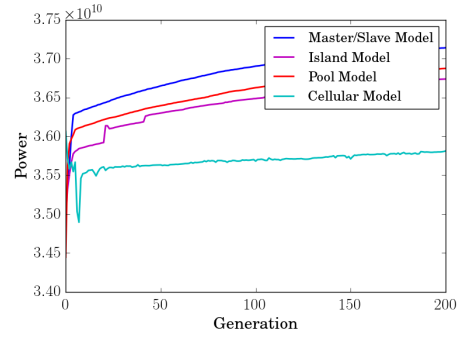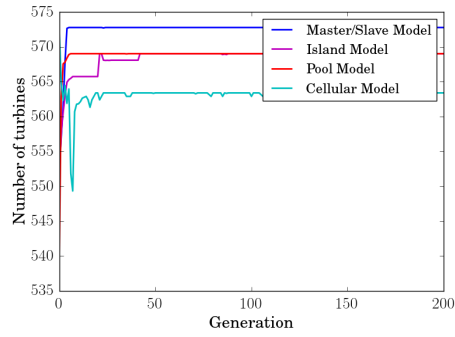
(a) Fitness
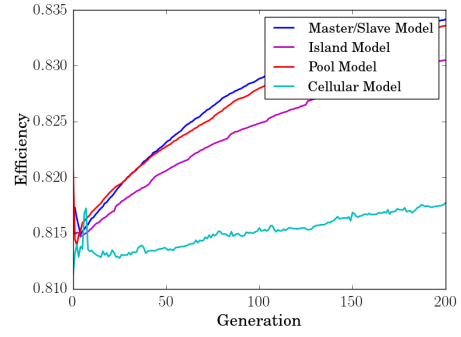
(b) Efficiency

(c) Cost

(d) Power

(e) Turbies

Figure 5.9: Scenario 05.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.

best and the Cellular model the fourth best, as can be seen in sub-figure 5.10a.

As for efficiency, the results are similar as those observed for scenario 00.xml and scenario 05.xml as can be seen in sub-figure 5.10b. Both the Pool model and Island model are able to optimize their solutions so that they produce more than 84.5 % of the wake free energy. The Island model however, is only able to produce just above 83.5 % of the wake free energy.

As shown in sub-figures 5.10c and 5.10e the Island- and Cellular model end up with the same number of turbines, while the Master/Slave- and Pool model end up with different number of turbines. The Island- and Cellular model ends up at the solution with the highest number of turbines and the Pool models finds the solution with fewest turbines. This explains why the Island model scored so badly on efficiency, it finds out that adding turbines leads to higher power production and therefore better fitness and therefore it ends up in a sub-optimal solution.

The Island model finds the solutions which produces most power, a reasonable result since no other model ends up with more turbines. However, the Island model is closely followed by the Master/Slave model even though it finds a solution with fewer turbines. This shows that the Master/Slave model is way better at optimizing the turbine positions. On problematic observation is that the Cellular model ends up with a solution that produces less power than all the other models even thought it ends up with a solution with the same number of turbines as the Island model. This results show that the Cellular model is not able to find a solution that position the turbines well within 200 generations.
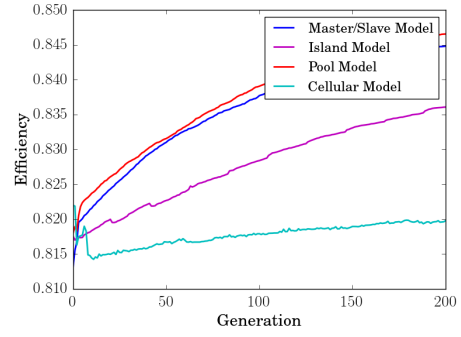
5.2.6   Results Scenario obs05.xml

Figure 5.11 shows the results from scenario obs05.xml averaged over 10 runs. Fitness is shown in sub-figure 5.9a. As can be seen, the results are similar to the results for the other 3 scenarios.

Since the result is similar to those observed before, they will not be discussed in details here, however one interesting observation has been made: In sub-figures 5.11c and 5.11e it can be observed that every model end up
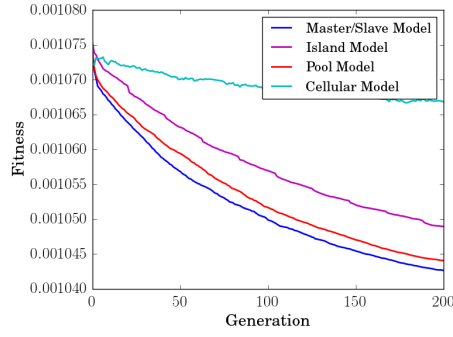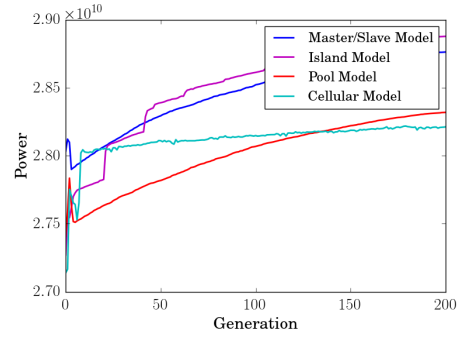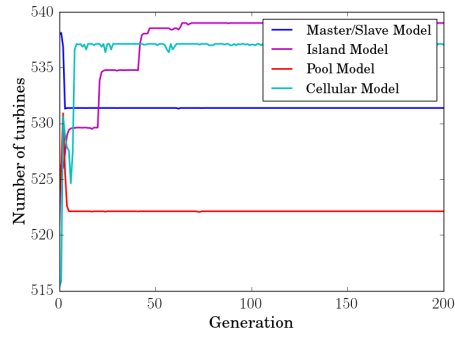
(a) Fitness

(b) Efficiency
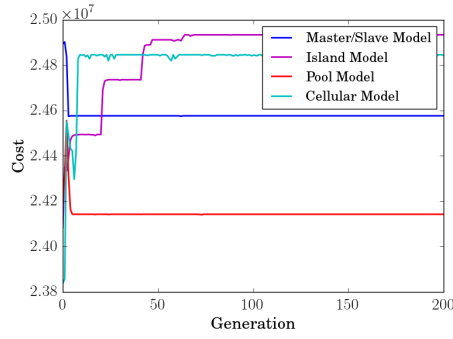
(c) Cost

(d) Power

(e) Turbies

Figure 5.10: Scenario obs00.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.

in solutions with a different number of turbines. These results shows the complexity of the problem by showing how easy it is to end up in a local minima. Because, even thought the global minima is not known it is known that at least 3 of the models are stuck in a local minima.

5.3 Discussion

In the previous section it was shown that the non-population distributed Master/Slave model consistently was able to evolve layouts with better fitness than the 3 population distributed models. The Pool model came in second, the Island model came in third and the Cellular model came in fourth and last. There can be many reasons for this, and they will all be discussed in this section. The main results are listed below.

- The non-population distributed Master/Slave model was the model able to obtain the best results.

- The Pool model obtained the second best results.

- The Island model obtained the third best results.

- The Cellular model obtained the forth best results.

As mentioned earlier, wind farm layout optimization is extremely hard, if not impossible, to solve analytically. The solution space is simply too huge for any computer to evaluate every possible wind farm layout to find the best layout. For a wind farm with only 500 legal turbine positions, 2^{100} different layouts exists! With a solution space this large, it is very unlikely that a genetic algorithm is able to find the global optimal solution. As the results show, the Master/Slave model only spend less than 10 generations searching for the optimal number of turbines before deciding on a number. This means that it is unlikely that it is able to find the optimal number of turbines, and one would expect that a model that spends more time testing different number of turbines would be able to come up with better solutions. However, the results contradicted this hypothesis. The Island model is the only model able to explore different number of turbines for a significant amount
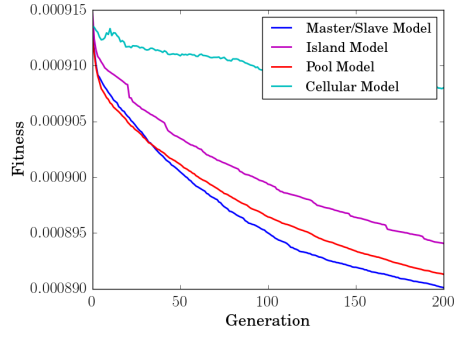
(a) Fitness
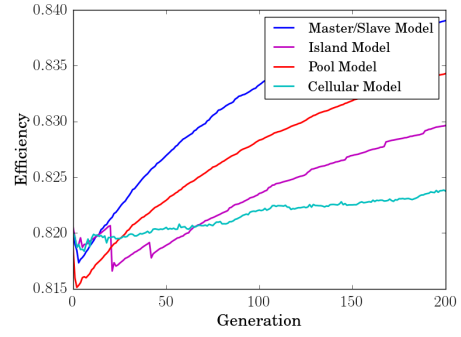
(b) Efficiency
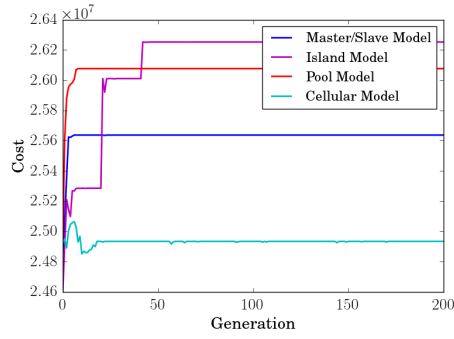
(c) Cost

(d) Power

(e) Turbies

Figure 5.11: Scenario obs05.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.

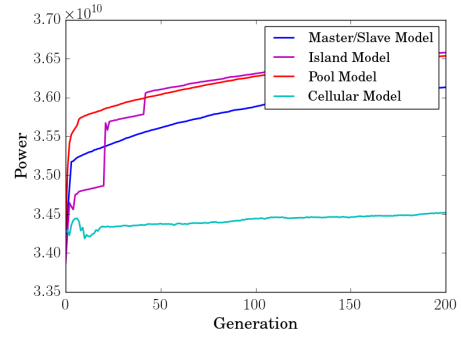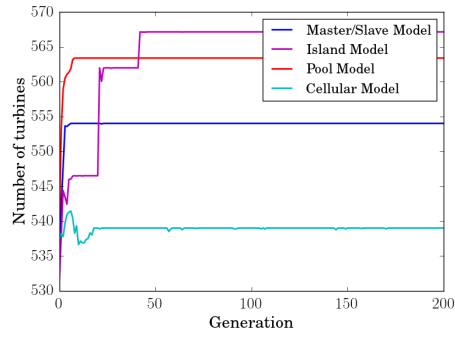of time. But, the Island model is still not able to find layouts as good as the Master/Slave model and the Pool model! These results suggest that the models benefit from settling for a sub-optimal number of turbines because then they have more time to spend optimizing the positions of the turbines. Put in another way, spending many generations testing out different number of turbines might be a waste of time, it might be better to settle on a sub-optimal number of turbines, and optimize turbine positions.

Another reason that could explain why the Master/Slave model were able to obtain better results than the population distributed genetic algorithms could simply be that the parameter values were tailored for the Master/Slave model giving it an advantage. Even though the parameters, selection mechanisms, and genetic operations are common for all the models, it is not certain that the settings that works best for the Master/Slave model works best for the other models. If these settings had been tailored for each model it could have been the case that one or more of the population distributed genetic algorithms would be able to perform better. Sadly, the 20-week time frame of this project prevented the author form tailoring these settings for each of the population distributed genetic algorithms. However, since each model is solving the same problem it is not very likely that there would be a huge variability in the settings for the different models.

An adjustment in the current thesis that the author believes could affect the results more is optimization of the parameter values that are unique for the different population distributed models. Parameters such as *deme size*, *deme count*, *migration rate*, *migration interval*, and *number of migrations* for the Island model, and *number of workers* for the Pool model were selected based on educated guessing. Different topologies for the Island model and the Cellular model should also have been tested in order to find out which where the optimal ones. However, as mentioned in the paragraph above, the time frame prevented these variables to be tested and optimized.

In spite of the fact that the results show that the traditional non-population distributed Master/Slave model works best for the wind farm layout optimization problem it is important to note that these results could be different under different circumstances. It has only been proved that it performs best with the same amount of resources, and with the settings used in this project. (Sentence about too many choices for genetic algorithms). Even though pro-

viding each of the models with the same amount of computer resources seems fair it also introduces problems. The main property that distinguished population distributed genetic algorithms from non-population distributed algorithms is that they spend more time exploring different solutions in order to hopefully find the global optimal solution. This means that population distributed genetic algorithms by nature require more time than non-population distributed genetic algorithms. In order to satisfy this requirement, but also provide a fair comparison one could allow the the models to run for the same number of generations after the population has decided on the number of turbines it wish to pursue. This way, the population distributed genetic algorithms would not be punished because the spend time exploring different number of turbines longer. As mentioned earlier, the Island model is the only one of the population distributed genetic algorithms that spend a significant amount of time exploring different number of turbines, so this model would be the only one potentially benefiting from this scheme.

In chapter 3 Grady et al. [2005] and Huang [2007] showed that under different circumstances, the Island model were able to find better wind farm layouts than the Master/Slave model. It was therefore a little surprising that the Island model were outperformed by the Master/Slave model in this project, but there can be many reasons for this. Grady et al. [2005] implement an Island model and compare the results obtained with those of Mosetti et al. [1994], which is obtained with a traditional non-population distributed genetic algorithm. However, Grady et al. [2005] increases both the population size and the number of generations in their experiment. It can therefore not be stated with certainty that the Island model were the key to their success, it could just as likely be the increase in population size or the increase in the total number of generations. Huang [2007] on the other hand, implements one traditional genetic algorithm and one Island model and compare their results when both models have the same total population size and generations. Therefore, it was more surprising that the results in this thesis contradicted the results they obtained. But, as is always the case with genetic algorithms, there can be many reasons for this. In the experiments done by Huang [2007], the population size were 3 times as large as in this project and the total number of generations were more than 12 times the total number of generation used in this thesis. This fact can be the reason for the difference. Obviously, a larger population size is an advantage for genetic algorithms. The probability of finding the global best solution increase with the population size.

This is also true for the number of generators, increasing the total number of generations increases the probability of finding the global optimal solutions. Another difference between this project and the project of Huang [2007] is the topology. In the topology of Huang [2007], 20 Islands are used, five times as many islands as in this project. This means that it would take longer time before individuals on one Island is able to spread their genes to the entire population, therefore different solutions have more time to evolve. Yet another different between this project and the project of Huang [2007] is the fitness function. In this project, the fitness function for which the models try to optimize is way more complex than the one used in Huang [2007]. Both the cost model and the wind model used in this thesis is more realistic and complex. Therefore, it could be the case that while the Island model is able to outperform the simple genetic algorithm with a simple fitness function, it comes up short when the function to be optimized is more complex, realistic and time consuming.

The Pool model stands out as the population distributed genetic algorithm fittest for solving the wind farm layout optimization problem. One reason why this might be is that out of the 3 population distributed models, the Pool model is the one most similar to the Master/Slave model. As mentioned above, the parameter values were decided when tested on the Master/Slave model, and since the Pool model is most similar to the Master/Slave model it is likely that the same parameter values would work for this model. As the Master/Slave model, the Pool model does not waste many generations searching for the optimal number of turbines. It makes the decision in less than 15 generations, and spends the rest of its time optimizing solutions with that given number of turbines. This worked well for the Master/Slave model, and it worked well for the Pool model too.

The Cellular model is outperformed by all the other models. It is not able to get results that are close to those obtained by the others. On reason for this is that the Cellular model is by far the model that need most generations to find a good solution. For example, for the individual in the upper left corner, it would take a minimum of 14 generations to spread its genes to the individual occupying the lower right corner. What is even more scary is that on average it would take the same individual how many? generations to spread its genes to the lower right individual. Because of this slow converges, it is not surprising that the Cellular model is outperformed by the other models

89

when they all run for 200 generations with 200 evaluation each generation. A solution to this problem would be to use a larger square for which parents for the middle individuals can be selected from. This would however make the model more similar to the Master/Slave model, and make it lose its unique features. As mentioned in chapter reference if mentioned for each generation every individual is replaced by a newly generated individuals. Something that explains why the fitness increases in the first 5-10 generations before it starts going down. This could be solved by implementing the Cellular model with the same replacement strategy as the Pool model; only replace individuals with new individuals with higher fitness. However, this was tested by the author on scenario 00.xml, and it actually lead to worse fitness than the current strategy. Should these results be included? Appendix?

Even though it is outside the scope of this thesis, it is interesting to discuss the fitness function. It is very hard to find the optimal fitness function. The fitness function is based on cost estimated, power estimates and wind predictions.Therefore, it might not be the case that those who came up with the fitness function is 100% certain that it is correct. Because of this, they might be interesting in taking a look at the different solutions obtained by the different models even though the best fitness is obtained by the Master/Slave model. For example, in figure 5.10, the Pool model is able to come up with a solution with fitness that is very similar than the fitness obtained by the Master/Slave model where the cost is largely reduced. Therefore, it might be interesting for the wind farm owners to take a look at different solution suggestions and compare them.

In summary, it has been showed that population distributed genetic algorithms are not always able to come up with better results than the traditional, non-population distributed genetic algorithm. But, as mentioned above, it is extremely difficult to find the optimal settings for every parameter of the genetic algorithm, and it can only be concluded that the Master/Slave model outperforms

Chapter 6

Conclusion

In this thesis, different genetic algorithm models have been implemented and tested. The goal of the thesis was to investigate how population distributed genetic algorithms would perform when used to solve the wind farm layout optimization problem. The performance of the population distributed genetic algorithm were compared to the performance of the simple genetic algorithm as well as to each other. In this chapter, an overview of the thesis will be presented in section 6.1. In section 6.2 the goal and research questions are discussed. Section 6.3 contains a discussion of how this thesis contribute to the field of wind farm layout optimization using genetic algorithm technology, and section 6.4 contains suggestions to further work.

6.1 Overview

In chapter 2, the wind farm layout optimization problem was introduced and challenges to wind farm construction were presented. A thorough review of the genetic algorithm was also given, and the 3 population distributed genetic algorithms investigated in this thesis were introduced.

Chapter 3 contains a thorough review of research within the field of wind farm layout optimization using genetic algorithm technology. In table 3.3 the most important features of each publication is presented. Chapter 3 also contain a brief overview of other artificial intelligence methods that have been used to optimize wind farm layout. During the research phase, it became ap-

parent that no attempt had been made to investigate the performance of the Pool model and the Cellular model on wind farm layout optimization. The only population distributed genetic algorithm reported tested was the Island model Grady et al. [2005], Huang [2007], Wan et al. [2009], Şişbot et al. [2010] and Gao et al. [2015]. Although [Grady et al., 2005] stated that by using the Island model they were able to come up with better solutions than when the simple genetic algorithm had been used, the models had not been compared when given the same amount of resources. These observations was the main motivation behind the choice to investigate the performance of different genetic algorithm models on the wind farm layout optimization problem.

Chapter 4 presents the methodology for investigating the goal- and research questions. The decision was made to implement the genetic algorithm from scratch and connect it to the wind farm simulator provided by GECCO 2016. The chapter first gives a system overview. Next, the implementation details for the genetic algorithm are presented and discussed. The wind-, wake-, and power model, fitness function and test scenarios are also given. Additionally, the design decisions made when implementing the 3 population distributed genetic algorithms are presented and explained.

The final results of the experiments are presented in chapter 5. The first part of the chapter contains the results of testing different parameter values, selection mechanisms and recombination mechanism. The second part of the chapter contains the results obtained when running the different genetic algorithm models on 4 different wind scenarios, 2 containing obstacles and 2 not containing obstacles. A discussion of the results are provided at the end of the chapter. The results show that the Master/Slave model is able to find better results for each of the 4 scenarios. The Pool model performs second best, the Island model third best and the Cellular model forth best.

6.2   Goal Evaluation

The goal of this thesis was to investigate the possible advantages of using population distributed genetic algorithms in optimizing wind farm layout. In this section, the research questions will be reviewed, answered and discussed.

Research question 1: *Can distributed genetic algorithms improve the quality of the solution to the wind farm layout optimization problem as compared to the simple genetic algorithm?*

The results from this thesis show that population distributed genetic algorithm does not necessarily find better solutions than the simple genetic algorithm for the wind farm layout optimization problem. The Master/Slave model consistently found better wind farm layouts than the population distributed genetic algorithms.

As mentioned in chapter 1 the wind farm layout optimization problem is impossible to solve analytically because the search space is too big. For a wind farm site with 500 legal turbine positions 2^{500} different layouts exist, that is, more layouts than the number of particles in the universe! This analogy explains just how complex the wind farm layout optimization problem is, and it illustrates how unlikely it is that a genetic algorithm is able to come up with the global optimal solution. The Master/Slave model is the model that spends less time exploring different number of turbines. It settles on a given number of turbines after approximately 10 generations and starts optimizing the turbine positions within that solution space. This strategy of settling for a local minima fast and optimize turbine positions for as many generations as possible seem to work! And, as the results imply, this might be the property that makes the Master/Slave model outperform the other models. This explanation is reinforced by the results obtained in section 5.1. It was shown that by decreasing the time it takes for the best solutions to monopolize the population, the algorithm was able to come up with better solutions.

Another possible explanation why the Master/Slave model obtains better results than the population distributed genetic algorithms could be that the genetic algorithm parameters and selection strategies was optimized running this model. It might be the case that the population distributed models could obtain better results if these settings where tailored for every model. Sadly, the time frame of this thesis prevented the author to perform these experiments for every model.

It is important to note that this thesis have only proven that the Master/Slave model performs better than population distributed genetic algorithm when each model is provided the same amount of resources and with the given

settings and design decisions used in this thesis. These results should not be interpreted as absolute, but that under the given circumstances the Master/Slave model is able to get better results. It could be argued that it is wrong to compare the population distributed genetic algorithms with the simple genetic algorithm with the same amount of resources. The essential property of population distributed genetic algorithms is that they spend more time exploring different number of turbines before they settle down on a given number and start optimizing turbine positions. It would be interesting to observe if letting the algorithms run for the same number of generations after they have settled on a given number of turbines would improve the results of the population distributed genetic algorithms. However, this seems unlikely for every model except from the Island model. As the results showed, the Island model was the only model that was able to explore different number of turbines for a significant amount of time, the other population distributed models settled almost as fast as the Master/Slave model. A problem with the suggested strategy is that it is very difficult to compare models without providing them with the same amount of resources.

Another fact that might explain why the population distributed genetic algorithms were not able to come up with results as good as the Master/Slave model could be that the parameters that are specific for a given model was not optimized. For the Pool model, the number of workers were not optimized, the number of Island, Island topology and migration strategy could be optimized for the Island model and the topology should also be optimized for the Cellular model. In this thesis these values are set based on the authors previous experience with genetic algorithms because there was not time to optimize these settings for each model.

Research question 2: *Which distributed genetic algorithm works best for the wind farm layout optimization problem? What properties are essential for its success?*

Out of the population distributed genetic algorithms, the Pool model was able to find the best solutions. The Island model also came up with acceptable results, while the Cellular model had trouble keeping up with the other models.

Above it was discussed that the property that made the Master/Slave model

superior to the other models could be that it does not waste time searching for the optimal number of turbines, but settles fast in a local minima, and spends most of its time optimizing wind turbine positions. The results obtained by the Pool model supports this explanation. The Pool model is the population distributed genetic algorithm that resembles the Master/Slave model most. The results show that it settles on a given number of turbines almost as fast as the Master/Slave model and spend an extensive amount of time optimizing turbines positions. The difference in fitness between the Master/Slave model and the Pool model is small enough so that it might seem reasonable to conclude that if the settings had been tailored for the Pool model, it might would be able to obtain similar results.

As mentioned, the Island model was the only model able to take advantage of its population distributed property by using it to explore different number of turbines. It was expected that this property would be crucial for the success of the populating distributed genetic algorithms. However, this thesis have proven that under the given circumstances this property was not an advantage. As discussed before a number of reasons could explain this result. First, it might be the case that this property is only advantageous when the model is allowed to run for more generations, and is allowed to optimize each solution longer. It might also be the case that this exploration is a waste of time because the algorithms are not able to find the global optimal solution, and should concentrate on optimizing a local minima.

The Cellular model was not able to come up with good solutions for the problem. The model performed so much worse than the other models that it is safe to say that it is not suited for wind farm layout optimization. It is structured in a way that requires way more time for good solutions to spread in the population that it would be too time consuming to use this model in wind farm optimization. The computational load is already extremely high, so increasing the number of generations so extensively that the Cellular model could stand a chance against the other models is too time consuming, at least on a personal computer.

In summary, genetic algorithms have been used trying to solve the wind farm layout optimization problem for more than 20 years. Genetic search algorithms have become the primary artificial intelligence technique in wind farm optimization because the problem is too complex to solve analytically. How-

ever, the complexity of the problem is also apparent when genetic algorithms are used. It is shown that properties that usually are advantageous for genetic algorithms such as the property of slowing down convergence in order to explore different solutions might not benefit in wind farm optimization. However, the suggestions presented in 6.4 should be investigated before firm conclusions can be made.

6.3   Contributions

The research presented in chapter 3 have shown that genetic algorithms are suitable for wind farm layout optimization. Improvements such as improvements to the genetic algorithm, the fitness function, and the power model, have been investigated. [Grady et al., 2005] showed that using the Island model on the scenarios provided by [Mosetti et al., 1994] lead to better results. However, no attempt, as far as the author knows, have been done in investigating the performance of the Pool model and Cellular model on the wind farm layout optimization problem. Therefore, this thesis contributes to the field by being the first project attempting to show which, if any, of the population distributed genetic algorithms are best equipped for wind farm layout optimization. The results obtained in this thesis can be used as a basis for further research within the field. Section 6.4 contains suggestions on how the research presented in this thesis could be investigated further. Many interesting directions can, and should, be taken in order to find out which method is truly the primary method for wind farm layout optimization.

The main lesson from this project:

> *Using population distributed genetic algorithms to solve the wind farm layout optimization problem will not necessarily lead to better fitness than with simple genetic algorithms. Population distributed genetic algorithm demand more computational resources and will therefore have trouble beating a simple genetic algorithm with the same amount of resources. Population distributed genetic algorithms are more advanced than simple genetic algorithms and it will therefore be more difficult to find the correct settings for*

them. However, when handed more resources it is clear that popu-
lation distributed genetic algorithms are powerful algorithms that
should be investigated further on the wind farm layout optimiza-
tion problem.

6.4   Further work

As far as the author knows, this is the first time different population distributed genetic algorithms have been compared against each other and against a simple genetic algorithm in their ability to solve the wind farm layout optimization problem. This thesis can be view as a first attempt to understand how population distributed genetic algorithm can affect wind farm layout optimization, and it should be explored further.

Both research questions should be further investigated. As has been mentioned, further work should be done in optimizing the parameters that are specific for the each of the population distributed models. For the Island model experiments should be performed to optimize *deme size*, *deme count*, *migration rate*, *migration interval* and the *topology*. Especially, the author believes that optimizing the *migration interval* would have a major impact on the performance of the Island model. As the results showed, the Island model was the only model able to explore different number of turbines for a significant amount of time. However, it was not given enough time to optimize each of these number of turbines, and therefore it was not able to realize the potential of the solution it has found before it jumped to another solution. Increasing the migration interval might have changed the final results. For the Cellular model, different topologies should be investigated and compared. A larger square, or another form of topology would affect the results. The replacement strategy of replacing every individual at every generation even thought the new individual had worse fitness than the previous individual should probably be revised. It might also be the case that a different parent selection function, such as roulette wheel, would work better for this model, even though it did not work for the Master/Slave model. For the Pool model, different number of workers should be tested. It would also be very interesting to take control over the scheduling of the different workers, and investigate the effect of letting one or more of the workers to get far ahead of the others in evolution and compare this to simulations where the threads

are at about the same generation.

In addition to optimize the parameters that are specific for each model it would also be desirable to redo the parameter settings from section 5.1 for each of the population distributed genetic algorithms. This would make sure that the Master/Slave model does not have an advantage because the settings used for every model is only tested on that model. Even though the author does not believe this part to be as important as the improvements recommended above, it could have an impact on the final results.

All 3 population distributed genetic algorithms were implemented as homogeneous models. It would be very interesting to observe how implementing these as heterogeneous models would affect the results. This would require an extensive amount of work because so many different settings could be used, that it might be wise to start with only one of the models. It would be very interesting to observe how the Island model could be improved by using different parameter settings, selection mechanisms and recombination methods on each Island and compare the results against a homogeneous implementation. It would also be very interesting to let the workers of the Pool model run with different settings, and to let different settings apply to different parts of the grid in the Cellular model.

The Island model and Cellular model were both synchronous. Implementing these as asynchronous models could also be interesting to investigate, although the author does not believe that this would impact the results as much as implementing heterogeneous models.

Since this thesis is a contribution to GECCO 2016, where the goal is to optimize the objective function provided by the contest a multi-objective function was not implemented. However, it would be very interesting to solve the wind farm layout optimization problem with a multi-objective genetic algorithm, and combine this with population distributed genetic algorithms. Finding the best objective function is a difficult job, which includes a lot of estimation. It is extremely difficult to come up with a fitness function that guide the population to the optimal solution. Because of this, it is often more accurate to use a multi-objective function because then you do not have to weight the importance of the different objective against each other. A population distributed genetic algorithm would be perfect for optimizing a mulit-objective

function because each Island, worker or area could be assigned a different objective function.

[Gong et al., 2015] also present population distributed genetic algorithms which are combinations of each other. The Island model implemented for this thesis was actually one of the models mentioned, because the Master/Slave model was run on each Island. However, the population distributed genetic algorithms were not combined. [Gong et al., 2015] suggest combining the Island model with the Cellular model (one cellular model on every Island) or running an Island model on each Island in the Island model. These models go even further in their attempt to slow down take over time, and investigate different solutions.

[Saavedra-Morena et al., 2011] used a greedy heuristic to decide the initial position of the wind turbines, and then used the genetic algorithm to improve the results. [Saavedra-Morena et al., 2011] showed that the final results were better than those obtained by the genetic algorithm when it started out with random solutions. This strategy could also be implemented together with the models implemented in this thesis to investigate if this would improve the final results obtain by the population distributed genetic algorithms.

In section 3.2 it was shown that greedy heuristics, simulated annealing, ant colony algorithms and other swarm algorithms could also be used to solve the wind farm layout optimization problem. A study comparing different artificial intelligence methods in solving the wind farm layout optimization problem would be very interesting. It would also be possible to combine the different algorithms either by letting a genetic algorithm run first at trying to let one of the other methods improve the final results, or the other way around.

As discussed, this thesis has answered some questions and left the author with many new questions that it would be very interesting to investigated and answer. This thesis has laid the foundation for further work within population distributed genetic algorithms and wind farm layout optimization.

Bibliography

Albring, W. (1967). Betz, introduction to the theory of flow machines. *Journal of Applied Mathematics and Mechanics*, 47(2):140–141.

Bilbao, M. and Alba, E. (2009). Simulated annealing for optimization of wind farm annual profit. In *Logistics and Industrial Informatics, 2009. LINDI 2009. 2nd International*, pages 1–5. IEEE.

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

EIA (2015). Accessed December 2015. www.eia.gov.

Emami, A. and Noghreh, P. (2010). New approach on optimization in placement of wind turbines within wind farm by genetic algorithms. *Renewable Energy*.

Eroğlu, Y. and Seçkiner, S. U. (2012). Design of wind farm layout using ant colony algorithm. *Renewable Energy*, 44:53–62.

Frandsen, S. T. et al. (2007). *Turbulence and turbulence-generated structural loading in wind turbine clusters*. Risø National Laboratory.

Gao, X., Yang, H., Lin, L., and Koo, P. (2015). Wind turbine layout optimization using multi-population genetic algorithm and a case study in hong kong offshore. *Journal of Wind Engineering and Industrial Aerodynamics*, 139:89–99.

Goldberg, D. E. (2005). *Genetic Algorithms in Search, Optimzation and Machine Learning*. Addison-Wesley Publishing Company.

Gong, Y.-J., Chen, W.-N., Zhan, Z.-H., Zhang, J., Li, Y., Zhang, Q., and Li, J.-J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*.

González, J. S., Rodriguez, A. G. G., Mora, J. C., Santos, J. R., and Payan, M. B. (2010). Optimization of wind farm turbines layout using an evolutive algorithm. *Renewable Energy*.

Grady, S. A., Hussaini, M. Y., and Abdullah, M. M. (2005). Placement of wind turbines using genetic algorithms. *Renewable Energy*.

Holland, J. H. (1992). *Adaptation in Natural And Artificial Systems*. The MIT Press.

Huang, H.-S. (2007). Distributed genetic algorithm for optimization of wind farm annual profits. *Intelligent Systems Applications to Power Systems*.

IRIT (2015). Accessed December 2015. www.irit.fr.

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

Justus, C., Hargraves, W., Mikhail, A., and Graber, D. (1978). Methods for estimating wind speed frequency distributions. *Journal of applied meteorology*, 17(3):350–353.

Katic, I., Højstrup, J., and Jensen, N. (1986). *A simple model for cluster efficiency*. European Wind Energy Association Conference and Exhibition.

Kusiak, A. and Song, Z. (2010). Design of wind farm layout for maximum wind energy capture. *Renewable Energy*, 35(3):685–694.

Liang, S. and Fang, Y. (2014). Analysis of the jensen's model, the frandsen's model and their gaussian variations. In *Electrical Machines and Systems (ICEMS), 2014 17th International Conference on*, pages 3213–3219. IEEE.

Magalhaes-Mendes, J. (2013). A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem. *WSEAS Trans. Comput*, 12(4):164–173.

Mora, J. C., Barón, J. M. C., Santos, J. M. R., and Payán, M. B. (2007). An evolutive algorithm for wind farm optimal design. *Neurocomputing*.

101

Mosetti, G., Poloni, C., and Diviacco, B. (1994). Optimization of wind turbine positioning in large windfarms by means of genetic algorithm. *Journal of Wind Engineering and Industrial Aerodynamics*.

Murata, T., Ishibuchi, H., and Gen, M. (2001). Specification of genetic search directions in cellular multi-objective genetic algorithms. In *Evolutionary multi-criterion optimization*, pages 82–95. Springer.

Ozturk, U. A. and Norman, B. A. (2004). Heuristic methods for wind energy conversion system positioning. *Electric Power Systems Research*, 70(3):179–185.

Razali, N. M. and Geraghty, J. (2011). Genetic algorithm performance with different selection strategies in solving tsp.

Saavedra-Morena, B., Salcedo-Sanz, S., Paniagua-Tineo, A., Prieto, L., and Portilla-Figueras, A. (2011). Seeding evolutionary algorithms with heuristics for optimal wind turbines positioning in wind farms. *Renewable Energy*.

Samorani, M. (2013). *The Handbook of Wind Power Systems*, chapter The Wind Farm Layout Optimization Problem. Springer Berlin Heidelberg.

Şişbot, S., Turgut, Ö., Tunç, M., and Çamdalı, Ü. (2010). Optimal positioning of wind turbines on gökçeada using multi-objective genetic algorithm. *Wind Energy*, 13(4):297–306.

Soni, N. and Kumar, T. (2014). Study of various mutation operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*.

Wan, C., Wang, J., Yang, G., Gu, H., and Zhang, X. (2012). Wind farm micro-siting by gaussian particle swarm optimization with local search strategy. *Renewable Energy*, 48:276–286.

Wan, C., Wang, J., Yang, G., Li, X., and Zhang, X. (2009). Optimal micro-siting of wind turbines by genetic algorithms based on improved wind and turbine models. *Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*.

Zhao, M., Cheng, Z., and Hjerrild, J. (2006). Analysis of the behaviour of genetic algorithms applied in optimization of electrical system designs for offshore wind farms. *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on.*