Chapter 1

Background

1.1 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. [Holland, 1975]

1.1.1 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."*

1.1.2 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.

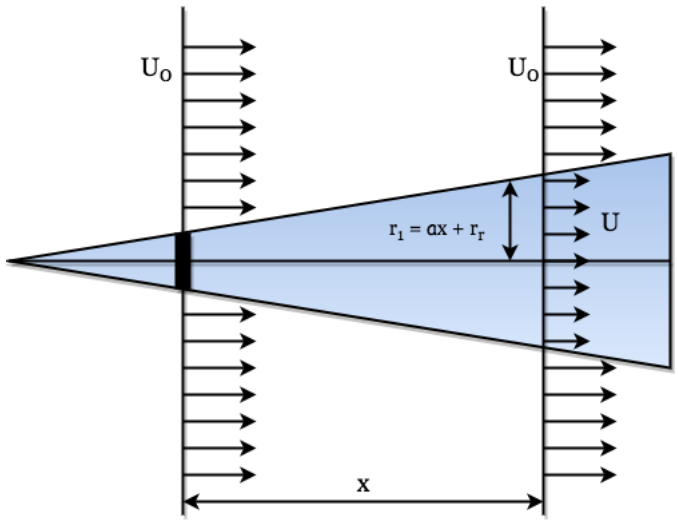

Figure 1.1: The wake effect [Samorani, 2013]

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 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.

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" [Holland, 1975] [Goldberg 1989].

1.2.1 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

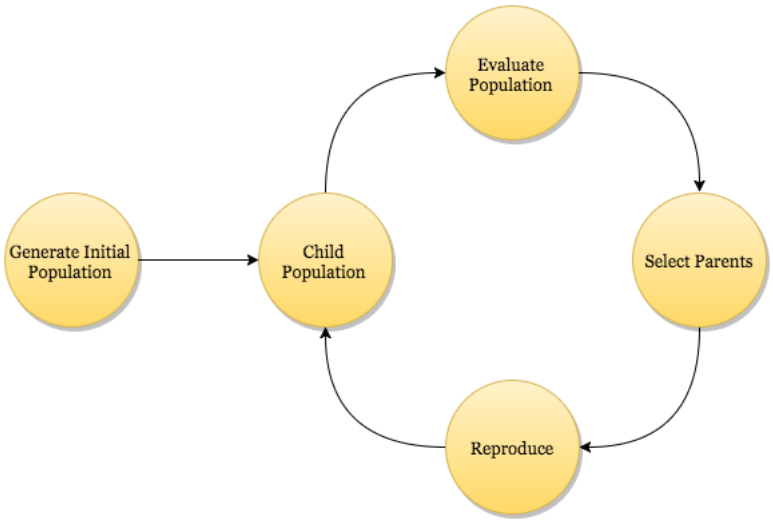

Figure 1.2: Overview of the phases of the genetic algorithm. First an initial population is generated and initialized as the first child population. Second, each individual in the child population is evaluated. Third, based on their evaluation, individuals are selected to become parents for the next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

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 1.2.

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.

1.2.1.1 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 1.3 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.

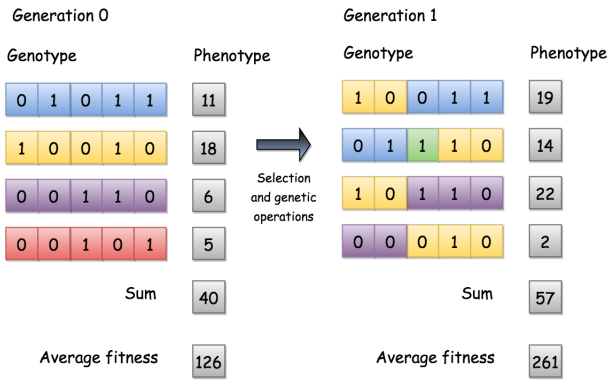

Figure 1.3: Genotypes and phenotypes for four individuals in two generations. On the left side is the randomly generated first generation, and on the right side is the next generation after selection, mutation (red square) and crossover operations. The fitness function used is x^2 and as shown the average fitness of the populations increases from 126 to 261 in only one generation.

The different colors of each individual in figure 1.3 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.

1.2.1.2 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.

1.2.1.2.1 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 1.4 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.

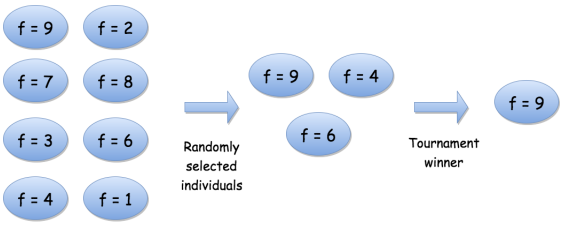

Figure 1.4: Tournament selection. A group of three individuals are randomly drawn from the pool of all individuals. The best individual in the group, the one with fitness 9, is selected for reproduction.[Razali et al., 2011]

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 1.5 presents the roulette wheel for the individuals in figure 1.3. 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 1.3 and use them in the explanation, not colors.

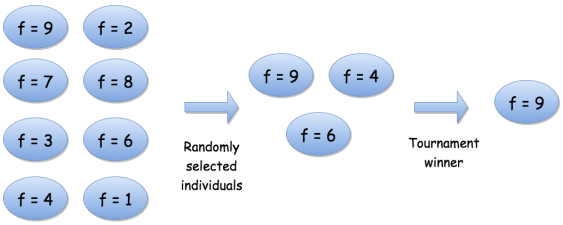

Figure 1.5: Roulette wheel selection. [Goldberg, 1989].

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 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 1.6 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.

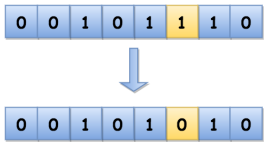

Figure 1.6: Mutation of a single bit. The bit in position 6 at the upper bit string has the value 1 before the mutation, while after mutation the value if flipped into 0.

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.

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 1.7 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 1.7 there will be 7 possible crossover positions, one between each of the 8 values of the genotype. In figure 1.7 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 1.7 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 1.8, 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-

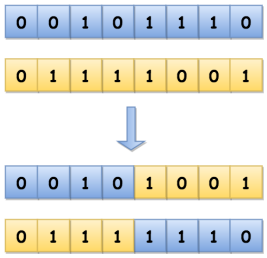

Figure 1.7: Single point crossover at position 4.

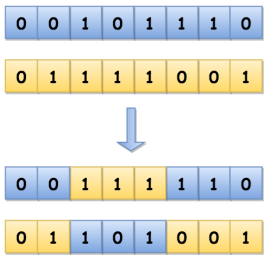

Figure 1.8: Double point crossover at position 2 and 5.

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.

1.2.2 Distributed Genetic Algorithms

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].

1.2.2.1 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 1.9 displays a population divided into four sub populations.

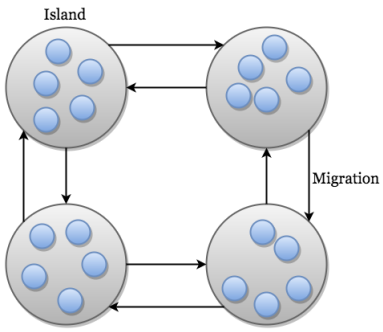

Figure 1.9: Add caption!

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 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

$$Objective = \frac{1}{P_{total}} w_1 + \frac{cost_{total}}{P_{total}} w_2 \tag{1.1}$$

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 1.10.

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.

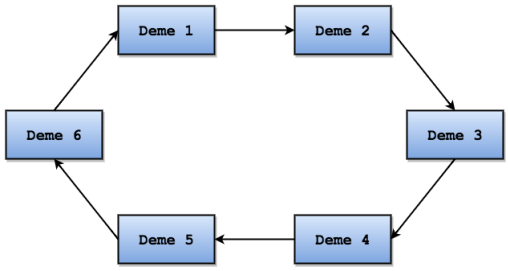

Figure 1.10: Ring topology example with 6 demes [Huang, 2007].

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 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

$$\forall i : f_i(\mathbf{x}) \leq f_i(\mathbf{y}) \; and \; \exists j : f_j(\mathbf{x}) < f_j(\mathbf{y}) \tag{1.2}$$

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].

Bibliography

Holland, J. H. (1975). Adaptation in natural and artificial systems. *Test*.