Contents

Chapter 1

Introduction

1.1 Introduction

1.1.1 Background and Motivation

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 17.05 percent of the worlds energy sources as of 26th of May 2015 (renewableenergyworld.com).

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 [Samorani, 2013]. For wind energy to become a bigger player in the worlds energy sources, sophisticated methods for wind turbine placement in wind farms needs to be developed so that each turbine produces as much energy as possible.

Wind turbine positioning is hard to optimize analytically. Fortunately, use of genetic algorithms shows promising results. As more advanced approaches to evaluate layouts are developed, and more realistic constrains are introduced, more sophisticated genetic algorithms are required.

To come up with more sophisticated genetic algorithms for solving the wind

farm layout optimization problem, the annual Genetic and Evolutionary Computation Conference (GECCO), launched a competition where different contestants provide their own implementation of a genetic algorithm. 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 (http://www.irit.fr/wind-competition/).

Wind parameters and evaluation mechanism are provided by GECCO, therefore the goal of this project will be to optimize the genetic algorithm for solving the problem not wind farm parameters and models. However, some knowledge of wind turbines, wind farm layout and wake models are useful in understanding the project and is therefore introduced in the background section.

1.1.2 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 to 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 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?*

Chapter 2

Background

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 [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 wind turbine 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 states 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 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 to costly, or it could require costly road work because current roads cannot 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 needs 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. Larger turbines usually generate more power, but they are also more expensive than smaller ones. There is therefore a trade off between the cost and power production. 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. However, 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 also be used briefly in this project, to give an intuitive explanation of the wake effect.

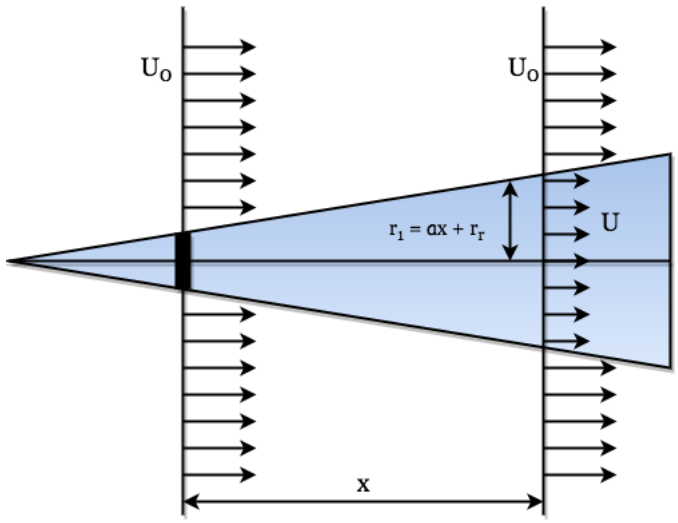

Figure 2.1: The wake effect [Samorani, 2013]

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 α is the entrainment constant constant which decides how fast the wake expands. For a detailed, mathematical explanation of the Jensen model and other wake models see [Jensen, 1983], [Liang et al., 2014].

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

2.2 Genetic Algorithms

The genetic algorithm (GA) was first introduced by John Henry Holland, even though he did not call it the genetic algorithm at the time. This section gives an overview of the workings and most important operations of the genetic algorithm. If not otherwise stated, everything in this section is based on Holland's book "Adaptation in natural and artificial systems" as well as Goldberg's book "Genetic algorithms in search, optimization and machine learning" [Holland, 1975] [Goldberg 1989].

2.2.1 Simple Genetic Algorithms (SGAs)

Genetic algorithms are probabilistic search algorithms inspired by evolution. By performing selection based on survival of the fittest and genetic operations, genetic algorithms are able to solve problems that are too complex to be modeled mathematically. By utilizing evolution as search strategy, genetic algorithms are able to find sophisticated solutions to problems where the structure of the solution is complex, maybe even unknown.

Genetic algorithms starts out by randomly generating a set of solutions to a problem, the set of solutions constitutes the population of the first generation of solutions. Next, each individual in the initial population is evaluated

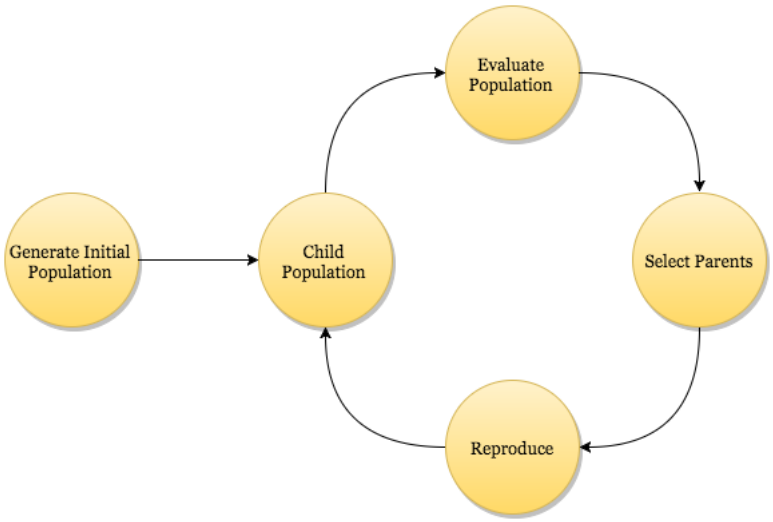

Figure 2.2: Overview of the phases of the genetic algorithm. First an initial population is generated and initialized as the first child population. Second, each individual in the child population is evaluated. Third, based on their evaluation, individuals are selected to become parents for the next generation. Forth, a new generation is generated by reproduction, and fifth, the newly generated population becomes the child population.

based on some predetermined objective function. Based on their objective function values, the fittest individuals are selected for reproduction, meaning that they become the parents of the next generation. By utilizing different genetic operations on the parent solutions a new child population is generated, hopefully with higher average fitness than the previous population since it is based on the fittest individuals from the previous population, and the process repeats itself until the population has found an desirable solution - a population of high-fitness solutions. The different phases of the genetic algorithm is displayed in figure 2.2.

Two key properties are crucial for the genetic algorithm to be useful, (a) there has to be a way to measure the fitness of the individuals, (b) there need to be a way to represent the individuals so that genetic operations can be performed on them. The section below discuss how individuals usually are represented for genetic algorithms.

2.2.1.1 Representation

In genetics, they call an organisms hereditary information for 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 works on genotypes represented as bit strings. Goldberg 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 phenotype value between 0 and 31. Figure 2.3 displays the genotype and phenotype for four randomly generated individuals for two generations. 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.

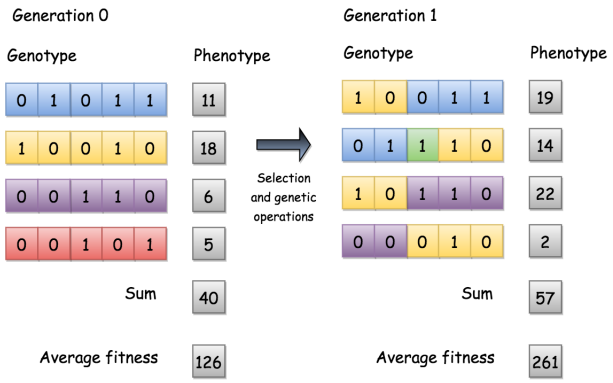

Figure 2.3: Genotypes and phenotypes for four individuals in two generations. On the left side is the randomly generated first generation, and on the right side is the next generation after selection, mutation (red square) and crossover operations. The fitness function used is x^2 and as shown the average fitness of the populations increases from 126 to 261 in only one generation.

The different colors of each individual in figure 2.3 is just there so that it is easy to understand the genetic operations. The operations will be explained in the section below, for now just accept that some fitness-based selection

and genetic operations are performed on the individuals to the left to produce the individuals to the right. As you can see the average phenotype values increases from generation 0 to generation 1, and the average fitness, which is just the average of the squared phenotype values, increases form 126 yo 261 in only one generation.

2.2.1.2 Operations

The sections above explain that the genetic algorithm uses selection of the fittest and genetic operations, but not exactly how this work. This section will explain different selection strategies and the most common genetic operations - mutation and crossover.

2.2.1.2.1 Selection

Selection is the process of selecting which individuals from a given population that will be the parents of the next generation. It can be done in a number of ways, and some of the most popular strategies will be presented here.

The simplest form of selection is called *elitist selection*, meaning that the n best individuals from the population are selected as parents for the next generation. This strategy is extremely simple, but unfortunately not very good because choosing only the best individuals each generation often leads to fast convergence of an non-optimal solution. 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 is cut off too soon. To maintain diversity in the group long enough to find high-fitness solutions *controlled elitist selection* is preferred.

A very popular controlled elitist selection strategy is *tournament selection* [Razali et al., 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.

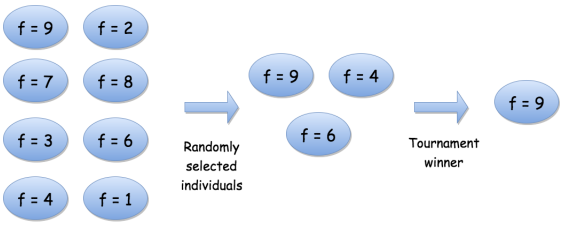

Figure 2.4: Tournament selection. A group of three individuals are randomly drawn from the pool of all individuals. The best individual in the group, the one with fitness 9, is selected for reproduction.[Razali et al., 2011]

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, if n is equal to 1 however 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 makes 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 is prioritized at the end.

Goldberg presents another popular selection process, *roulette wheel selection*. This form of selection uses a roulette wheel where each individual has a slot with size proportional to its fitness. This way, all individuals has a chance of being picked for selection, but the fittest individuals are more likely to be picked. Figure 2.5 presents the roulette wheel for the individuals in figure 2.3. The yellow individual has a fitness of 18 and therefore a probability of $\frac{18}{40} = 0.45$ of being picked, the blue has the probability of $\frac{11}{40} = 0.275$, the purple $\frac{6}{40} = 0.15$ and the red $\frac{5}{40} = 0.125$. Based on these probabilities the yellow individual was picked twice for selection, the blue and purple one time each, while the red one was not picked at all, and therefore not represented in the next generation. This lead to a new generation of individuals with much higher fitness than the previous one, because of inherited properties from good parent solutions. The colors don't make sense in black and white, need to give numbers to each individual in figure 2.3 and use them in the explanation, not colors.

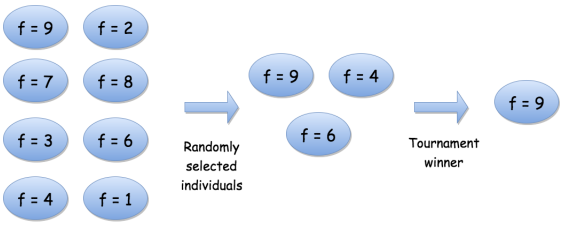

Figure 2.5: Roulette wheel selection. [Goldberg, 1989].

2.2.1.2.2 Mutation

In biology, mutation is defined as a permanent alteration in the DNA sequence that makes up a gene. Mutations vary in size, sometimes just a small base pair of the gene is altered, while other times the mutation can alter large parts of a chromosome. There are many different forms of mutations. Sometimes it can simply mean changing the value of a nucleotide to its complement (nucleotieds are the building blocks of DNA and consists of the values A, C, G and T), other times it can mean insertion of extra nucleotides into the DNA, deleting some of the nucleotides from the DNA, or invertion of nucleotides [Compeau et al., 2014].

In genetic computation the mutation process is simplified. Since the search space only consist of strings of bits with constant size, a mutation in genetic computing simply consists of flipping of bits, this is called single point mutation. In Figure 2.6 we have a bit string of size 8, where the 6th element is mutated from the value 1 to the value 0. Mutation is usually implemented by having a given probability of each value in the genotype being flipped.

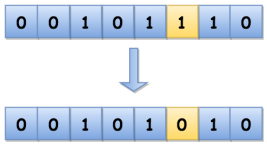

Figure 2.6: Mutation of a single bit. The bit in position 6 at the upper bit string has the value 1 before the mutation, while after mutation the value if flipped into 0.

Mutation is important because without mutation a population can con-

verge 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 for maintaining diversity in the population.

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 reproduction between two individuals with high fitness in the previous population, if mutation heavily changes the new individual, it will not inherit the good features of its parents and the whole point of evolutionary search will be gone.

2.2.1.2.3 Crossover

As in nature, we want individuals of the evolutionary search to inherit features from individuals in the previous population. In genetic algorithms this is performed by applying the crossover operation. Crossover is a simple method where a new individual gets some parts of its genotype from one parent, and the other parts from its other parent. Figure 2.7 illustrates the crossover operation. Before crossover is performed one needs to decide at which position it will be performed. If we are operating with genotypes of length 8 as in figure 2.7 there will be 7 possible crossover positions, one between each of the 8 values of the genotype. In figure 2.7 position 4 is picked as the crossover position, therefore the first of the newly produced individuals will have a genotype consisting of the first 4 values of the first individual of the old population and the 4 last values of the second individual of the old population, while the second newly generated individual will get the opposite genotype.
The crossover in figure 2.7 is called a single point crossover, because crossover is only performed at one position. It is also possible to have multiple crossover positions as shown in figure 2.8, where a double point crossover is performed with crossover positions 2 and 5.
Crossover probability is usually kept high because one is hoping that combining to good solutions from the previous population will produce even better solutions for the next population. However, it is desirable to keep some solu-

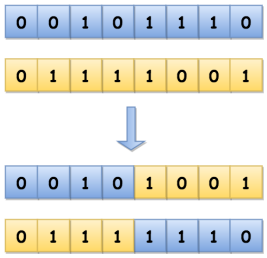

Figure 2.7: Single point crossover at position 4.

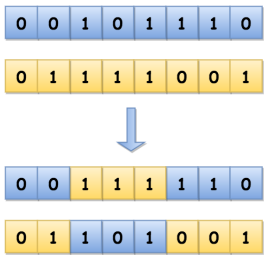

Figure 2.8: Double point crossover at position 2 and 5.

tions from the previous population as they are, because they might be better then combination of two of them.

Goldberg performs a very interesting analysis of the crossover operation by studying the probability of surviving a crossover operation for different schemata. A schema is a string taken from the library {0, 1, *}. A shema of size 8 could therefore for example look like this *0011*****, where *0* and *1* represents letters with values 0 and 1, while * represents letters that we don't care about. This means that both strings *00110000* and *00111111* are covered by the given schema. If desirable features are represented by schemata it is easy to see that some features are much more likely to survive a single point crossover operation than others. Lets study the shemata $s_1 = 11******$ and $s_2 = 1******1$. Both has size 8, meaning that there are 7 possible crossover positions. However, only one of the crossover positions will destroy s_1 while all 7 positions will break up s_2. This means that the genetic search would converge faster if s_1 was the desirable feature, then if s_2 was.

2.2.2 Distributed Genetic Algorithms (DGAs)

On of the main challenges of simple genetic algorithms is keeping diversity in the population long enough so that the population do 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 get the opportunity to find better, more sophisticated solutions. This section introduces two distributed methods that will be explored in this project; the island model, and the cellular model. If not otherwise stated, everything in this section is taken from [Gong, 2014].

2.2.2.1 The Island Model

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

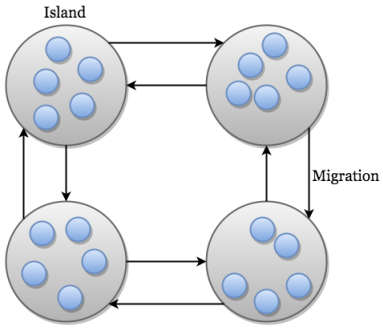

Figure 2.9: An island model using a ring topology with four demes of size five. [Gong, 2014]

According to [Huang, 2007], six parameters needs to be specified when using the island model. First of all, the deme size needs to be specified; the number of individuals on each island (deme). Second, one needs to decide on the number of demes. In figure 2.9 the deme size is five and four demes are used. Third, the topology must be specified; the allowed routes to migrate from one population to another. Numerous topologies can be used. In figure 2.9 the arrows are the 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 of selection emigrants and how to replace existing individuals with new migrants.

The above parameters 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 implement 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, while as asynchronous migration is used migration can be performed whenever one of the parallel processes is ready. Additionally, it has to be decided if the island model is homogeneous or heterogeneous. By a homogeneous island model, Gong 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.

2.2.2.2 The Cellular Model

Figure 2.10 displays the cellular model from [Gong, 2014]. In the cellular model the population is distributed in a grid of cells where each cell holds one individual. Each individual only "sees" the individuals of its neighborhood (as decided by the given neighborhood topology) and can only be compared with and mate with individuals in its neighborhood.

The takeover time is defined as the time it takes for one individual to propagate to the whole population. The neighborhood topology largely affects the takeover time. In figure 2.10 the neighborhood topology is defined as only the individuals left, right, over and under the given individual. Since the topology includes a small number of individuals the takeover time will be very long, meaning that exploration is prioritized. If the topology consist of a larger number of cells the takeover time will, off course, be mush shorter.

The cellular model can also be implemented in parallel, ideally with one

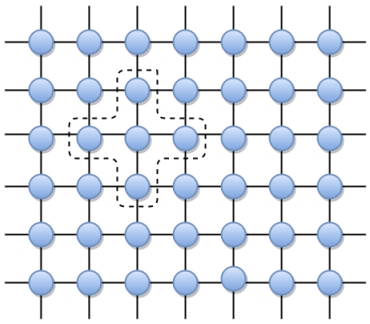

Figure 2.10: Cellular model where the neighborhood topology consist of the cells to the left, right, over and under the given cell. [Gong, 2014]

processor for each cell. As in the island model, updating of the cells can be both synchronous and asynchronous. Synchronous updating means that all the cells are updated at the same time, while as asynchronous updating means that each cell is updated at a certain time following some updating scheme.

Chapter 3

Related Work

3.1 A Survey of the State of the Art

In 1994, Mosetti et al. were the first to successfully demonstrate the utilization of the genetic algorithm in solving the wind farm layout optimization problem [Mosetti et al., 1994]. 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 one have to specify a wake model, a power curve and a cost function. To model the wind decay, Mosetti et al. used a wake model similar to the Jensen model [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 summarized to get the total power produced by the farm in one year as shown in equation 3.1. And last, cost was modeled as a simple function of the number of turbines N_t, assuming a cost reduction when a large number of turbines are installed as shown in equation 3.2.

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

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

With goal of producing a great amount of power at low cost, the objective

function 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 10×10 quadratic grid so that a wind turbine could be installed in the middle of each cell. The optimization problem would then be to find which cells wind turbines should be installed in, in order to 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 an wind turbine is installed in the corresponding cell, and a value of zero means that there is no wind turbine in the corresponding cell. The genetic algorithm used was a simple, single-population genetic search where the fittest individuals where selected for reproduction using crossover and mutation. The crossover operation was performed at random locations with probability $0.6 < P_c < 0.9$ and mutation was performed with probability $0.01 < P_m < 0.1$. Figure 3.1 illustrates how an individual represents a wind park for a wind farm partitioned into nine cells.

The model was tested on a single type of turbines in three different scenarios (a) single wind direction, (b) multiple wind direction with constant intensity, and (c) multiple wind direction and intensity. For each scenario, the results was measured against random configurations of 50 turbines. In scenario one, the efficiency of the random configuration was 0.50, while the efficiency of the optimized solution was 0.95. In the second wind scenario, the efficiency was decreased from 0.35 in the random configuration to 0.88 in the optimized configuration. And, in the last scenario 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.

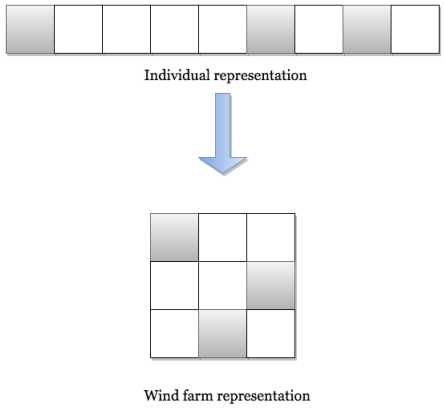

Figure 3.1: A simple example of how an individual of size nine represents a wind farm, where grey cells demonstrates positions containing a wind turbine and white cells demonstrates empty positions.

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

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

As discussed in the paper, different simplifying assumptions are made in the model such as the wake effect model, the cost function, turbine type and layout model. The results are also only compared against random configurations, not configurations optimized by other optimization approaches, and no attempts is 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.

In 2005, Grady et al. picked up where Mosetti et al. left of [Grady et al.,

2005]. They recognized that while the results of Mosetti et al. beat random configurations they were not close to configurations made on simple empirical placement schemes. In their study, they opt to determine the effectiveness of the genetic algorithm compared to optimal configurations to investigate their usefulness in wind farm layout optimization.

As Mosetti et al., they also used a wake model similar to the Jensen model (do I have to reference it again?) as well as the same cost- and power function. However, the objective function was changed into

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

to minimize the cost per unit of produced energy, losing the weight from Mosetti et al presented in equation 3.3.

The same three scenarios as Mosetti et al. was considered. However, the number of individuals was increased from 200 to 600, and run for 3000 generations instead of 400. Grady et al. also implemented a distributed genetic algorithm where the individuals was divided into 20 sub-populations, sadly not sharing any more implementations details. On the first scenario, Grady et al. recognized that with uniform wind distribution the optimal solution could be obtained by optimizing on single row of the layout, and copy it to the rest. As opposed to Mosetti et al., their results are identical to the optimal solution. In scenario (b) and (c) however, the optimal solution can not be obtained empirical, and therefore the results are just compared against those of Mosetti et al. The results for each scenario is displayed in table 3.2.

Table 3.2 compare the solutions of the two studies. The first thing to notice is the difference in number of turbines, where 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 the objective functions. Objective function 3.3 prioritizes low cost and hence prioritizes a lower turbine count. Fact check! The number of turbines for each case explains the largely explains the results. For each scenario the fitness of Mosetti et al. is higher than the fitness obtained in Grady et al. With exception of the first scenario, the efficiency is also larger in Mosetti et al., which makes sense since fewer turbines leads to less wake effect to decrease efficiency Fact check!. However, in each case, the total power production is largely increased in the current study, which also makes sense based on the

turbine count Fact check!.

Table 3.2: Current results compared against the restults 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. is able to show that an optimal solution for scenario (a) can be obtained by genetic algorithm search. It also shows that the power production obtained in Mosetti et al. can be increased by optimizing of parameter values and 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.

Huang presented a study in 2007, showing that a distributed genetic algorithm performs better than a simple genetic algorithm [Huang, 2007]. 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 uses the Island model, with 600 individuals divided among 20 demes, using the ring-topology shown in figure 3.2.

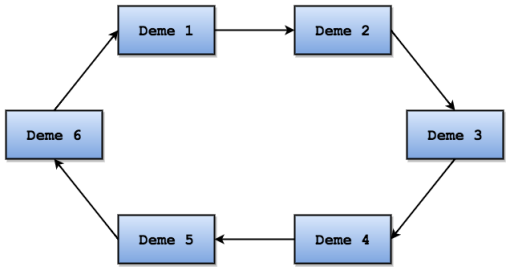

Figure 3.2: Ring topology example with 6 demes [Huang, 2007].

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. and Grady et al., against a simple genetic algorithm. In case (a) the 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 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 resulting in higher efficiency for the distributed algorithm, while in scenario (b) and (c) the distributed algorithm produced solutions with one more turbine than the simple algorithm, resulting in slightly lower efficiency. Huang also briefly gives a thin explanation that the results obtained are slightly better than those of Grady et al., but acknowledges that the results are hard to compare because of different objective functions.

Şişbot et al. published a case study of wind turbine placement on a wind farm at the Island Gökçeada, at the north east of the Aegean Sea [Şişbot et al., 2010]. 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, solutions $\mathbf{y}$ is said to dominate solution $\mathbf{x}$ if

$$\forall i : f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \; and \; \exists j : f_j(\mathbf{x}) < f_j(\mathbf{y}) \tag{3.5}$$

where f_i is objective function [Murata et al., 2001]. Other interesting features of this study is the introductions 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 thought 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 of 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.

Certainly wind farm layout optimization using genetic search needed to be tested in a more realistic environment, and in 2010 Gonzalez et al. successfully attempted to do just that [González et al., 2010]. Three main improvements were presented

1 A more realistic cost model that takes into account wind speed- and direction distributions, turbine count, type, rated power, and tower height, wake effect and other costs such as foundations, road work, building, electrical networks, maintenance and removal costs.

2 A more realistic wake effect model based on [Frandsen, 2006, 2007].

3 An genetic algorithm that handles forbidden areas, soil capacity, roughness length for wind directions, and that are able to set a limit on the number of turbines or initial cost.

The genetic algorithm proposed was different than the previous ones on two points. First, for each new generation the turbine height- and type was optimized to find the maximum net present value (NPV). Second, if the best fitness was unchanged from one generation to another, a local search was performed to improve the fitness of the individuals even more if possible. Results was compared against Grady et al., and the proposed algorithm was also able to find the optimal solution for wind scenario (a). For wind scenario (b), and (c), the proposed solution produced slightly more energy than that of Grady et al., and it needed a lot fewer generations to do so - which one would expect when local search is used.

One of the most complete studies of the wind farm layout optimization problem was done by Kusiak et al. (2009). Should I write et al. when only two authors? 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 are 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.

Another very interesting solution to the wind farm layout optimization problem was proposed by Saavedra-Moreno et al. (2010). 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

height differences. 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 has been discussed and requested by many Reference everyone who has requested seeded GA.. Instead of starting the genetic search from a random population, a greedy-constructive heuristic is used to decide the initial positioning of the turbines. The greedy heuristic works by placing turbines one by one in the position with maximal wind speed. First, 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 is 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 alone, but more importantly, in each case, the seeded genetic algorithm beats the results of the simple genetic algorithm.

In 2015, Gao et al. implemented a distributed genetic algorithm to solve the wind farm layout optimization problem. Unfortunately, they have kept most of the implementation details of the distributed genetic algorithm for themselves. One interesting feature about their layout optimization is that they do not use the constraint that the turbines have to be positioned in the middle of a cell, but can be placed anywhere within the farm. This property is made possible by inspecting the positions of the turbines relative to each other, and reassign the turbines that are too close to other positions, a method introduced by [Wan et al., 2009]. The algorithm is tested on the same three scenarios from Mosetti et al. and compared against all previous studies using the same scenarios [Mosetti et al., 1994], [Grady et al., 2005], [Gonzàlez et al., 2010], [Mittal, 2010], [Pookpunt et al., 2013,][Wan et al.,

2009, 2010], [Zhang et al., 2011]. For the comparison to be valid they force their solutions to have the same number of turbines as the previous studies. In each case their resulting layout is able to produce more energy, and has higher efficiency, while never achieving 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 demonstrates that the distributed genetic algorithm can be applied to a real-world wind farm layout optimization problem. The resulting wind farm layout was able to produce 9.1% of yearly electrical consumption in Hong Kong (2012).

Chapter 4

Technical Description

Since this thesis is part of the Genetic and Evolutionary Computation Conference 2015, the conference has provided all the contestants with an API available on a GitHub. API's for different programming languages are provided, but the Java API will be used in this thesis. The API includes an implementation of a simple genetic algorithm, an evaluation method and ten different wind scenarios. This section will give an explanation of the provided API, display the results from a few test simulations (which will not be included in the final thesis), and discuss future work.

4.1 API

A class diagram of the provided API is displayed in figure 4.1. As can be seen, the diagram consist of six classes. The GA.java class which contains a simple genetic algorithm, i.e. the whole process displayed in figure Want to reference the first figure of the genetic algorithm process. Layout evaluation is taken care of by the evaluation classes. WindFarmLayoutEvaluator.java is an abstract class, implemented by KusiakEvaluator.java and CompetitionEvaluator.java, where the last one connects the evaluation cost function of the competition to the online server of the competition. Next, the class called WindScenario.java is used to initialized one of the ten wind scenarios provided by the competition. Running the program is simple, it only requires initializing the wind scenario, layout evaluator, and the genetic algorithm, and then start the run() method in the GA.java class.

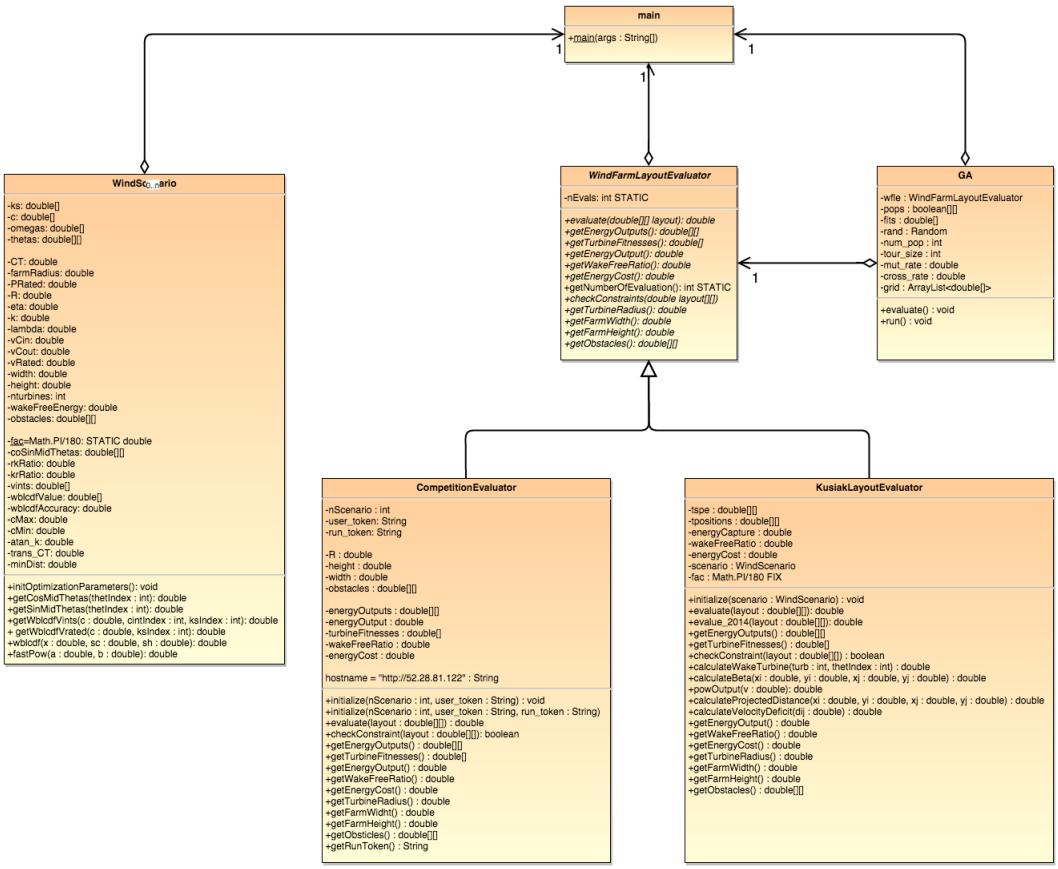

Figure 4.1: Class diagram of the API provided på GECCO 2015.

4.1.1 Genetic Algorithm

A short description of special features of the genetic algorithm.

4.1.2 Evaluation

The main task of the evaluation classes are to calculate the fitness of each individual based on the objective function. The objective function to be optimize is provided by GECCO, and is displayed in equation 4.1.

$$fitness = \frac{\left(c_t \cdot n + c_s \cdot \lfloor \frac{n}{m} \rfloor\right)\left(\frac{2}{3} + \frac{1}{3} \cdot e^{-0.00174n^2}\right) + c_{OM} \cdot n}{\left(\frac{1-(1+r)^{-y}}{r}\right)} \cdot \frac{1}{8760 \cdot P} + \frac{0.1}{n} \quad (4.1)$$

Description and value of all parameters given in equation 4.1 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 of 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 is 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. Not finished with this section. Don't know if wake, power curve and stuff should be here. Explain how power is calculated, since it is needed to calculate fitness. Explain how every term in the objective function is calculated,

then this is done.

4.1.3 Wind Scenarios

As mentioned, ten wind scenarios are provided by GECCO 2015. As an example, the first 10 out of 24 rows of wind scenario 1 is provided in table 4.2.Not done.

Table 4.2: Wind scenario 1 provided by GECCO 2015.

Index	**c**	**k**	*omega*	θ
0	7.0	2.0	0.0002	0
1	5.0	2.0	0.0080	15
2	5.0	2.0	0.0227	30
3	5.0	2.0	0.0242	45
4	5.0	2.0	0.0225	60
5	4.0	2.0	0.0339	75
6	5.0	2.0	0.0423	90
7	6.0	2.0	0.0290	105
8	7.0	2.0	0.0617	120
9	7.0	2.0	0.0813	135

4.1.4 Test Simulation

Change title. Include results here.

4.1.5 Future Work

In order to investigate the goal statement and answer the research questions the provided genetic algorithm needs to be extended as shown in figure 4.2. The provided genetic algorithm will be used as a super class, which will be extended by different distributed genetic algorithms, such as the master slave model, the island model, the cellular model and a few hybrid models as well. For details of the distributed models that will be implemented see section Reference background.

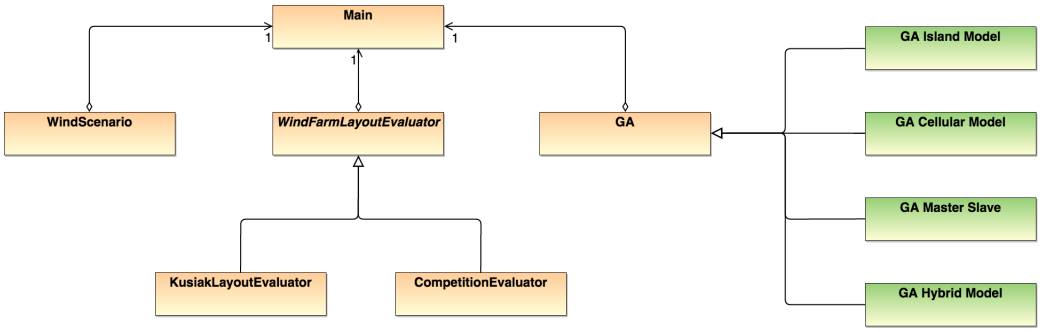

Figure 4.2: Simplified version of class diagram from figure 4.1 extended with different distributed genetic algorithms (green).

In addition to implementing the distributed genetic algorithms listed above, there might be a few changes done to the class GA.java depending on the "goodness" of the results, such as changes to the selection-, mutation and/or crossover methods. Other extensions might also be included if needed, such as seeding the genetic algorithm in order to get as good results as possible for the competition.

After the different distributed models are implemented, extensive simulations will be run for all the models, and results will be presented and discussed thoroughly.

Bibliography

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*.