Chapter 1

Background

1.1 The wind farm layout optimization problem

This section defines the wind farm layout optimization problem and explains the key factors that makes the problem so complex.

1.1.1 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 the 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 in more detail 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."

1.1.2 The wake effect

Different mathematical models, that vary in both complexity and quality, exists to models to the wake effect. However, most research within the field of wind farm layout optimization use the Jensen model [Jensen, 1983]. The Jensen model is preferred both for its proven ability to accurately model the wind speed reduction caused by the wake effect, but also because of its simplicity. The Jensen model is explained below.
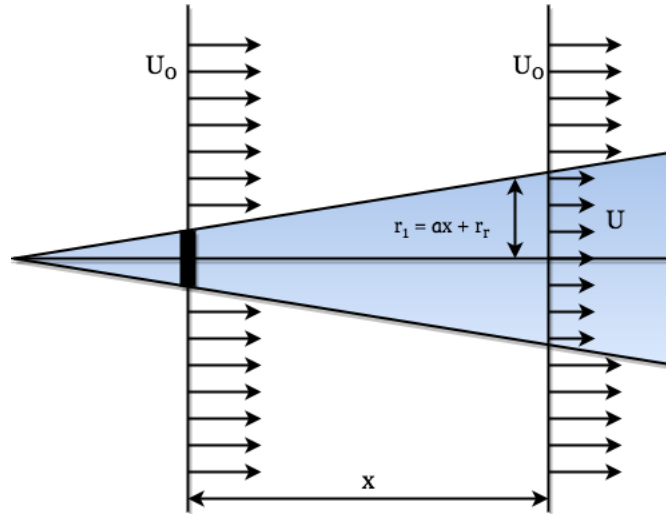
1.1.2.1 The Jensen model

Figure 1.1: The wake effect [Samorani, 2013]

In figure 1.1 the small black rectangle represents a wind turbine, and the blue area behind it illustrates the area that is affected by the wake 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 where r_r is the rotor radius and α is the entrainment constant constant which decides how fast the wake expands

$$\alpha = \frac{0.5}{ln\frac{z}{z_0}} \tag{1.1}$$

where z is the hub height of the wind turbine and z_0 is a surface roughness constant. The wind speed U is therefore Jean help!

$$U = U_0 \left(1 - \frac{2a}{1 + \alpha \left(\frac{x}{r_1} \right)} \right) \tag{1.2}$$

where a is called the axial induction factor, and C_T thrust coefficient.

$$a = 0.5 \left(1 - \sqrt{1 - C_T} \right) \tag{1.3}$$

1.2   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" [J. Holland, 1975] [D. Goldberg 1989].

1.2.1   Simple Genetic Algorithm

1.2.1.1   Representation

1.2.1.2   Operations

1.2.1.2.1   Selection

1.2.1.2.2   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
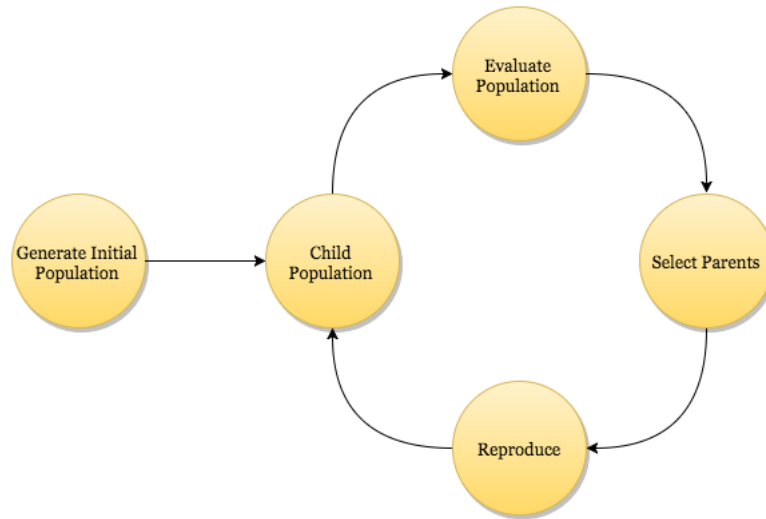
Figure 1.2: Overview of the phases of the genetic algorithm. First an initial population is generated and initialized as the first child population. Second, each individual in the child population is evaluated. Third, based on their evaluation, individuals are selected to become parents for the next generation. Forth, a new generation is generated by reproduction, and fifth, the newly generated population becomes the child population.

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 [P. Compeau, 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 1.3 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.

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
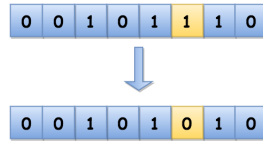
Figure 1.3: Mutation of a single bit. The bit in position 6 at the upper bit string has the value 1 before the mutation, while after mutation the value if flipped into 0.

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.

1.2.1.2.3 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 1.4 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 1.4 there will be 7 possible crossover positions, one between each of the 8 values of the genotype. In figure 1.4 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.
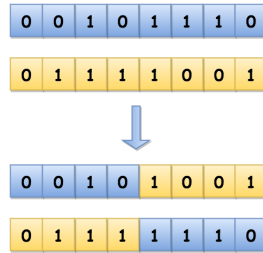
Figure 1.4: Single point crossover at position 4.

The crossover in figure 1.4 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 1.5, where a double point crossover is performed with crossover positions 2 and 5.
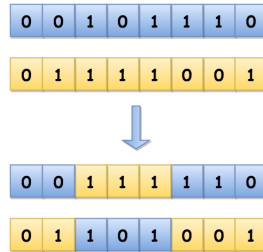

Figure 1.5: Double point crossover at position 2 and 5.

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 solutions 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.

1.2.2 The Island Model

The island model is an extension of the general GA where the initial population of size N_{total} is partitioned into N_{island} such that each island holds a population of size N_{total}/N_{island}. The separated populations can then evolve on each island, exploring different solutions, until a migration takes place. A migration is when some individuals from one island migrate to another island. Figure ?? shows an island model consisting of four island where the arrows shows legal migration routes. The topology of the island network and migration routes can take many different forms.

The advantage of using the island model, as compared to the classical GA is that in the classical GA the population can converge to a non-optimal solution quite fast, because a few individuals can take over the entire population and guide it towards a sub-optimal solution. Using the island model the probability of converging to a sub-optimal solution will decrease because each island can investigate a different part of the solution space.

1.2.3 The Cellular Model

1.3 Wind Farm Layout Optimization using Genetic Algorithms

In 1994, Mosetti et al. successfully utilized the genetic algorithm on the wind farm layout optimization problem [Mosetti et al., 1994]. To model the wind decay Mosetti et al. used a model similar to the Jensen model [Jensen, 1983]. 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
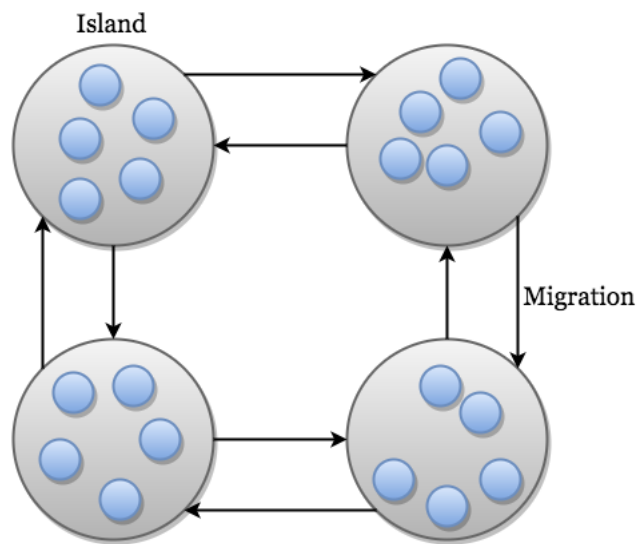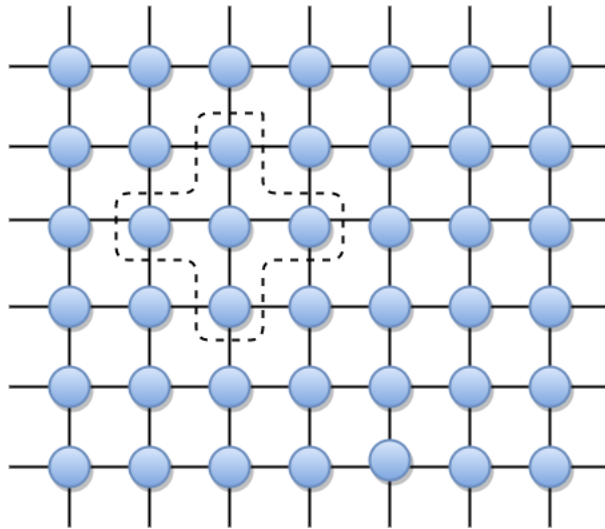
Figure 1.6: Island Model

Figure 1.7: Cellular Model

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 $0.01 < P_m < 0.1$. The fitness of the individuals was determined by the objective function

$$Objective = \frac{1}{P_{total}}w_1 + \frac{cost_{total}}{P_{total}}w_2 \qquad (1.4)$$

where P_{total} is the total energy produced in one year, $cost_{total}$ is a function of the number of wind turbines installed, and 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. The model was tested on a single type of turbines in three different scenarios (a) constant wind direction and intensity, (b) constant wind intensity, but from a 360° variable direction, and (c) a realistic wind scenario. This work laid the foundation for a number of studies of wind farm optimization using genetic algorithms. The genetic algorithms have been optimized, and they now run on much more realistic wind scenarios, cost-, and power models.

Huang improved the results of Mosetti et al. by using a distributed genetic algorithm [Huang, 2007]. Huang stated that using a distributed genetic algorithm will not only decrease the computation time by using parallelism, but also achieve higher fitness because more of the solution space will be explored. The distributed algorithm distributed the population of 600 individuals among 20 demes, using a ring-topology as shown in figure 1.8.

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.

In 2015, Gao et al. used a distributed genetic algorithm to solve the wind farm optimization problem [Gao et al., 2015]. Unfortunately, they have kept most the implementation details of the distributed genetic algorithm to themselves, but they introduce an interesting hypothetical case study of wind turbine placement on an offshore farm located in the Hong Kong southeastern
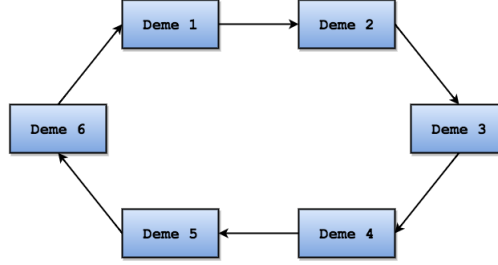
Figure 1.8: Ring topology example with 6 demes [Huang, 2007].

water. By using real wind data, collected over 19 years, they demonstrates that the distributed genetic algorithm can be applied to a real-world wind farm layout optimization problem. It should also be noted that Gao et al. do not use the constraint that the turbines have to be positioned in the middle of a cell. 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 They borrowed this method from Wan et al., 2009.

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

$$\forall i : f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \; and \; \exists j : f_j(\mathbf{x}) < f_j(\mathbf{y}) \tag{1.5}$$

where f_i is objective function [Murata et al., 2001]. Other interesting

10

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].