# 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

# Preface

This thesis aims to show how genetic algorithms can contribute in solving the problem of positioning wind turbines in a wind farm in an optimal way. Specifically, population distributed genetic algorithms will be implemented and tested in order to find out if their properties can contribute to find an even better solution than simple genetic algorithms.

- Assignment given: August 20th, 2015.
- Supervisor: Professor Keith Downing.

# Contents

# Chapter 1

# Introduction

This thesis is a contribution to a contest launched by the annual international conference of evolutionary computation (GECCO) 2015, involving optimizing the number of turbines, and turbine positions in a wind farm with goal of producing maximum power to minimum cost. A wind farm simulator is provided by GECCO 2015, and therefor will the focus of this thesis be on improving the genetic algorithm that will be used to search for the optimal turbine positioning. The background and motivation behind the thesis is presented in section 1.1, section 1.2 introduces the goal, and research questions that will be answered, and section 1.3 gives an overview over the rest of the thesis.

## 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 that non-renewable energy and should therefore be preferred. However, renewable energy sources make up only 21% of the worlds energy sources as of 2011 [EIA, 2015]. Wind turbine technology is a promising source of renewable energy. Wind turbine technology advances has led to wind turbines able to produce more energy to lower costs. However, wind turbines still produces 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 worlds 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 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 constrains 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), 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, therefore, the focus of this thesis will be on optimizing the genetic algorithm. Still, knowledge of wind turbines, wind farm layout and wake models is useful in understanding the 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 have 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 authors have implemented the population distributed genetic algorithms called the Island model, and it has shown promising results. However, as far as the author know, no attempt has been made in implementing any of the other population distributed models. This fact is the main motivation behind this thesis, and have inspired the author to investigate the effect different population distributed algorithms can have on the wind farm layout optimization problem.

## 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 distributed genetic algorithms in optimizing wind farm layout, i.e. solving the wind farm layout optimization problem.* [Samorani, 2013]

The performance of distributed genetic algorithms will be studied and compared to the performance of a simple genetic algorithm (not population distributed) as well as to each other, with the goal of answering the research questions stated below.

**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 2015, on twenty different wind scenarios which are also provided by the contest. Research question 1 will be answered by implementing different types of population distributed genetic algorithms and compare their results with a simple genetic algorithm on each wind scenario. Research question 2 will be tested the same way, and the results of each of the population distributed algorithms will be compared.

## 1.3 Thesis Structure

The thesis is divided into four chapters. Chapter 2 contains an introduction to the wind farm layout optimization problem, a description of how the genetic algorithm works, and a description of each of the population distributed genetic algorithms that will be implemented. Chapter 3 is a survey of the state of the art within wind farm layout optimization, one section describing approaches using genetic algorithms, and one section describing other approaches. Chapter **??** is a description of the application user interface provided by GECCO 2015 that will be extended with different distributed genetic algorithm implementations. It also contains test simulations, and a discussion of future work.

# Chapter 2

# Background

In this chapter the wind farm layout optimization problem will be defined and explained in section 2.1. Section 2.2 is partitioned into two subsections. In 2.2.1 the simple genetic algorithm is explained, and in 2.2.2 the seven distributed genetic algorithms that will be implemented in this thesis are described.

## 2.1 The Wind Farm Layout Optimization Problem

The goal of this section is to give the reader an understanding of the wind farm layout optimization problem, and explain the key factors that makes the problem so complex.

### 2.1.1 Definition of 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 cost. 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 challenge 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 later. 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". However, in this thesis, the problem formulation will be extended to include cost constraints and also the problem of deciding 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 within a given budget."*

### 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. Sites are classified in 7 different wind power classes, where sites with power class 4 or higher are suitable for hosting a wind farm with today's turbine technology. But, even though the wind farm has the required wind conditions, it might not be suitable for hosting a find farm after all, 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 albe to handle the transportation trucks. Second, land owner 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. 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 similar all year long, but if the wind conditions vary extensively over the year this step can take a few years.

An evenly important step is to decide on which turbines to buy for the wind farm. A tradeoff exists between power and cost, since larger turbines usually

generate more power, but they are also more expensive than smaller ones. 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.

After the site is found, turbine type is decided and wind distribution is measured, the layout optimization can begin. Layout optimization faces different challenges, such as positions of the terrain that contain obstacles so that turbines cannot be positioned there. There are also constraint 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]. Other wake models exist, but most research in wind farm layout optimization use the Jensen model because it is quite accurate and simple. The Jensen model will be explained intuitively below in order to give a brief understanding of how the wake effect is calculated.

In figure 2.1 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 $\alpha$ 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, time consuming process. In order to even start the layout optimization process consecutive important decisions has to be made. The layout optimization is dependent

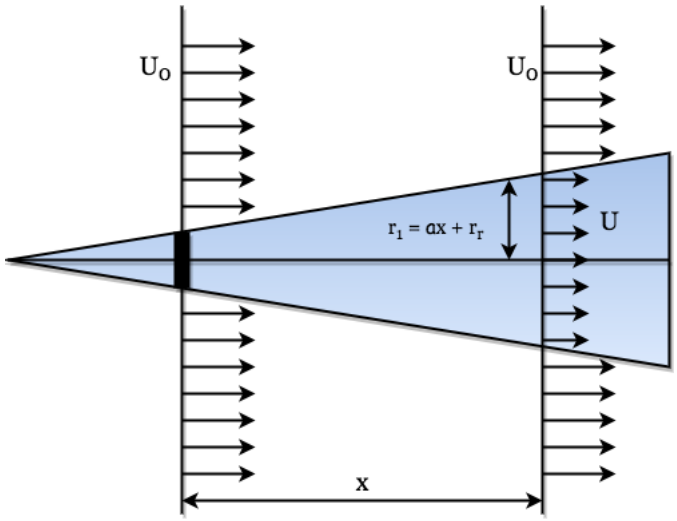



Figure 2.1: The wake effect [Samorani, 2013].

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 first explains the simple genetic algorithm (SGA), invented by Holland [1992]. If the reader is familiar with the genetic algorithm he or she can skip the first part. If not otherwise stated, the first part is based on references [Holland, 1992] and [Goldberg, 2005]. Next, the seven different distributed genetic algorithms that will be implemented in this thesis are explained, these are all taken from Gong et al. [2015].

### 2.2.1 Simple Genetic Algorithms (SGAs)

Genetic algorithms are probabilistic search algorithms inspired by evolution. Figure 2.2 gives an intuitive explanation of how the algorithm works. The genetic algorithm operates on a population of individuals each representing a solution to a problem. Usually, the initial population consist of randomly generated individuals which become the first child population. For each

generation, the child population is evaluated based on some predefined fitness function (objective function), and the fittest individuals are selected as parents for the next generation. Note that the terms fitness function and objective function will be used interchangeably in this thesis. Next, the parent population produces a new child population based on different reproduction schemes, such as recombination of parent genes to form child genes. Some genes are also altered in the process. Finally, the next generation of child solutions are generated and the process starts again.

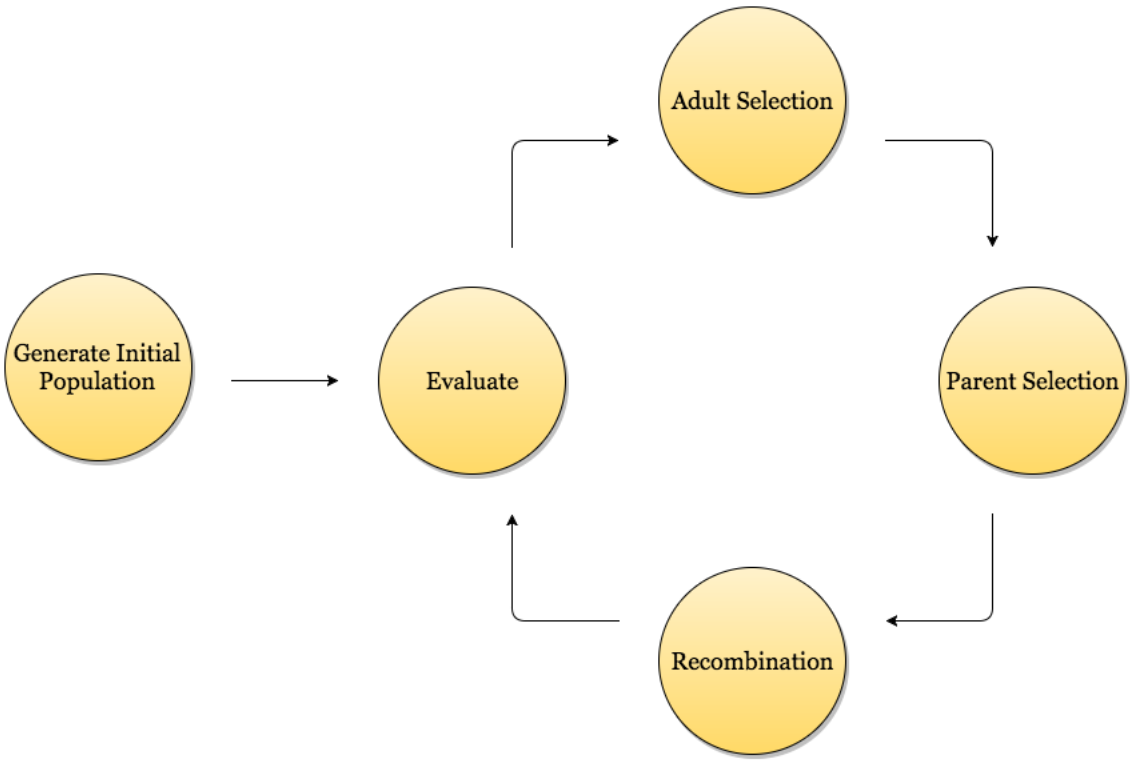



Figure 2.2: Overview of the phases of the genetic algorithm.

Inspired by survival of the fittest, the population evolves into a population of better solutions to the given problem. Two key properties are crucial for the utilization and success of the genetic algorithm; (1) there has to be a way to evaluate the fitness of the solutions, and (2) there has to be a way to represent individuals so that genetic operations can be performed on them. Examples

of representation, fitness calculation, selection processes, and genetic operations will be given below. Note that there exists numerous different selection schemes and ways to perform mutation and crossover (genetic operations), but here, only the types that will be used in the given thesis are presented.

### 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 an 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 2.3 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.

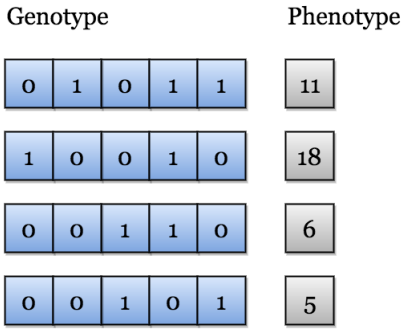



Figure 2.3: Genotypes and phenotypes for four individuals where the phenotype is the decimal value of the genotype (binary number).

### Selection

Selection is the process of selecting which individuals from a given population that will be the parents of the next generation. The simplest form of selection

is called *elitist selection*, meaning the best (highest fitness) individuals from the populations are selected. 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 2.4 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. The individual with fitness 9 wins the tournament and is chosen for reproduction.

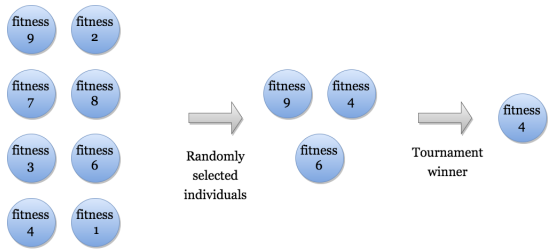



Figure 2.4: Tournament selection. A group of three individuals are randomly drawn from the pool of all individuals. The best individual in the group, the one with fitness 4, is selected for reproduction [Razali and Geraghty, 2011].

By varying the value of $n$ you can control how much exploration your algorithm should do. If $n$ is equal to the population size, this is elitist selection, 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, and higher values of $n$ leads to faster convergence. 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.

### Crossover

Crossover means combining genes of a parent solution to produce a child. The crossover scheme used in this thesis uses 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 2.5 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.

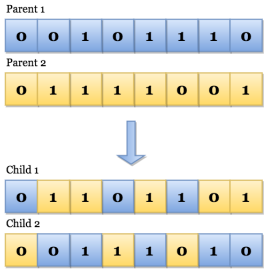



Figure 2.5: Uniform crossover. A child genotype is created by a combination of the genotypes of both parents. Each gene is drawn from one of the parents with equal probability.

### 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 2.6.

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, reproduction will never be able to make a new individual that doesn't 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

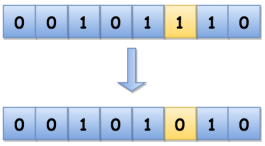



Figure 2.6: Mutation of a single bit. The bit in position 6 at the upper bit string has the value 1 before the mutation, while after mutation the value is flipped to 0.

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 does 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 genetic algorithms where processing is distributed among different processors, meaning that parallelism is utilized in the implementation. This section introduces seven distributed models. Each model will be implemented with parallelism in order to improve the running time of the algorithm, however the first model introduced, 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 seven different distributed algorithms presented by Gong et al. [2015]. These will all be implemented and tested in this thesis.

### Master-Slave Model

As mentioned, 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 2.7.

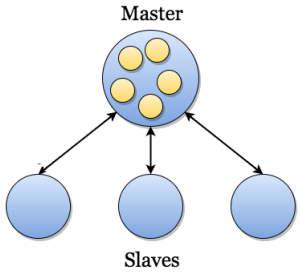



Figure 2.7: Master-slave model. The master process distributes the population to different slave processes, which calculate the fitness of each individual and return the results to the master process [Gong et al., 2015].

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.

### 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 2.8 displays a population divided into four sub-populations.

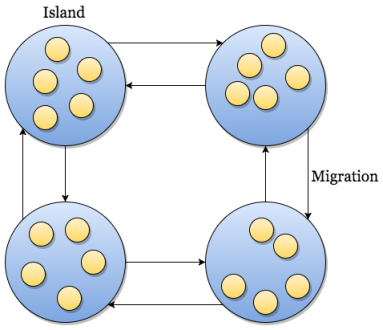



Figure 2.8: An island model using a ring topology with four demes (Islands) of size five. [Gong et al., 2015]

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 2.8 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 2.8 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 2.9 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.

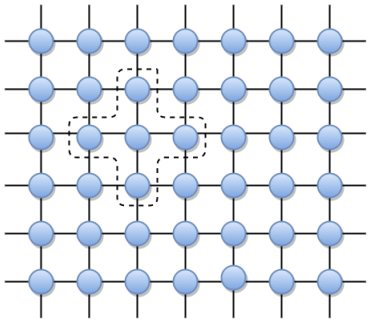


Figure 2.9: Cellular model where the neighborhood topology consist of the cells to the left, right, over and under the given cell [Gong et al., 2015].

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 2.9 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

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 2.10.

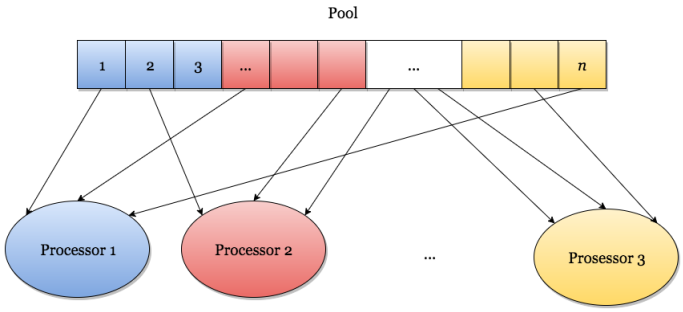



Figure 2.10: The pool model. Each processor has its own positions in the pool which it returns individuals to. The red processor is responsible for the red positions in the pool, and so on. Processors draw individuals from random positions in the pool, as indicated by the arrows, but can only write them 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].

A processor $p_1$ is responsible for $k$ positions in the pool. This is indicated by the coloring scheme in figure 2.10, 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.

In summary, this chapter has introduced the wind farm layout optimization problem and explained its complexity. A simple genetic algorithms has been introduced, with the selection-, crossover- and mutations types that will be used in this thesis. Last, the seven 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, the three hybrid models 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.

# 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. 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 model a wind farm, a wake model, a power curve and a cost function needs to be specified. 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 function 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 $10 \times 10$ 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 park for a wind farm partitioned into 100 cells. The genetic algorithm used was a simple, single-population genetic algorithm where the fittest individuals were selected for reproduction. 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.

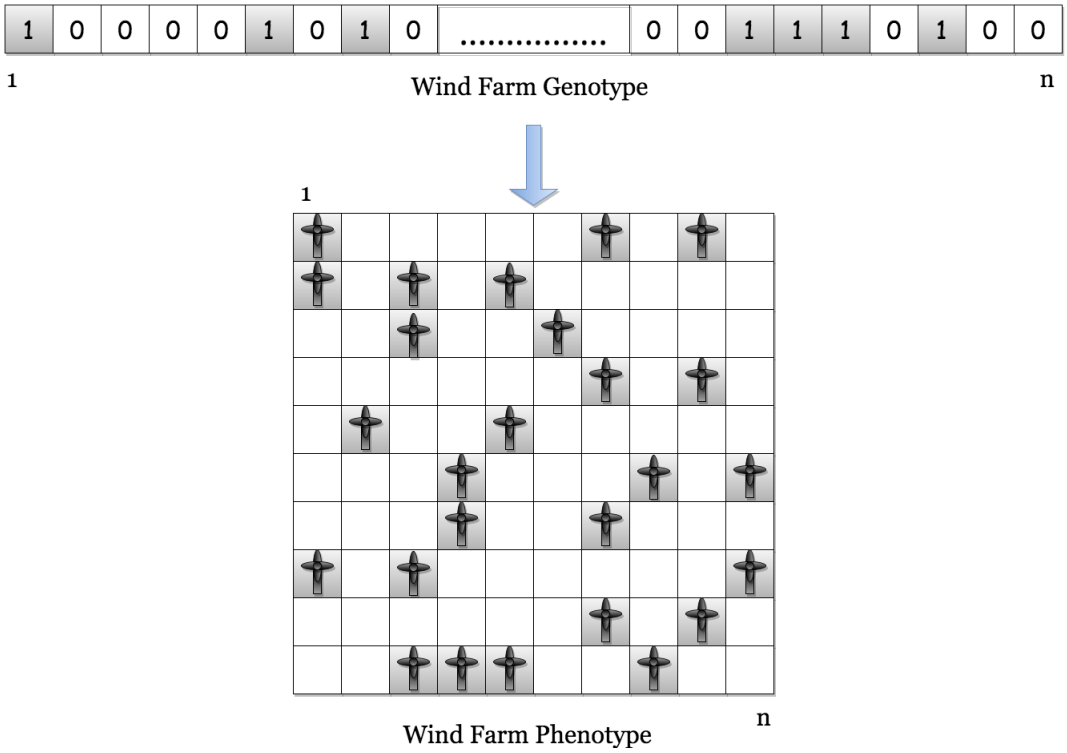



Figure 3.1: An example of how the wind farm is represented in the genetic search from Mosetti et al. [1994]. An individual is represented as a bit-string of size 100, where each cell can either contain the value 1 or 0, representing a position containing a turbine and a position not containing a turbine, 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 | $2.57 \times 10^{-3}$ | 50 |
| | Optimized | 0.95 | 12375 | $1.57 \times 10^{-3}$ | 25 |
| (b) | Random | 0.35 | 9117 | $3.68 \times 10^{-3}$ | 50 |
| | Optimized | 0.88 | 8711 | $1.84 \times 10^{-3}$ | 19 |
| (c) | Random | 0.34 | 4767 | $7.04 \times 10^{-3}$ | 50 |
| | Optimized | 0.84 | 3695 | $3.61 \times 10^{-3}$ | 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.

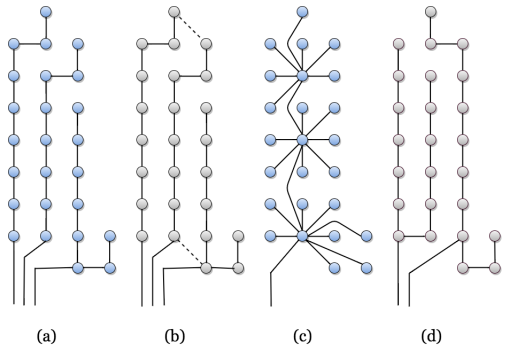



Figure 3.2: Four example clustering designs [Zhao et al., 2006].

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

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.

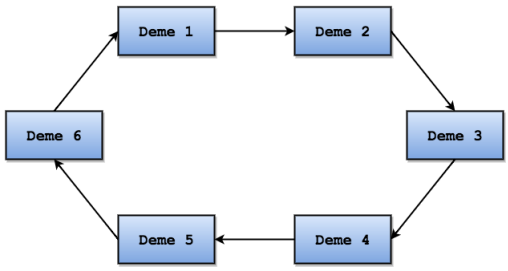



Figure 3.3: Ring topology example with 6 demes [Huang, 2007].

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.

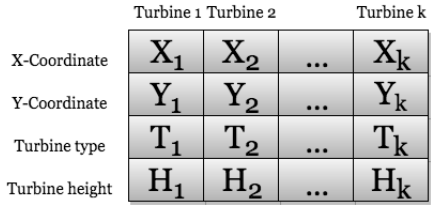



Figure 3.4: Representation of an individual of length $k$ (layout with $k$ turbines), where the first row represents x-coordinates of the turbines, the second row y-coordinates, the third row turbine type, and the forth row turbine height [Mora et al., 2007].

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 section **??**, 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.

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 removed 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.

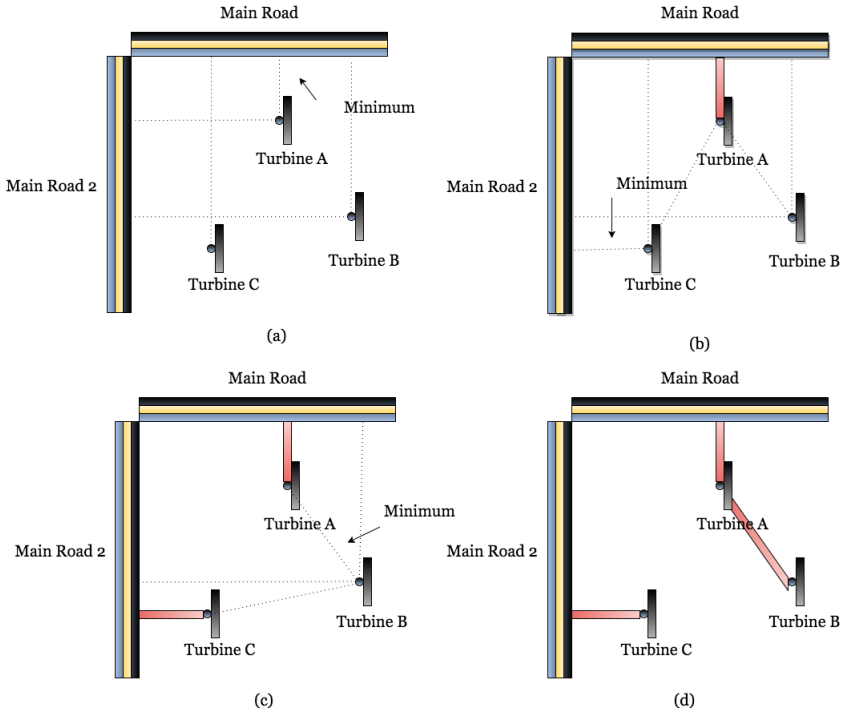



Figure 3.5: Greedy algorithm to estimate civil cost. [González et al., 2010].

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

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

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 even 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 is 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).

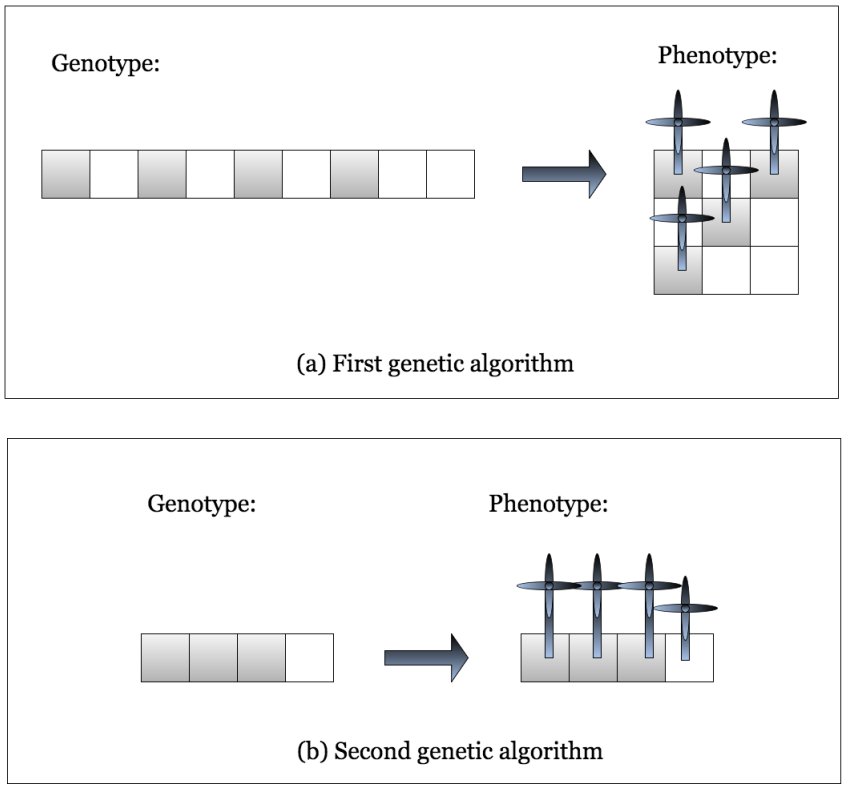



Figure 3.6: Representation for both genetic algorithms from [Chen et al., 2013]. (a) Binary string representing turbine positions, (b) binary string representing height of the given turbines positioned by the first genetic algorithm.

## 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 3.3. 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 3.3). 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 section **??**.

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 section **??**.

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 3.3 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. Section **??** 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.

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] | Jensen [1983] | Weibull distribution | Betz [Albring, 1967] | Complex | Complex | SGA | Integer | Seeded genetic algorithm. |
| Chen et al. [2013] | Frandsen et al. [2007] | 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 4.3 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 4.4, 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.

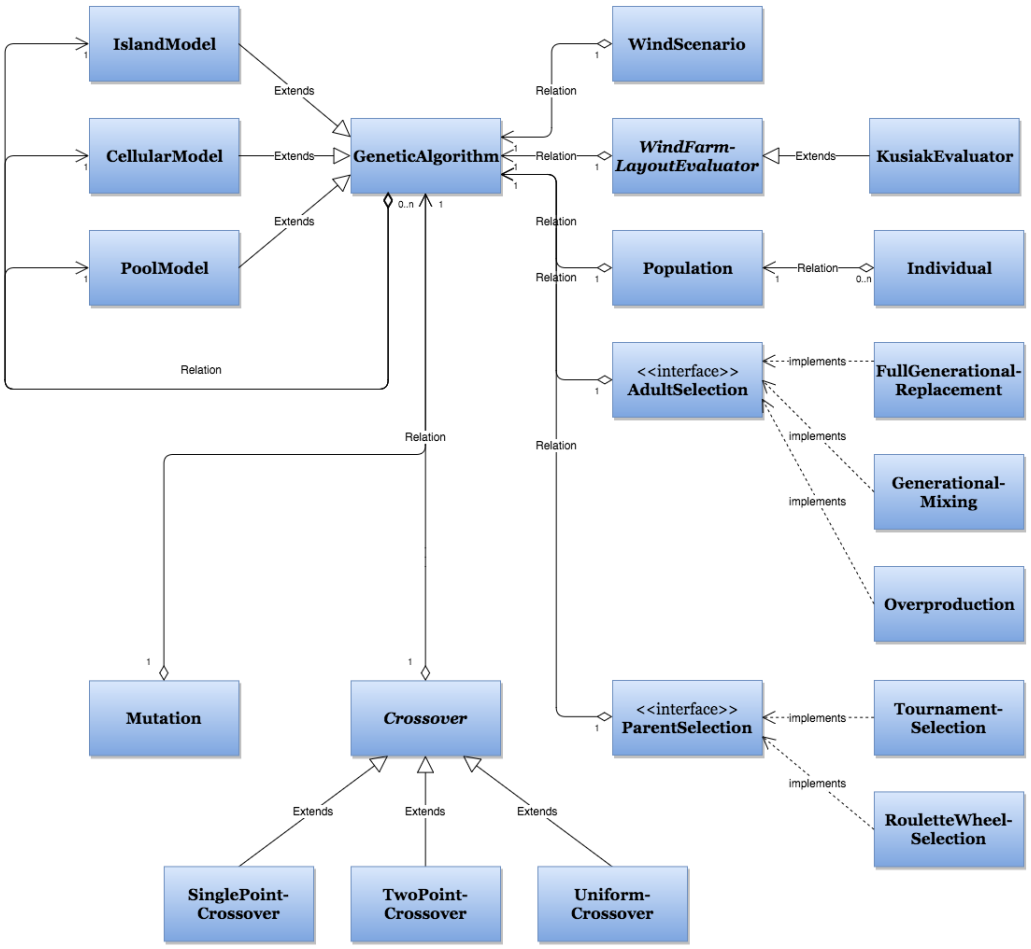



Figure 4.1: Class Diagram.

## 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.

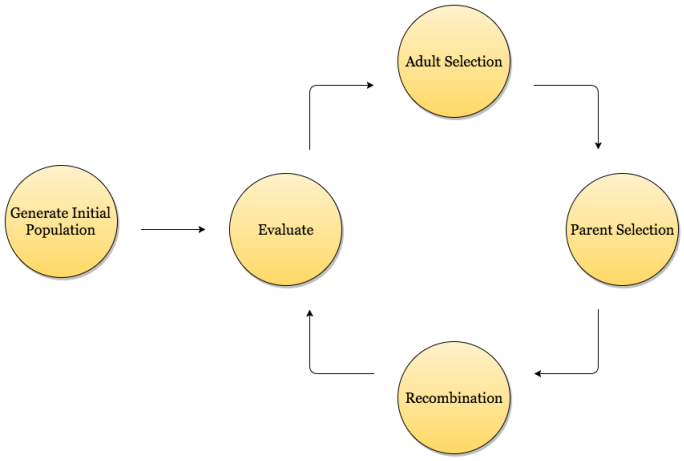



Figure 4.2: Genetic algorithm.

### 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.

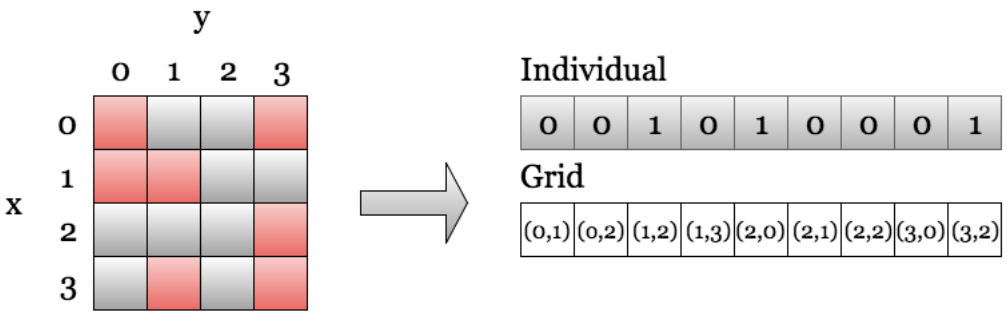



Figure 4.3: Individual representation. The grey squares represent legal turbine positions in the terrain, while the red represent illegal positions. Since there are only nine legal positions in this example, an individual is represented as a binary string of length 9. The grid array is shared between 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

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.

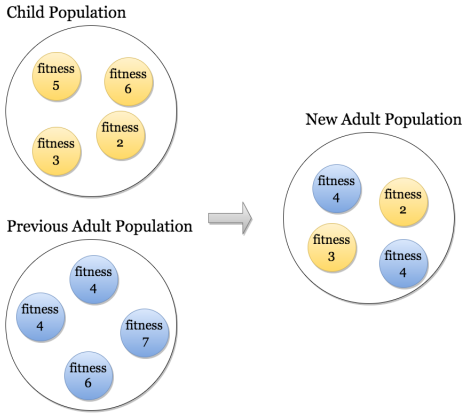



Figure 4.4: Generational mixing. The best individuals, from the pool consisting of individuals from the previous adult population and the new child population are selected as new the adult population.

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

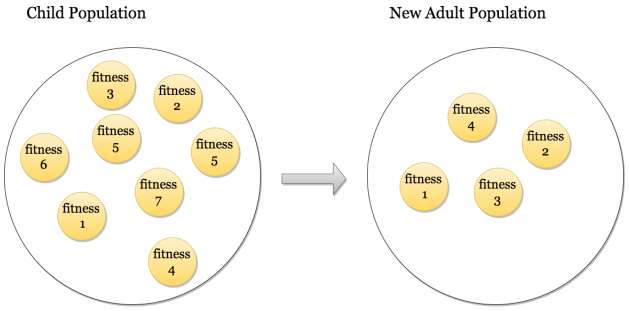



Figure 4.5: Overproduction. The newly generated child population consist of twice as many individuals as there are room for in the adult population. Therefore, only the fittest individuals from the large child population grow up into adults.

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

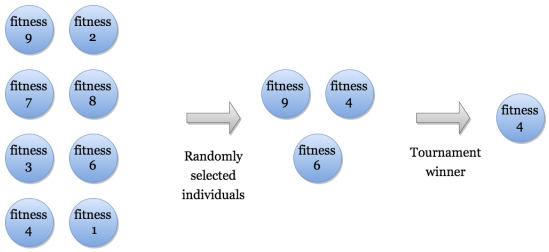



Figure 4.6: Tournament selection. Three individuals with fitness 9, 4 and 6 respectively, are selected randomly from the adult population. The individuals with best fitness, fitness 4, is the tournament winner and is selected into the parent pool.

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.

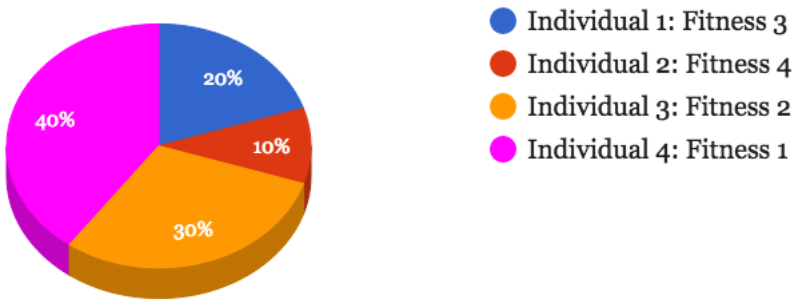


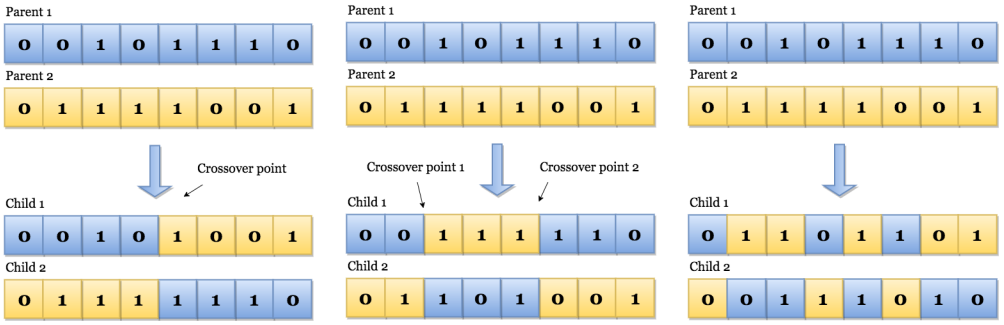

Figure 4.7: Roulette wheel selection. The roulette wheel is shown to the left, the four individuals to the right. Individual$_4$ has a four times better fitness than individual$_2$ and therefore has a four times larger probability of being selected.

### 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. These are all presented in figure 4.8. As shown in figure 4.8a, single point crossover is a method where a random position, called the crossover point, is generated. 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 results in child individuals that might not inherit the gene patterns from their parents the same way as in the other two methods.

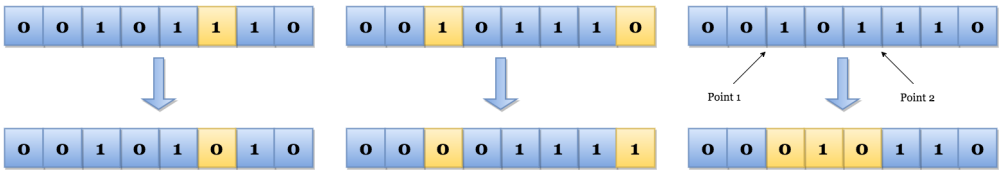



(a) (b) (c)

Figure 4.8: Crossover methods: (a) Single point crossover, (b) two point crossover, and (c) uniform crossover.

#### 4.2.4.2 Mutation

Although crossover is a powerful genetic operator that able child individuals to inherit genes from the best parent individuals, it is not enough on its own. 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 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, four mutation methods are implemented to mimic the mutations methods that can happen in genes, 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 imply, flip mutation works by flipping the value of single 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 in 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 3.3.

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 2015, $k \approx 2$, 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 $c$.

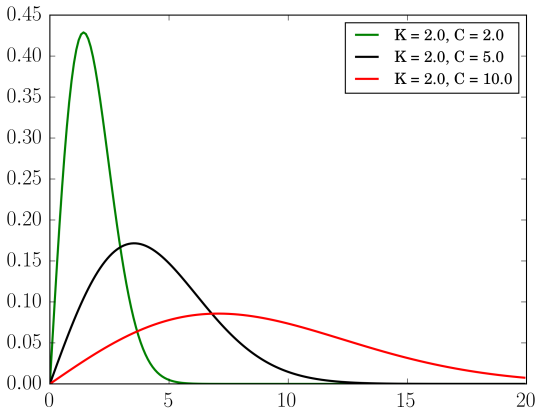



Figure 4.10: The Weibull distribution plotted for $k = 2$ for different values of the scale parameter $c$.

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 2015, ten which simply specify wind distribution parameters, and ten 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 a 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 $\lambda$ is the slope parameter, $v$ the wind speed, $\eta$ 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, and is displayed in equation 4.3.

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

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 Population Distributed Genetic Algorithms

Since all the population distributed genetic algorithms implemented are described in chapter 2 they will not be described in detail here. However, each design decision will be described. As mentioned in chapter 2, for the Island model, one have to make the decision if migration is synchronous or asynchronous, meaning that all migration is performed at the same generation. For the Island model implemented in this thesis, that is the case. As the Island model, the Cellular model will also perform updating at the same time, i.e. at the same generation. The pool model on the other hand is asynchronous by nature. Each processor performs its own assignment without even knowing about the presence of the other processors. The second design decision made is whether each model is homogeneous or heterogeneous, where homogeneous means that each deme, cell or processor uses the same

parameter settings and methods. In this thesis, each model will be homogeneous. This decision is based solidly on the fact that there will be too many combinations of different settings if each deme, cell or processor used different settings that it would be impossible to do proper testing for all of them.

## 4.4 Scenarios

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 one and the blue lines gives the boundaries of obstacle two. 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.

## 4.5 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

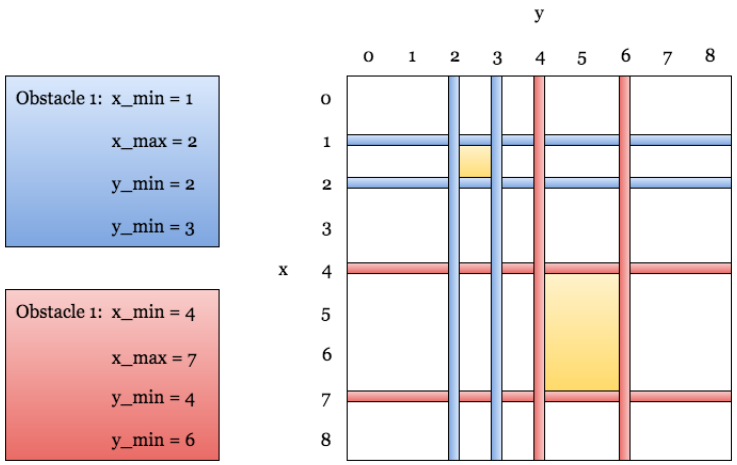



Figure 4.11: Obstacles are described in the scenarios with its (x, y)-boundaries.

changes and to 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 which 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.

# 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 different settings might work better on different problems and the nature of the wind farm layout optimization problem is unique because of the importance of the wind turbine positions relative to each other. It is important to note that even though effort was made to find good settings for the genetic algorithm, it is impossible to obtain the optimal ones. Just imagine trying the test every single value for the continuous parameter crossover rate against every possible value of each of the other settings, just that would be impossible! Therefore, the values that are tested for each parameter setting is largely tested against values that are

Table 5.1: Values kept fixed while one by one was changed in order to find the best settings for the wind 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 |
| Reversing mutation rate | 0.0 |
| Parent selection | Tournament selection |
| Tournament size | 5 |
| Epsilon | 0.0 |
| Crossover method | Uniform crossover |
| Crossover rate | 0.9 |

based on the authors previous experience with genetic algorithm and values close to those.

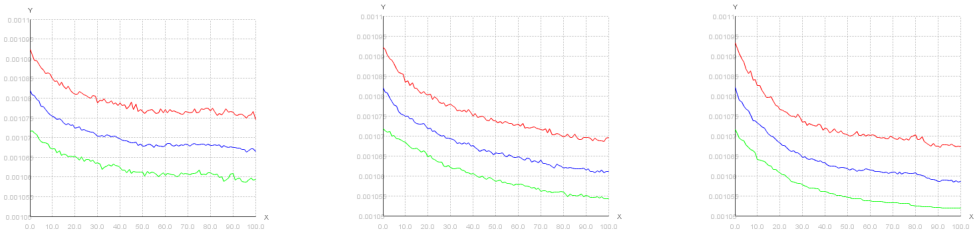
For each simulation one of the settings were tested while the others were kept fixed as shown in table 5.1. As can be seen in this table, other settings such as wind scenario, population size, number of generations, whether elitism should be used and mutation rate for interchange mutation and inversion mutation could also be tested, but since evaluation of each farm takes a lot of time, even on an 8 core computer running in parallel, simulations are not run to set these values. These settings are set using the authors previous experience and educated guessing. Testing different parameters 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 genetic algorithm largely. Greater population size often leads to better performance since more solutions increase the probability of finding the global optimal solution. 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 are 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, evaluating each population for 100 generations takes time and it the main reason why the algorithm were not run for more generations, however, as can be seen on the graphs in the sections below the fitness looks like it is starting to flat out after a 100 generations indicating that a 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 for of mutation in this thesis. Inversion mutation and interchange mutation are implemented, but they will only happen seldom 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 make occasional "jumps" to different solutions spaces so that the algorithm do not get completely stuck in a local optimal solution.

### 5.1.1 Adult Selection

Figure **??** shows the results of running the genetic algorithm with the different adult selection methods. Figure **??**, 4.4, and 4.5 shows the results for full generational replacement, generational mixing and overproduction respectively. As can be seen in the figures, full generational replacement ends up with an average best fitness of average of best fitness, generational mixing with an average best fitness of average of best fitness, and overproduction with an average best fitness of average of best fitness. These results are as expected. Note that generational mixing could never do worse than full generational replacement. If the results of reproduction leads to a population of individuals with strictly better fitness, generational mixing will replace the entire previous generation with the new one, as full generational replacement.

However, if the new generation produces individuals with worse fitness than some of the individuals in the previous generation these old individuals will be kept instead, making sure that the new population has equal or better fitness than the previous one. Overproduction does not provide the same "newer worst" Check word! guarantee, however, as shown in figure 4.5 it still outperforms generational mixing. Even though overproduction wipes out the entire previous population, it generates twice as many children at the reproduction step so its probability of getting it right is doubled.



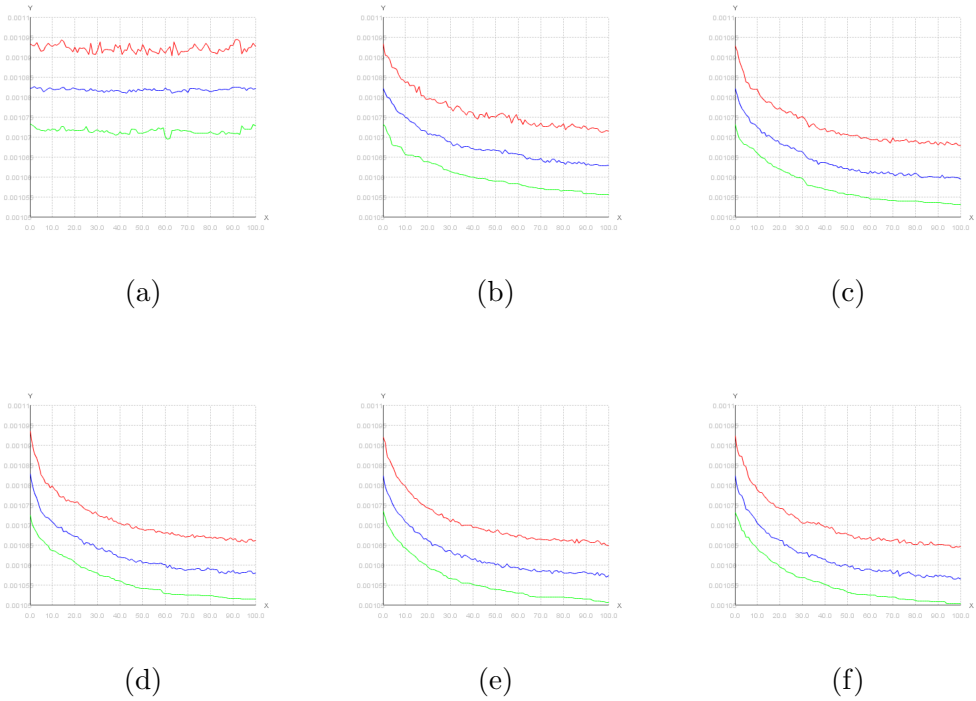
(a) (b) (c)

Figure 5.1: Adult selection methods: (a) Full generational replacement, (b) generational mixing and (c) overproduction averaged over 10 runs.

### 5.1.2 Parent Selection

### 5.1.3 Crossover Methods

Figure 4.8 displays the results from running the genetic algorithm with single point crossover, two point crossover and uniform crossover showed in sub figures 4.8a, 4.8b and 4.8c respectively. As can be seen in the figure, there is no crossover method that stands out, but end up with similar fitness. Single point crossover ends up with an averaged best fitness of averaged best fitness, two point crossover ends up with an averaged fitness of averaged best fitness and uniform crossover ends up with an averaged fitness of averaged best fitness. Intuitively, one would expect uniform crossover to perform worse than the others because it mixes up the relative positions between the wind turbines more than single- and two point crossover, however it actually obtains slightly better fitness. Even though it obtains slightly better fitness this

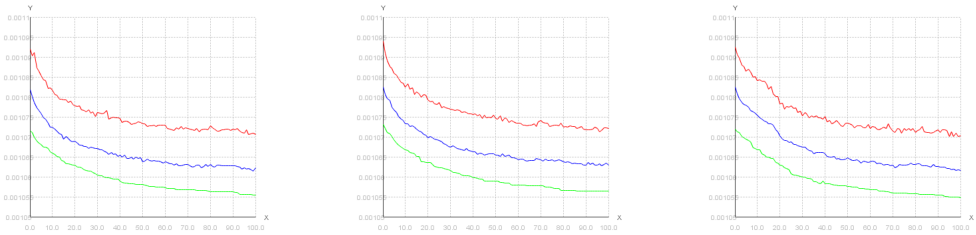


(a) (b) (c)

(d) (e) (f)

Figure 5.2: Parent selection methods: (a) Roulette wheel, (b) Tournament selection averaged over 10 runs, 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.

could be a coincidence since the different is so small, and the simulations are only averaged over ten runs. Explanation behind the results, talk to Keith. Stupid paragraph change everything!



(a) (b) (c)

Figure 5.3: Crossover methods averaged over 10 runs: (a) Single point crossover, (b) two point crossover and (c) uniform crossover.

### 5.1.4 Crossover Rate

### 5.1.5 Mutation Rate

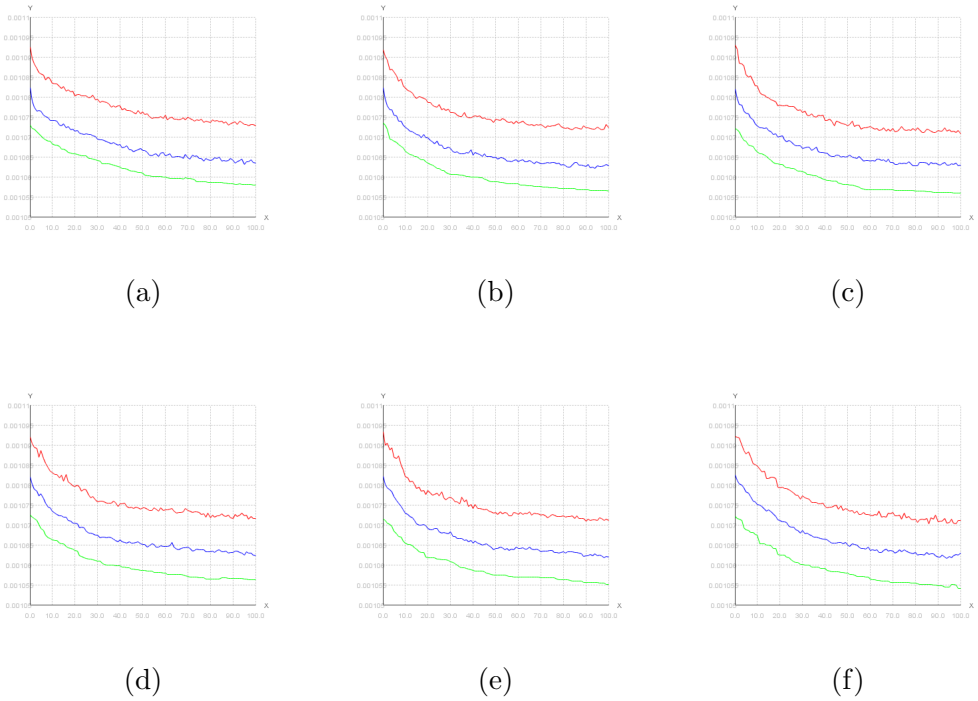
## 5.2 Results

In results: Write why population size kept 100: Overproduction lead to 200 and not point to get best solution, but to compare solutions.

### 5.2.1 Master Slave Model

Table 5.2: Results for the master slave model averaged over 10 runs. The first coloumn describes the scenario, the second, third, and forth gives best, averaged and worst fitness, the fifth, sixth, seventh and eight gives cost, power, efficiency and number of turbines for the best wind farm in the last population.

| Scenario | Best fitness | Avg. fitness | Worst fitness | Cost | Power | Efficiency |
|---|---|---|---|---|---|---|
| 00.xml | 0.0010398824104367332 | 0.0010433013036231363 | 0.0010521535224480483 | 2.5214864684269022E7 | 2.9443327731499153E10 | 0.843134544902673 |
| 05.xml | 0.0010400030013951972 | 0.0010436595078417523 | 0.0010524272799087857 | 2.5214864684269022E7 | 2.943909054887005E10 | 0.843017874402203 |
| 00obs.xml | 0.0010396895859709405 | 0.001043324775785618 | 0.0010520592949268786 | 2.5214864684269022E7 | 2.9449836783491558E10 | 0.843327487955956 |
| 05obs.xml | • | • | • | • | • | • |
| Average | • | • | • | • | • | • |

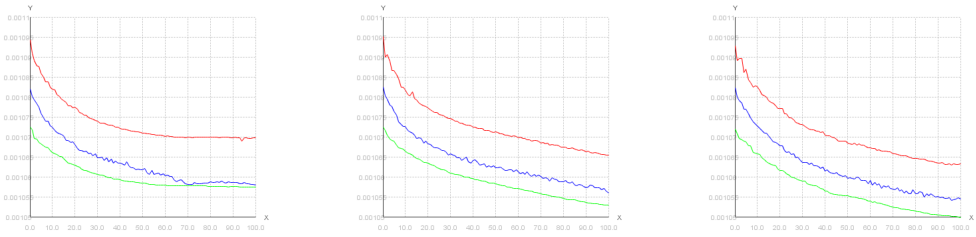


(a) (b) (c)

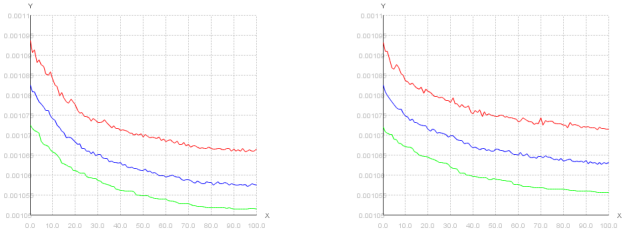
(d) (e) (f)

Figure 5.4: Crossover rates: (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.

## 5.3 Discussion

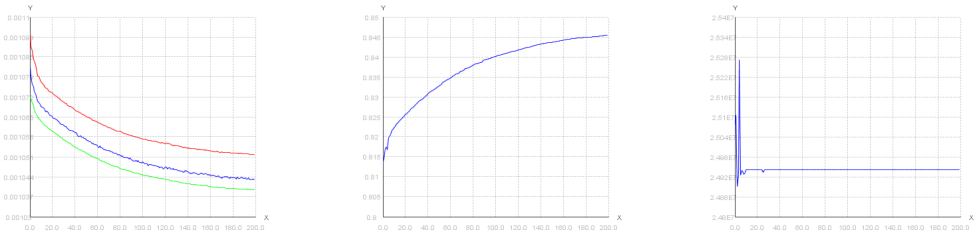


(a) (b) (c)

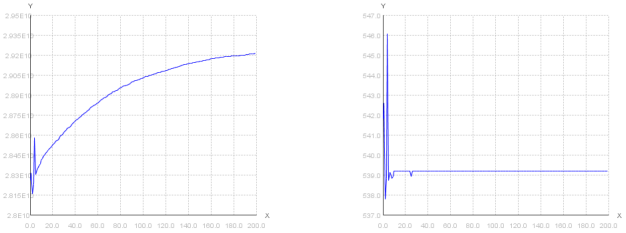


(d) (e)

Figure 5.5: Mutation rates: (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.

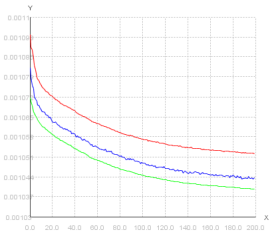


(a) Fitness (b) Efficiency (c) Cost

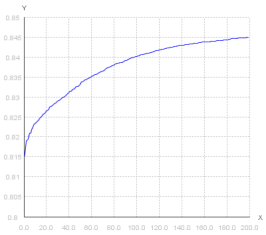


(d) Power (e) Turbies

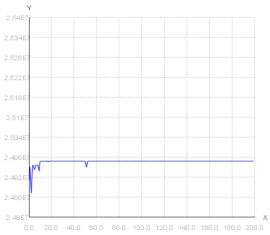
Figure 5.6: Master slave model on scenario 00.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.



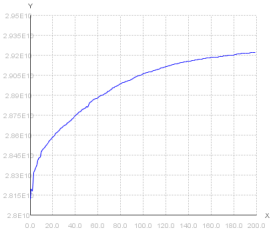
(a) Fitness



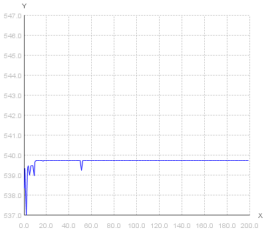
(b) Efficiency



(c) Cost

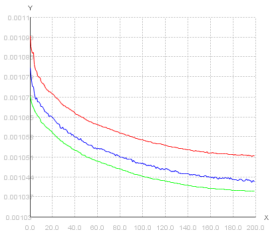


(d) Power

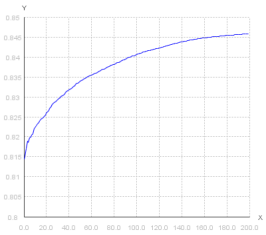


(e) Turbies

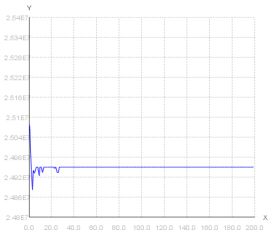
Figure 5.7: Master slave model on scenario 05.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.



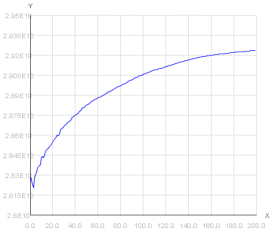
(a) Fitness



(b) Efficiency



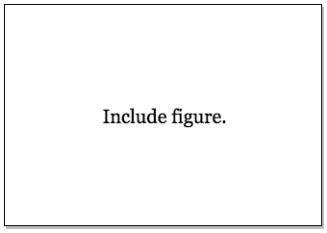
(c) Cost



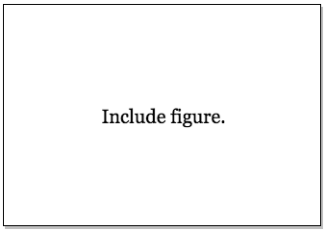
(d) Power

(e) Turbies

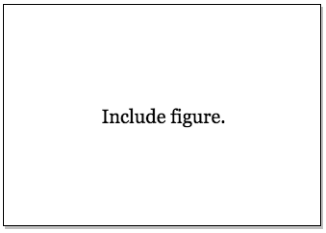
Figure 5.8: Master slave model on scenario obs00.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.



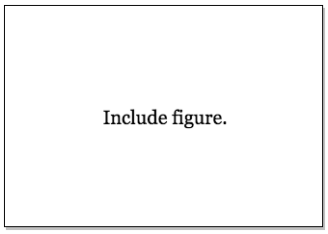

(a) Fitness



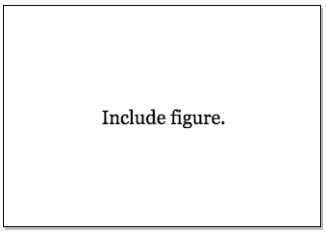

(b) Efficiency




(c) Cost




(d) Power




(e) Turbies

Figure 5.9: Master slave model on scenario obs05.xml averaged over 10 runs: (a) Fitness plot, (b) efficiency plot, (c) cost plot, (d) power plot, and (e) number of turbines.

# 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.

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*.

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.

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*.