UNIVERSITY OF EXETER

Final Report

The use of Genetic algorithms for the optimisation of tidal generator arrays

JAMES RICHMOND

2014

ECM3102

3rd Year MEng Individual Project

College of Engineering, Mathematics, and Physical Sciences

University of Exeter

Abstract

Genetic algorithms present a method to solve engineering problems typically considered too complex to calculate optimal solutions arithmetically. Problems that typically fall into this description often include large numbers of variables and non-linear interactions between them. A common approach is the modelling of the relationship between the input variables and a set of output variables and manually changing values looking for an optimal result. A genetic algorithm automates this process by simulating a process comparable to a trial and error approach but using principals of biological evolution to effectively identify regions of optimal solutions faster than a purely brute force method. The engineering problem of designing a tidal generator array to optimise the yield in relation to initial cost is considered. By managing the implementation of the problem within GANetXL (a genetic algorithm solver plugin for Microsoft Excel), a range of optimal solutions could be found along with a series of considerations for future expansion of the model used.

Keywords: genetic algorithm, tidal array, global optimisation

Table of Contents

1 – Introduction

Genetic algorithms present a possible solution to the optimisation of complex engineering problems. They have the capacity to alter a set of input variables for defined problems to an optimum value for the given criteria. The method can be followed computationally and in practise yields better results faster than a computational brute force approach or a human's trial and error approach. The benefit of the use of genetic algorithms in this regard becomes much more evident as problems become more complex. Because of this, genetic algorithms should be considered a key tool in the process of engineering design since they allow the operator to find a solution to a problem that balances several criteria to produce the best solution. In the case of a business application such as tidal generator arrays, this approach would allow for optimum arrangements to be found on an ad hoc basis specific to the site. This report will look at the viability of using genetic algorithms on complex engineering problems and will consider the associated problems will such an approach.

2 – Background

2.1 – Genetic Algorithms

A genetic algorithm is a combination of a structure and a procedure. A set of input variables can be optimised by being fed into a genetic algorithm structure and then manipulated by the algorithm to reach a better solution. The stages of a genetic algorithm and the associated considerations for each stage are shown below.[10]

2.1.1 – Problem Definition

The initial step in the creation of a genetic algorithm is the definition of the problem and specifying the genes that make up the solution. Typically and problem such as this can be broken down into one of two goals; either maximising or minimising a value. Other more complex goals can be broken down into one of these types such as the aim to get close to a specified value is effectively minimising the absolute value of the difference of the solution and target variable. It is also possible to combine several factors into a single goal such as the aim of minimising both the cost and the carbon footprint of a product by summing the two with appropriate weighting. However, in a situation such as this it may be more preferable to perform a multi-objective algorithm.

In terms of the optimisation of a genetic algorithm, by minimising the number of genes in a solution, the computation time is lowered and is therefore more efficient and faster. However, it is important not to attempt to remove necessary genes or combine genes together unnecessarily. If the problem is not fully defined, the solution may not be accurate and the solution will become invalid.

After it has been decided what the genes that make up the solution are they can be defined in practical terms. That is to say how they will be stored in the computer's memory. This requires some knowledge of how data is stored within a computer and some practical knowledge on how these data types can be used to represent more abstract data. The aim throughout this process is to minimise the amount of memory needed to define a genome while maintaining a suitable level of precision. GANetXL offers four types of variables: bounded integer, bounded float, bounded grey integer, bounded grey float. Of these the most common to be used are integer and float. An integer represents a whole number between a minimum and maximum value (inclusive). It is useful for anything which would be counted in whole numbers such as employees or turbines and is also useful for defining abstract types by means of indexing. Float values are useful for defining values that exist on a continuous scale between a minimum and maximum value (inclusive). These are used for variables such as length, speeds and angles. It is worth considering that floats are often more memory intensive than integers so it may be worth considering if they are necessary to the solution while considering the nature of the solution (i.e. while a temperature can exist on a continuous scale, if it can only be kept to the nearest degree in practise an integer scale may prove more efficient). In GANetXL there is an option to set the precision of a float; the default is four but it can go up to 8. Increasing this will increase the memory overhead.

There are some things that can be defined as a variable which do not directly correspond to a number such as a material, a manufacturing method or simply a choice whether or not to do something. In these cases the best solution is to use an integer and then take a given value to correspond to a choice (i.e. the gene says '5' so select the 5th material for the solution). In the case of a simple choice an integer limited between 0 and 1 can represent a binary decision. However, if the nature of the mutation algorithm is not known, alternate solutions such as defining the gene as a float between 0 and 1 and rounding to the nearest integer to feed into the solution may provide better results.

If this is approached from a programmatic standpoint, there is more control over the types of variables used and therefore much more scope for optimisation of a genomes definition. However, some caution is needed when following common programming practise. It might be considered good practise to combine variables together into a byte stream or similar structure. Using a method such as this requires careful consideration of the operation of the mutation and breeding functions. For example, if a variable is treated as a number and mutated to another close value, it is easy to keep track of. If the value is defined as a set of 1s and 0s then mutating an individual bit could change the value by just 1 or by 256 or even more with the same probability and mutation is harder to keep track of. In a situation like this, some programmers opt to use grey code to limit the effect of any single mutation on a bit level.

Take the example of the design of a shelf. The problem can be defined as having four variables: width, depth, thickness and material. The first three variables can be expressed as floating point numbers while material can be expressed as a set of bits corresponding to a material from an index (a variable such a byte or a short might be used in a programmatic approach but it depends on the number of materials in the index). The aim of the algorithm would be to minimise the cost while providing an adequate storage area and safety factor. If say the width and the depth were combined into a single variable 'storage area', the solution may be easier to define but there is ambiguity in the problem. Without a dimension defined the shelf could be either long or short while maintaining the same area. In this case the best approach may be to fix the 'storage area' in the problem definition and cut down to the variables: width, thickness and material. The depth can then be worked out from the area and the width which lowers the number of genes while keeping the problem fully defined. While this is a simple solution and the problem is easy to spot, as problems get more complex it is best to keep to the principal of defining all the variables conceivable. After that point, it is possible for genes to be eliminated after careful consideration of the effects.

2.1.1.1 – Multiple Objective Solvers

A multiple objective solver is one that aims to minimise several goal variables of a problem to find the best result. The results of the genetic algorithm will be displayed as a surface on an N dimensional plane which in turn can be converted into an N-1 dimensional boundary. After this, it is up to an operator or some separate solver to decide on which of the solutions is preferable. Later on in this report the design of a pump system is considered as a multiple objective problem. With a number of designs possible an expensive system with a low running cost can be built or a cheap system which is expensive to run can be built. On top of this, there are a number of optimal solutions in between these extremes. Based on other factors such as budget or income the then chosen solution can be found. Naturally, this will be discussed in more depth in its section later in this report.

2.1.2 – Initial population

The creation of an initial population if the first important step in the operation of a genetic algorithm although it is a relatively simple one. The aim of this step is to provide a basis of solutions for the algorithm to build upon. Consider a problem defined with 4 variables, it is unlikely that a solution exists with any of these values at their minimum or maximum values exists but if it does it is somewhat preferable to find out at the start. There are 16 permutations of four variables either at their minimum or maximum value. By placing these in at the start it can be quickly determined if the boundaries are incorrectly placed; although this may not work every time it can prove useful. With a population size of 100, this leaves 84 unaccounted for genomes. GANetXL opts to set all genes to their minimum value but it may be better practise to simple set them as a random variable in the range. When tracking the algorithms progress, time may be wasted at the start of the calculation since it relies on mutation alone to move from the extreme values of the range to the more ideal values closer to the middle of the range. By setting values randomly it makes this initial progress faster although in a long run would have a negligible effect on the overall result. In practise this is entirely up to the preference of the operator.

2.1.3 – Selection

Selection is the name given to the process of evaluating the solutions in a given generation to get a numerical representation of the quality of a solution. This is done by taking the associated genes of a solution and running them through a fitness function with some relation to the mathematical theory the problem is based on. Taking the example of the design of a shelf, the genes of width, depth, thickness and material could be taken to get a coefficient for cost, weight, and deflection under load or other similar measures. Each one of these could act as a fitness function and the chosen one would depend on the approach to the problem. In a multiple objective solver, each one of these could be assigned as a target variable.

Another possible approach is the combination of several target variables into a single target variable. This would reduce the complexity of the problem and in the event that a multiple objective problem can be reduced to a single objective problem, the algorithm will become more efficient and easier to analyse. An example of this might occur in the design and selection of a piece of large production machinery such as an industrial scale printer. The design can be ranked based on initial cost and running cost which would require a multiple objective solver. However, if it is assumed to have a lifespan of 10 years before the system would be replaced a single variable of lifetime cost can be found which reduces the problem to a single objective.

It is worth considering the scale on which solutions are being ranked. The scale should be sufficiently large so it is not commonplace for multiple solutions to receive the same fitness ranking. Some care is necessary when manipulating the fitness function, especially when values are expected to be around 1. Table 1, shown below shows some of the problems and reasoning behind the manipulation of the fitness function.

Alteration	Reasoning	Drawback
Multiplying outcome by constant greater than 1	By multiplying by a number greater than 1, large numbers are affected more than small ones. This makes the algorithm focus on dealing with solutions with large fitness values.	Solutions with small fitness values are ignored to a degree. Has no effect on multiple solutions with the same fitness value.
Multiplying outcome by constant less than 1	By multiplying by a number less than 1, large numbers are reduced to a scale comparable to smaller numbers. This makes the algorithm place less focus on dealing with solutions with large fitness values.	Solutions with large fitness values are ignored to a degree. Has no effect on multiple solutions with the same fitness value.
Raising outcome to a power	Places intense focus on solutions with fitness values at the end of the scale depending on the value of the exponent.	Never use if fitness values cross between <1 and >1 since values will diverge from that point and the data will become useless. In general is it unadvised to raise the solution to a power.

Table 1: Summary of effects for altering the fitness function

2.1.4 – Breeding

Breeding is a key stage to the genetic algorithm process, a good understanding of how the breeding function works is critical to the operation of a genetic algorithm. In essence a breeding function will select two ‘parents’ from an existing population based of their ranking as determined by the fitness function. These parents will then be spliced together to create two ‘children’ which are placed into the new generation. The operating principal behind this is that each of the parents is achieving a high fitness score due to several genes in its genome working well in partnership. By combining these parents it is possible a child will inherit the harmonising variables from one parent and will gain some gene from the other parent which adds to the effect of the harmonising genes making the child more fit than the parent. The drawback of this is this is an entirely blind process since the algorithm has no way to determine which genes are harmonising or how it might improve other genes. By observing the exact genes being transferred and the change in fitness it might be possible for the algorithm to begin to determine how the genes affect each other but that is beyond the scope of this report.

There are a number of different methods of implementing a breeding function and each has their associated benefits and drawbacks. In most cases it comes down the problem in question top determine the most effective breeding function and should be judged on a case by case basis.

2.1.4.1 – Generational Breeding

Generational breeding is the most simple and straightforward breeding algorithm. In a generation of size N, the population is broken down into a set of N/2 pairs which are selected based on their fitness function. The fittest solution is paired with the second fittest; the third is paired with the fourth and so forth until all are accounted for. Assuming each solution is made up of K genes, for each pair a number K_i (where $K_i \leq K$) is randomly generated. A child is created where the first K_i genes are from its first parent and the rest of its genes are from its second parent. A second mirror child is created using the same crossover value but selecting the genes from opposite parents. All the children are then placed into a new set and are henceforth the next generation of the algorithm. From this stage they may be fed into a mutation function or sent straight for evaluation. This method of breeding is the most basic form of generation breeding and would more accurately be called a ‘single-point crossover generation breeding function’.

2.1.4.2 – Multiple-Point Crossover

The previous section describes a single point of crossover where all genes before the crossover point come from parent A and all after come from parent B. While this is a simple concept to grasp it is not very useful in practical terms. Since the computer or the operator has no way of knowing how the genes interact as the solution is optimised it is entirely possible that the first and last gene in the genome have a close connection. In a case where a certain solution has values for the first and last gene which cause it to be ranked highly by the fitness function it is in the best interest of the genetic algorithm and its operator to allow these genes to remain present as a pair in one of the solution’s offspring. The multiple-point crossover algorithm creates a set of crossover points referring to points in a solutions genome. A child will take all the genes up to the first crossover point from its first parent, all genes between the first and second crossover point from its second parent and will alternate back and forth until all genes are accounted for. This is a much more realistic breeding algorithm and in practise gives better results.

2.1.4.3 – Per-Gene Generational

The per-gene algorithm is simply an implementation of the multiple-point crossover. For each gene in a child a random number is generated which will refer which parent to take that gene from (i.e. if the value is even take from parent A and if it is odd take from parent B). This method can become really useful if the algorithm want to use more than two parents since the number generated for each gene could refer to which solution from the previous generation to take the gene. This presents a simple way for creating a new generation from the best solutions from a previous generation. After the solutions have been ranked, separate off the top 10 solutions. For each gene in each child of the new generation generate a number 1 to 10 to determine which of the top 10 solutions from the past generation to take the gene from. This creates a whole new generation made up of the best values from the last. A method like this is more likely to produce quick results but is liable to overlook an optimum solution since the solutions do not diverge much.

2.1.4.4 – Weighted Generational

A weighted generational breeding function operates like one of the previously described breeding functions but allows fitter functions to parent more children. Depending on how fit a solution is, it may be allowed to breed with more solutions so that there are more offspring in the new generation with favourable genes. This is slightly different from ‘top 10’ concept explained in the previous paragraph as it does not eliminate solutions from the breeding pool but instead makes it less likely for them to breed. This method represents a small alteration to an existing algorithm that may yield slightly better results but it does not create a large change to the operation of the algorithm.

2.1.4.5 – Generational Elitist

An elitist algorithm is a useful choice when dealing with large generations or large sample regions. In essence the best solutions of a given generation are carried over without alteration so they may breed with the next generation. This can be a very good thing if it is likely that breeding or mutation would dramatically alter the solution to be significantly worse. It also makes it much more likely for a small change to occur which improves the result. The drawback of a method such as this is that it removes places for bred offspring in the new generation. A typical figure for a generational elitist model is 5% carryover which in small samples can represent significant numbers of lost breeding opportunities. Despite this, by cutting down on the number of breeding operations, the algorithm can run much faster and this can serve as a method to speed up a calculation on a slower computer without affecting the validity of the result. It is worth noting that although these solutions may be carried over to the next generation untouched, they are not precluded from breeding within their generation to pass on favourable genes.

2.1.5 – Mutation

Mutation is a cornerstone of the evolutionary process both in reality and in the operation of genetic algorithms. The operating principal is that a given solution is close to the ideal solution and by changing one of the genes by a small amount; the solution will become closer to the ideal result. Since the algorithm is blind to the ideal solution and the nature of the genes can vary, the process of mutation becomes more complicated. A couple approaches to mutation are discussed below.

2.1.5.1 – Mutation Chance

Mutation is essentially a random operation and should be controlled as such. If the mutation rate is too high it is possible to lost good solutions to random noise. If the rate is too low the effect of mutation will be negligible and the progress of the algorithm will stagnate. The rate represents the chance that any given solution after breeding will mutate in some regard. Past this it is up to the implementation to decide which genes will mutate and how they will mutate.

2.1.5.2 – Mutation by Gene

Mutation by gene is a small alteration of the method described above where instead of defining the chance that a genome will mutate, the chance a given gene in a genome will mutate is defined. This is slightly more computationally intensive but would provide a more realistic mutation scenario. This also removes arbitrary limits on how many genes in a genome can mutate by allowing all the same chance. Also, by creating a per-gene system, the nature of the mutation can be controlled on a gene by gene basis allowing mutations to better suit the type of gene. This is discussed more in depth in the 'mutation by type and gray code' section.

2.1.5.3 – Controlled Mutation

There are some cases where simply changing the value of gene by some number will produce an invalid genome. One area genetic algorithms are used is in the creation of nodal systems and networks. If a path is modelled by a set of genes representing the order of nodes to be traversed, a mutation of a gene could mean the same node is listed twice or a node is removed from the list. In a case like this, the nature of a mutation must be carefully considered to be non-destructive to the integrity of the solution. One possible solution to this example would be using the mutation chance to decide which nodes would swap places within the genome so the order is altered without removing nodes from the system.

2.1.5.4 – Mutation by Type and Gray Code

Not all types of genes can be mutated the same way. Typically the aim of a mutation process is to change a value by a relatively small amount to give a value close to the original without losing its general value. This requires some consideration when deciding how a gene will be mutated since mutations of fixed rates can affect different variables in significantly different ways. For example, if a mutation function changes a number to a random number within a range of 100, which will have a negligible effect if the gene in question has a number in the millions. However, if the gene is only expected to reach up to a few thousand, a mutation of 100 could be significant. One approach may be to base mutation magnitude based of the range the gene operates in to keep the mutation to a reasonable range. Since the values of the specific genes are stored as binary throughout the calculation, one approach to mutation is to randomly change a bit from a 1 to a 0 or vice versa. In theory this mimics real world evolution but due to the nature of binary this mutation could change the value by just 1 or 1024 with equal chance. Because of this, this method is often used in coordination with gray code which is a binary format in which any random change of a bit will only ever change the associated value by 1 making it very suitable for use with a genetic algorithm.

2.1.5.5 – Elitist Mutation

If the breeding algorithm uses an elitist Passover, it should be considered whether or not to allow mutation of these solutions. Since they represent the best solutions that are known an argument could be made to keep them safe from mutation so the accuracy of the algorithm cannot degrade. However, since they may only be a small value away from an ideal solution, mutation may be the best method of improving the solution towards the ideal result. This choice is up to the operator and the limitations of the algorithm being used.

2.1.6 – Constraints

A constraint upon a genetic algorithm is a rule defining a range in which a resultant variable can exist. Since the nature of input variables can be known before calculation they can easily be kept within a range defined by the operator. However, the nature of an output variable cannot be known until after calculation. In the case that an output variable is not reasonable it can be detected by a constraint and removed from the gene pool. Examples of this may be extraneous cost, unrealistic size or infeasible design. A consideration should be given to what should be done with designs that fail constraints as to whether they should be allowed to breed with other solutions or not. These solutions may represent the boundary between optimum feasible solutions and optimum infeasible solutions and simple removing them would be counterproductive. Managing a system where solutions failing constraints may be allowed to create solutions on a boundary of a constraint may be beneficial to the finding of an ideal solution.

2.1.7 – End Point Definition

Due to the nature of genetic algorithms the process of evolution is in theory infinite since there is no way to determine when the ideal solution has been reached or even how close to an ideal solution the best current result is. It is necessary to enforce some end point upon the algorithm for practical use within industry. Typically this end point is defined as a number of generations to evolve through before the best solution is extracted. This has the drawback of taking a significantly long amount of time every calculation and may provide a wide variety of results in terms of accuracy. Another approach is to extract the best solution after a certain tolerance has been met which cuts down on the computation time at the cost of finding a more ideal solution. In large scale industrial use other approaches such as allocating a calculation time or dedicating a computer to persistently run a genetic algorithm and extract the best known solution during working are approaches that may yield the best results in terms of cost and benefit.

2.1.8 – General Terminology

Some technical terms will be used throughout this report and are summarised in this section.

Operator – The person(s) who control the running of a genetic algorithm for a specific problem. They are assumed to have the ability to configure the algorithms settings, change the model calculating the fitness function and alter the format of the input data.

Solution – The exact values held by a genome within any given generation that correspond to an approach to the problem. It is important to note that the term *solution* does not imply an optimum solution or a viable solution but simple one given approach to the problem.

Gene / Input Variable – These terms are synonymous in that any input variable is stored within a gene for use with a genetic algorithm. A gene will have a minimum value, a maximum value, a format (e.g. integer, floating point number, etc.) and an identifier (name).

Genome – A genome is the combination of all the genes into a single object. If a genome has a set of values for its genes it can then be refer to as a solution. In this report the term genome is used to refer to as a generalisation of all solutions or in reference to a solution with no values.

Population – The population is the assembly of all the solutions currently in consideration by the algorithm. At the beginning of the algorithms operation a set of empty solutions are created making the initial population. This set then goes through the stages of a genetic algorithm as described in above sections.

Generation – This refers to the number of times the population has gone through the processes of evaluation, breeding and mutation to reach a new population. The first generation is a population made up of default values and the final generation represents the last population generated which is assumed to hold the best solution.

Pareto front –This is the manner in which results for a multiple objective algorithm is presented. The final generation contains a set of genomes all with varying values for each objective's fitness. When the fitness values are graphed against one another the result is the *Pareto front* on which each solution on it is optimal if one objective is chosen to be fixed at that given value.

2.2 – Fourier Synthesis Background

Fourier synthesis is a mathematical theory stating that any periodic signal can be modelled as the sum of a set of sinusoids of given magnitudes, frequencies and phases. This is a useful theory in terms of engineering design since it has the potential to reduce arbitrarily complex systems to a representative linear system by breaking down a given signal into a series of defined sinusoids. Some examples of this might include the modelling of a heartbeat or an earthquake or the velocity of ocean waves. As can be imagined, being able to represent these events in a mathematical model allow them to be considered directly in an engineering problem. The process of synthesizing a Fourier series should be considered a standard task when dealing with the engineering of complex systems.[11]

2.2.1 – Target Definition

I will be testing the algorithms ability to calculate the set of sinusoids that best fit a set of three simple periodic signals; namely the square wave, saw wave and triangular wave. The mathematical definition and the expected sets of sinusoids for each one are shown below. These functions will be constructed within Microsoft Excel for use with GANetXL; the concerns of this implementation are discussed in a later section.

2.2.1.1 – Square Wave

The square wave alternates between values of +1 and -1 for a fixed duration of time and has a binary transition. A mathematical expression of the waveform is shown below in equation 1.

Equation 1: $$f(x) = (-1)^{floor\left(\frac{x}{L}\right)}$$

To best match the characteristics of a standard sine wave, L will be set as π. This produces the waveform shown in figure 1.

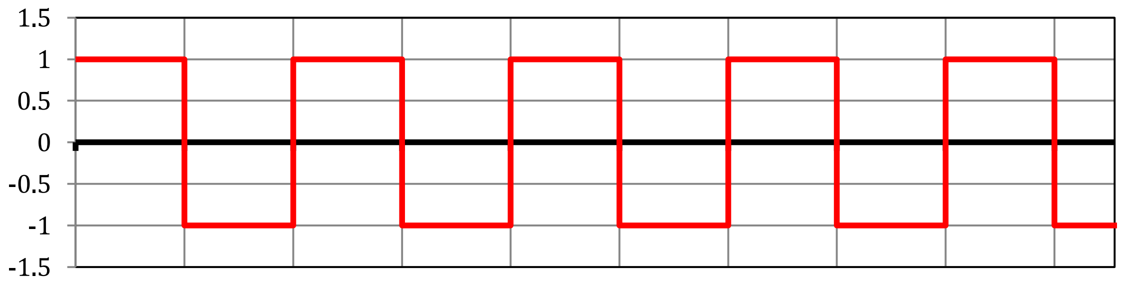

Figure 1: Plot of a standardised square wave

The set of sinusoids that best represent this waveform are given by the equation 2 below in a generalised form.

Equation 2: $$f(x) = \frac{4}{\pi}\sum_{n=1,3,5\ldots}^{\infty}\frac{1}{n}Sin\left(\frac{n\pi x}{L}\right)$$

Equation 3: $$S_i = \frac{4}{\pi} \times A \times Sin(F \times x)$$

In order to simplify the form of the solution, the algorithm will develop sinusoids in the form shown in equation 3. The amplitude A should be equal to the inverse of the frequency F where F is odd.

2.2.1.2 – Sawtooth Wave

The sawtooth wave is a wave that increases in amplitude at a constant rate but drops in a binary transition periodically to maintain itself within a constant range of amplitude. The equation used to generate the sawtooth wave is shown below in equation 4.

Equation 4: $$f(x) = \frac{2}{L}x + \left(2 \times -floor\left(\frac{x}{L}\right)\right) - 1$$

To match the wavelength of a standard sine wave, L is set to π. This produces the waveform shown in figure 2.

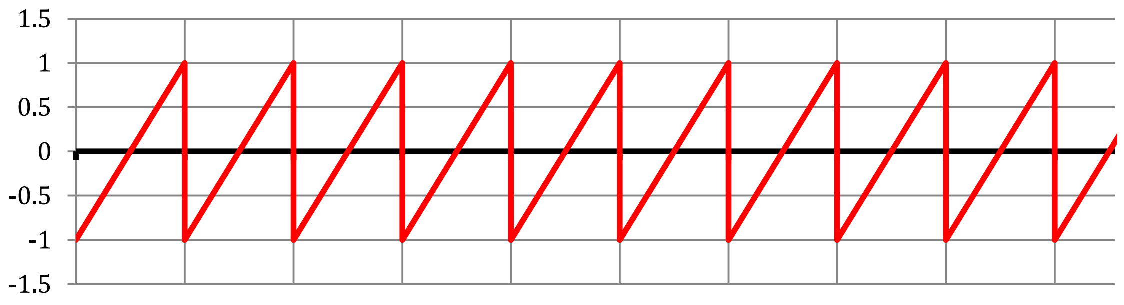

Figure 2: plot of a standardised sawtooth wave

The set of sinusoids waveforms that best represent this waveform are shown in equation 5 below.

Equation 5: $$f(x) = -\frac{2}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}Sin\left(\frac{n\pi x}{L}\right)$$

Equation 6: $$S_i = -\frac{2}{\pi} \times A \times Sin(Fx)$$

The negative sign takes account of the fact that the sine wave should be reversed to best represent the waveform. By keeping it in the form of the solution the amplitude can be kept positive, cutting the number of possible values in half making the algorithm more effective. The form of the solution that will be sought is shown in equation 6. In resultant set of sinusoids, the amplitude should be the inverse of the frequency, starting at one and increasing through integers.

2.2.1.3 – Triangular Wave

The triangular wave swaps between a positive and negative gradient of a fixed rate to maintain it within a fixed amplitude range. The mathematical expression of the waveform is shown in equation 7.

Equation 7: $$f(x) = \left((-1)^{floor\left(\frac{x+L}{2L}\right)}\right) \times \left(\frac{x}{L} - \left(2 \times floor\left(\frac{x+L}{2L}\right)\right)\right)$$

To best match the form of the input sinusoids for the purpose of use in a genetic algorithm, L is set to be equal to π/2 and forms the waveforms shown in figure 3.

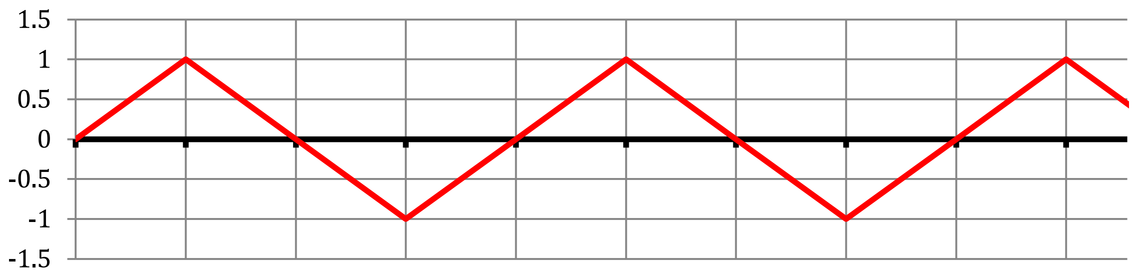

Figure 3: plot of a standardised triangular wave

Using this waveform definition, the set of sinusoids that best fit the solution are given by the formula in equation 8 below.

Equation 8: $$f(x) = \frac{8}{\pi^2} \sum_{n=1,3,5\ldots}^{\infty} \frac{(-1)^{\frac{n-1}{2}}}{n^2} Sin\left(\frac{n\pi x}{2L}\right)$$

Equation 9: $$S_i = \frac{8}{\pi^2} \times A \times Sin(Fx)$$

By substituting in L and taking out constants, the form of the solution to be sought is shown to be that in equation 9. The solution should consist of sinusoids where the frequency increases upwards through odd numbers and the corresponding frequency is equal to the inverse of the number squared, alternating between positive and negative. Because of this, determining the accuracy of the solution for this case may become difficult.

2.3 – GANetXL Genetic Algorithm Solver

There are a number of possible ways to implement a problem into a genetic algorithm but for the purpose of this project the GANetXL excel plugin will be used. GANetXL is a plugin developed by a team at the University of Exeter and was initially presented as a technical paper[1]. The primary benefit of using this GA solver is the inherent integration with Microsoft Excel, an industry standard spreadsheet management program. An overview of the process required to set up and operate GANetXL is described below.

2.3.1 – Configuration settings

GANetXL has a number of setting relating to the general operation of a given run of the algorithm. These include the size of the population, whether the problem is single or multiple objectives and the specification of the various (in-built) algorithms for the various stages of the genetic algorithms operations (i.e. breeding, mutation, etc.) along with their relevant options. These settings also allow for controlling factors such as number of runs, float precision and other data based controls.

2.3.2 – Microsoft Excel Linking

There are a number of inputs that must be connected to GANetXL in order for it to function. In all these cases, they must be represented by a single array of cells in Microsoft Excel with no empty cells or invalid values.

2.3.2.1 – Genes

Genes are specified to GANetXL as an array of cells which will be changed by the solver in order to calculate a fitness for a given genome. In practical terms, none of these cells can have formulas in since they will be overwritten by the algorithm. If a gene cell has a formula it should be broken down into a mutable value as the gene and the associated formula should be placed elsewhere as an intermediate value. Within GANetXL's interface, the minimum and maximum value for each gene can be specified along with the variable type (e.g. integer, floating point, etc.). The number of genes available is dependent on the license of GANetXL being used.

2.3.2.2 – Objectives

The objectives are specified to GANetXL as an array of cells, each representing a certain fitness of the design that the initial genes refer to. These cells must be influenced in some way by all the input genes in some way. By allowing a gene to mutate with no effect on the objectives it will end up presenting a set of values with no pattern in often unrealistic ranges. It is possible this can happen anyway but would only be caused by a poor understanding of the problem and the model being used in a mathematical sense. For each of the objectives the choice of whether to minimise or maximise the value is presented by GANetXL. If a single objective solver is chosen, GANetXL will only allow a single objective whereas a multiple objective solver only limits the number of objectives based on the license of GANetXL being used.

2.3.2.3 – Constraints

Constraints are introduced to GANetXL by the means of a single cell. The option of using constraints at all is also offered allowing any constraints system to be easily toggled on and off between runs. Since it is likely for a problem to have more than one constraint, the constraint cell specified to GANetXL should be the sum of the penalties generated by the various constraint formulas being used. GANetXL describes a constraint on a single objective solver as a penalty cost suggesting the genome in question can be made less favourable without elimination. The constraint on a multiple objective solver is described as an infeasibility case suggesting any constraint will exclude the genome in question. In practise, the exact nature of constraints is uncertain since results with constraints have been offered in the final generation of various problems and as such requires some experimentation to use correctly.

2.3.2.4 – Feedback

GANetXL has the capacity to transfer information pertaining to the running of the algorithm back into the spreadsheet during its operation. Values such as the generation number and the run that the algorithm is operating can be used numerically in this way. This allows for fitness functions and constraints to be changed in relation to how long the algorithm has been running. Solutions can be forced to converge faster towards optimal solutions using heavier constraints earlier which can then be removed allowing free mutation along a Pareto front or around an optimal point. By utilising the run number feedback the algorithm can be categorically changed between runs allowing multiple solutions to be generated from a single prompt. These tools are useful once a model has been fully developed to make the algorithm more practical to work with.

2.4 – Previous work on tidal generator arrays

The tidal generator array industry has a lot of work and research behind it. Much of the theoretical work originates from study of fluid dynamics and as such has benefitted greatly from the advent of computational fluid dynamics (CFD) and advancements in the field of computer simulation. The design of a tidal turbine is comparable to that of a wind turbine as well and the two industries have many similarities. The wind turbine industry is older than the tidal generator industry and the wealth of data from it has fed into the current research. This report is focussed on the design of tidal generator arrays. Due to the nature of water as opposed to air, tidal generator arrays face many more problems in terms of turbulence in the wake than wind farms. A typical wake spacing of a wind farm is 5D (five times the blade's swept diameter)[9] while the most common spacing in a tidal array is 8D-10D[5]. It has also been found that the lateral spacing of the arrays can affect the wake and tidal turbines should not be placed closed than 2D[5]. On top of this, techniques such as staggering the array have been shown to yield no benefit past the second row[8, 9]. These complications in the design of tidal generator arrays have led to much research into the characteristics of large tidal generator arrays.

3 – Test Cases

3.1 – Standard Functions

3.1.1 – Description

In order to test the initial setup of GANetXL, a series of mathematical functions will be set up for use by the algorithm. These are presented as a set of complex functions in the form of $z = f(x, y)$ with the general aim of minimising the z-component by finding the optimum x and y values within a predefined range. The functions used in this test allow for the speed and accuracy of a genetic algorithm solver to be quantified but also allow insight into the ideal setup for a variety of problems. The exact functions used are created for the purpose of testing such algorithms[2].

3.1.2 – Setup

The setup of the algorithm to run these test functions is relatively straight forward. Two cells within Excel are chosen to represent the x and y component of the input. Using these, a third cell containing the chosen function calculates the corresponding z-value. These three cells are then specified to GANetXL for their respective types. The input variables are restricted to within the boundaries specified by the problem and the settings of the GA solver are adjusted by the preference of the operator. The objective of the algorithm is to minimise the z-component generated by the input variables but the overall accuracy of the result is the divergence from the known solution. It is likely for small divergences to appear due to the high precision of the variables but larger divergences will be representative of a failure to optimise the function. The series of functions used are shown below in table 2 with corresponding graphic plots of the functions available in Appendix A.

Name	Function	Graphic
Ackley's function	$f(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right) - \exp\left(0.5(cos(2\pi x)+\cos(2\pi y))\right) + 20 + e.$	Figure 20
Sphere function	$f(\boldsymbol{x}) = x^2 + y^2$	Figure 21
Rosenbrock function	$f(\boldsymbol{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	Figure 22
Beale's function	$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$	Figure 23
Goldstein-Price function	$f(x,y) = (1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2))(30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2))$	Figure 24
Booth's function	$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$	Figure 25
Bukin function N.6	$f(x,y) = 100\sqrt{\lvert y - 0.01x^2\rvert} + 0.01\lvert x + 10\rvert$	Figure 26
Matyas function	$f(x,y)0.26(x^2 + y^2) - 0.48xy$	Figure 27
Lévi function N.13	$f(x,y) = \sin^2(3\pi x) + (x-1)^2(1 + \sin^2(3\pi y)) + (y-1)^2(+\sin^2(2\pi y))$	Figure 28
Three-hump camel function	$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$	Figure 29
Easom function	$f(x,y) = -\cos(x)\cos(y)\exp\left(-((x-\pi)^2 + (y-\pi)^2)\right)$	Figure 30
Cross-in-tray function	$f(x,y) = -0.0001\left(\left\lvert\sin(x)\sin(y)\exp\left(\left\lvert 100 - \frac{\sqrt{x^2+y^2}}{\pi}\right\rvert\right)\right\rvert + 1\right)^{0.1}$	Figure 31
Eggholder function	$f(x,y) = -(y+47)\sin\left(\sqrt{\left\lvert y + \frac{x}{2} + 47\right\rvert}\right) - x\sin\left(\sqrt{\lvert x - (y+47)\rvert}\right)$	Figure 32
Hölder table function	$f(x,y) = -\left\lvert\sin(x)\cos(y)\exp\left(\left\lvert 1 - \frac{\sqrt{x^2+y^2}}{\pi}\right\rvert\right)\right\rvert$	Figure 33
McCormick function	$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$	Figure 34
Schaffer function N. 2	$f(x,y) = 0.5 + \frac{\sin^2(x^2 - y^2) - 0.5}{\left(1 + 0.001(x^2+y^2)\right)^2}$	Figure 35
Schaffer function N. 4	$f(x,y) = 0.5 + \frac{\cos(\sin(\lvert x^2 - y^2\rvert)) - 0.5}{\left(1 + 0.001(x^2+y^2)\right)^2}$	Figure 36
Styblinski-Tang function	$f(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}$	Figure 37

Table 2: Table of standard test functions and associated figures in appendix A

3.1.3 – Results

		Target			Solution			
ID	**Function**	**X**	**Y**	**Value**	**X**	**Y**	**Value**	**Variance**
1	Ackley's Function	0	0	0	4.77E-06	4.77E-06	1.91E-05	1.91E-05
2	Sphere Function	0	0	0	-1.41E-05	-6.68E-06	2.44E-10	2.44E-10
3	Rosenbrock Function	1	1	0	0.99979176	0.999895964	4.34E-08	4.34E-08
4	Beale's Function	3	0.5	0	2.999798298	0.499952316	6.64E-09	6.64E-09
5	Goldstein–Price Function	0	-1	3	-3.81E-06	-0.999998093	3.000000007	6.81E-09
6	Booth Function	1	3	0	1.00006628	2.999950886	7.98E-09	7.98E-09
7	Bukin function N.6	-10	1	0	-12.10668288	1.46571776	0.044827078	4.48E-02
8	Matyas	0	0	0	-4.77E-06	-4.77E-06	9.09E-13	9.09E-13
9	Levi N.13	1	1	0	1	1.000009	1.03E-10	1.03E-10
10	Three-hump Camel	0	0	0	4.77E-06	4.77E-06	9.09E-11	9.09E-11
11	Easom	3.14159	3.14159	-1	3.141787742	3.149417137	-0.999908113	9.19E-05
12	Cross-in-tray	1.34941	1.34941	-2.06261	1.349721598	1.348825144	-2.062611817	1.82E-06
13	Eggholder	512	404.2319	-959.6407	512	404.2310173	-959.640662	3.80E-05
14	Holder Table	8.05502	9.66459	-19.2085	-8.055027988	9.664592583	-19.20850257	2.57E-06
15	McCormick	-0.54719	-1.54719	-1.9133	-1.5	-2.95604034	-2.98156341	1.07E+00
16	Schaffer N.2	0	0	0	-25.00000894	-24.99999702	0.401234664	4.01E-01
17	Schaffer N.4	0	1.25313	0.29258	0	1.25313	0.54018	2.48E-01
18	Styblinski-Tang	-2.9035	-2.9035	-78.332	-2.9035	-2.9035	-78.332	3.51E-04

Table 3: Target values, calculated values and fitness of results for various test functions

3.1.4 – Review

The purpose of this test is to evaluate how the GA solver deals with single objective optimisation problems. The exact performance can be evaluated both numerically and qualitatively from the results in table 3 and the observations made during runtime.

The most straight-forward test is the accuracy of the solution offered by the GA solver against the known solution. Generally speaking, the algorithm was able to reach a value less than 0.0001 away from the true value within a run time of 30 minutes and 10,000 generations. This is a suitable degree of accuracy considering the complexity of some of the functions. These functions were created to create a complex solution space to properly test the algorithm. As such, it is expected that the solution space of the other problems presented in this report will not be as complex suggesting there is no problem with the capability of GANetXL in this regard.

The accuracy of the solution should be considered in relation to the shape of the test region. The shapes of the test functions can be categorised into three groups to better evaluate how GANetXL deals with various types of problems. The first type is an open region with sloping sides leading towards a minimum on the surface. In a case like this the approach required is the cumulative effort of small mutations and inter-breeding of solutions close to the minimum. A simple example of this is the Sphere function but the Matyas function provides a more complex example. In both cases, GANetXL was able to produce the end result to an accuracy of around $1x10^{10}$ which shows the capacity of the solver to deal with test regions of this type.

The second type of problem is one in which sub-regions of test region have their own minimums of varying magnitude and are separated by regions or relatively large magnitude. In this case the approach by the solver required to reach the overall minimum is the use of large mutations to jump between the sub-regions to find the overall minimum. If the algorithm is not capable of providing a suitable number of large mutations, solutions will be confined to incorrect sub-regions. Once within the goal sub-region, the approach described as the first type of problem above must be utilised to converge on the minimum point. Some examples of a problem such as this are the Cross-in-tray function, the Hölder table function and the comparatively more complex Eggholder function. In these cases the GA solver was able to produce a result within $1x10^5$ of the known solution which although greater than the accuracy of the results in the previous case is still within a suitable degree of accuracy. As such, GANetXL can be deemed appropriate to deal with functions of this type.

The third case is one in which the minimum within the test region exists at a single point within a small sub-region. As such a degree of luck is required on behalf of the algorithm to locate the small sub-region within the test space in which the minimum exists by means of a random mutation. A case such as this tests the ability of the GA solver to evaluate an entire test region and will provide insight into the sizing of population, elitism level and number of generations required in order to find a single point solution. While it is unlikely for a problem such as this to be encountered in the following test cases and in the analysis of tidal generator arrays, the insight into the operation of the solver is useful. Based on the running of the Easom function and the Schaffer functions, it can be found that a high mutation rate (>50%) yields a greater benefit than a large population size. In these cases, a population size of 75 was sufficient and significantly faster than a population size of 100. In these three functions, the minimum was found to an accuracy of $1x10^5$ for the case of the Easom function and around 0.2 for the Schaffer functions. This suggests that a case like this should be considered carefully and solutions should be evaluated over a number of runs to better determine the shape of the test region and the required configuration. If a test region can be identified to be of this third type, GANetXL's generation feedback could be used to allow the entire test space to be examined more freely in earlier generations without affecting the accuracy of a fitness function towards the end of a run.

The key lessons to take from the running of these test cases are the setup of the solver with regards to the mutation of the genomes. From here on the genes will be considered as discrete and each will have a random chance of mutation varying between 30% and 60% to allow free mutation across the test region without losing the progress of the algorithm towards the true minimum. This increase over the default mutation rate of 5% will be taken forward to the other problems in this report.

The running of these cases has also provided insight into understanding the shape of the test region based upon the shape of the curve of the best solution against the generation. Certain trends within this curve can be interpreted as the shape of the test region which allows the operator to alter the configuration of the solver to be more accurate on the next run of the algorithm.

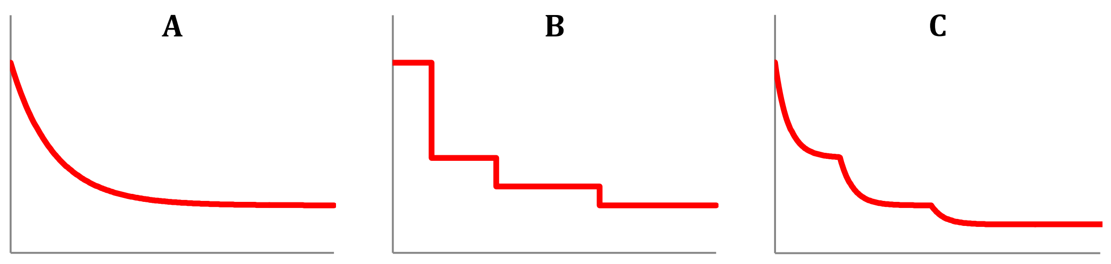

Figure 4: Representative shapes of best solution value (y axis) against generation (x axis)

The case presented above in figure 4A is typical of a problem where the optimal solution lies within a region that slopes towards it as described as the first case in previous paragraphs. As such the best approach to the solution is a large population size and the reliance on small mutations to converge towards the optimal value. The case presented in figure 4B is typical of a solution with a large number of sub-regions within it. The vertical sections of the graph represent the points in which a random mutation in one of the genomes reaches a sub-region that is generally more optimal than any of the sub-regions previously occupied. These vertical drops are likely to occur more frequently and to a higher degree closer to the beginning of the run of the algorithm since later on there will be fewer sub-regions available with more optimal values and thereby less chance of mutating to them. If a case such as this is observed, the best approach is to increase the chance of mutation in the genomes and introduce an elitism level if one is not already present to retain the benefit of such a random mutation and increase the chance of mutating to the optimal sub-region. The actual graph generated by a problem may be a combination of these two graphs as shown in in figure 4C which represents a problem with a number of sub-regions with sloping sides. These observations are very subjective and should only be used as an indication to the possible shape of the solution space.

3.2 – Single Objective Test Case – Fourier Synthesis

3.2.1 – Description

Fourier synthesis is a process by which a complex waveform is broken down into a series of sinusoidal waves which best approximates it. This serves a great practical purpose in the engineering industry since it allows for complex waveforms defined by a series of points to be converted into a set of sinusoids on which numerical calculations can be used. In terms of calculation using a genetic algorithm, the problem represents a single objective problem with a large number of genes per genome. On top of this, some complexity is added in the form of accurately calculating the fitness function and dealing with looping input variables, especially the phase of the sinusoids.

3.2.2 – Setup

The setup of this calculation will require a set of variables to define a series of sinusoids and a function to calculate their sum over a range. The main problem with the setup is that version of GANetXL being used for this project is limited to 30 genes which in terms of this problem means a limit of 10 sinusoids defined in amplitude, frequency and phase or 15 sinusoids defined only in amplitude and frequency. It can be shown that any input function can be represented by a set of sinusoids with zero phase. However, by allowing the algorithm to manipulate phase it can be found if the algorithm produces a comparable result.

In terms of gauging the accuracy of the solution a numerical approach must be used. While it may be possible to compute the divergence from the ideal solution using integral methods, this is not possible in excel. Instead a large set of angles will be generated at set intervals and the value of a given sinusoid at this point will be calculated. The sinusoids will be summed and the divergence from the input waveform will be found. Finally, these errors will be summed to provide a fitness function which the algorithm will aim to minimise.

3.2.3 – Results

The results of these test cases are in the form of the 30 genes that define the sinusoids for each case respectively. For each case, the values of A and F represent the amplitude and frequency respectively for the expected forms explained in the section above. The phase has been omitted from this set of results for reasons discussed in the review section below. While cases were run with phase included it was too problematic to include within final results. Due to the nature of the algorithm, the solution generated features some sinusoids with similar frequencies the need to be combined to test the accuracy of the result. This is done by evaluating from the frequency the expected frequency it is closest to. If more than one sinusoid has the same expected frequency, the amplitudes are summed and the frequencies are averaged. This will create a set of sinusoids with unique expected frequencies, making evaluation of the accuracy easier. For each of the three waveforms, the 15 sinusoids generated by GANetXL are shown below along with the refined list with similar frequencies combined.

3.2.3.1 – Sawtooth Waveform Results

Sinusoid	Amplitude	Frequency
1	0.08701703	0.002822887
2	0.128175548	0.999073025
3	0.824848929	1.000675204
4	0.005707135	1.982505732
5	0.137105536	1.998832698
6	0.274949643	2.001960762
7	0.188872612	3.001644141
8	0.068082769	4.003921524
9	0.000537142	5.103168881
10	0.000604285	9.56569506
11	0.000469999	11.55903457
12	0.000469999	12.55818389
13	0.000402857	13.80033035
14	0.000134286	17.78693309
15	0.000402857	17.83400663

Table 4: Sinusoids generated for a sawtooth waveform

Sinusoid	Amplitude	Frequency	Expected Amplitude	Expected Frequency	Amplitude Error	Frequency Error	Combined Error
1	0.08701703	0.002822887	0	0	0.08701703	0.0028229	0.0002456
2	0.953024477	0.999874115	1	1	0.04697552	0.0001259	5.914E-06
3	0.417762315	1.994433064	0.5	2	0.08223769	0.0055669	0.0004578
4	0.188872612	3.001644141	0.333333333	3	0.14446072	0.0016441	0.0002375
5	0.068082769	4.003921524	0.25	4	0.18191723	0.0039215	0.0007134
6	0.000537142	5.103168881	0.2	5	0.19946286	0.1031689	0.0205784
7	0.000604285	9.56569506	0.1	10	0.09939572	0.4343049	0.0431681
8	0.000469999	11.55903457	0.083333333	12	0.08286333	0.4409654	0.0365399
9	0.000469999	12.55818389	0.076923077	13	0.07645308	0.4418161	0.0337782
10	0.000402857	13.80033035	0.071428571	14	0.07102571	0.1996696	0.0141817
11	0.000537142	17.81046986	0.055555556	18	0.05501841	0.1895301	0.0104276
						Total	0.1603341

Table 5: Combined sinusoids and errors for a sawtooth waveform

3.2.3.2 – Square Waveform Results

Sinusoid	Amplitude	Frequency
1	0.288579625	0
2	0.095208448	0
3	0.009467131	0.000305176
4	0.987871574	1.000978472
5	0.003088567	2.992177948
6	0.301135323	3.002935415
7	0.030012818	5.002603536
8	0.102862724	5.005044947
9	0.016987121	5.005197535
10	0.0106757	7.000669481
11	0.068552768	7.006849302
12	0.034309955	9.006822599
13	0.00241714	9.009416598
14	0	10.68559777
15	0.012824269	11.00580407

Table 6: Sinusoids generated for a square waveform

Sinusoid	Amplitude	Frequency	Expected Amplitude	Expected Frequency	Amplitude Error	Frequency Error	Combined Error
1	0.393255204	0.000101725	0	0	0.3932552	0.0001017	4E-05
2	0.987871574	1.000978472	1	1	0.01212843	0.0009785	1.187E-05
3	0.304223891	2.997556682	0.333333333	3	0.02910944	0.0024433	7.112E-05
4	0.149862663	5.004282006	0.2	5	0.05013734	0.004282	0.0002147
5	0.079228469	7.003759391	0.142857143	7	0.06362867	0.0037594	0.0002392
6	0.036727095	9.008119599	0.111111111	9	0.07438402	0.0081196	0.000604
7	0.012824269	10.84570092	0.090909091	11	0.07808482	0.1542991	0.0120484
						Total	0.0132293

Table 7: Combined sinusoids and errors for a square waveform

3.2.3.3 – Triangular Waveform Results

Sinusoid	Amplitude	Frequency
1	1.019935299	0
2	-0.07502289	0.000152588
3	-0.00686077	0.000839235
4	0.007025575	0.001449588
5	0.002740646	0.003128058
6	0.993352866	1.000062943
7	-0.098431301	2.999883652
8	0.001232985	4.982614484
9	0.038686443	5.000085831
10	0.010993103	8.998964308
11	-0.000592077	9.437731624
12	4.27272E-05	15.52783113
13	-0.001941036	18.98898886
14	4.27272E-05	25.47329993
15	0.002185192	32.99948311

Table 8: Sinusoids generated for a triangular waveform

Sinusoids	Amplitude	Frequency	Expected Amplitude	Expected Frequency	Amplitude Error	Frequency Error	Combined Error
1	0.94781786	0.001113894	0	0	0.94781786	0.0011139	0.0010558
2	0.993352866	1.000062943	1	1	0.00664713	6.294E-05	4.184E-07
3	0.039919429	4.991350157	0.04	5	8.0571E-05	0.0086498	6.969E-07
4	0.010401025	9.218347966	0.012345679	9	0.00194465	0.218348	0.0004246
5	4.27272E-05	15.52783113	-0.004444444	15	0.00448717	0.5278311	0.0023685
6	-0.001941036	18.98898886	-0.002770083	19	0.00082905	0.0110111	9.129E-06
7	4.27272E-05	25.47329993	0.0016	25	0.00155727	0.4732999	0.0007371
8	0.002185192	32.99948311	0.000918274	33	0.00126692	0.0005169	6.549E-07
						Total	0.0045968

Table 9: Combined sinusoids and errors for a triangular waveform

3.2.4 – Review

A series of problems were encountered during the operation of this test case which shows some of the limitations of a general use GA solver such as GANetXL. While it is not always possible for a custom algorithm to be created and used for a problem, they can provide a great benefit for specific problems such as this one. However, a series of workarounds and techniques can be implemented to allow a reasonable result to be generated.

Working with the phase of the sinusoids created a series of problems since all variables in GANetXL are treated as being on a linear scale while phase loops every 360 degrees. Initially it would be reasonable to represent the phase as a real number between 0 and 360 but this restricts a small mutation from occurring close to 0 degrees to the opposite end of the scale which limits the effectiveness of the algorithm. Knowing this, the scale could be expanded to a real number between -20 and 380 but this just presents the same problem at a different value. The chosen solution to this problem was to define the phase as a real number between -180 and 540 but apply a negative weighting for every degree under -90 and above 450 the variable is. This allows for the phase to exist at any value without penalty and allows for small mutations to occur at every non-penalised value with minimal impact. This also prevents the variable from hitting the end of the scale to allow for a more natural optimisation. The ideal solution would be a function within the solver to allow for looping scales and normalise a variable to within a defined looping region. Despite a solution being reached, the phase was ultimately removed since the constraints to keep the phase in range were affecting the fitness of the entire genome and the large range of phases caused problems when breeding solutions.

When dealing with waveforms the effects of aliasing must be considered to get an accurate solution. The composite waveform in this test is measured at a series of points spaced evenly along a continuous scale of a given length. In order to accurately map the composite waveform, the spacing of the points must be less than a quarter of the wavelength of the sinusoid with the highest frequency. Without this specification the effect of the small wavelengths would be lost in the calculation. On top of this, the overall scale must be long enough to map the entire wavelength of the sinusoid with the lowest frequency. This is because it can have a negligible effect at certain regions of the scale while creating something analogous to a zero error at other parts of the scale. Generally speaking, the long the scale the better since it can check for the effect of resonance of the sinusoids at specific points but there is no reason to measure beyond the lowest common multiple of the input sinusoids wavelengths. There is no way within Excel to dynamically allocate a number of test cells or alter the spacing of the test point without greatly diminishing the speed of the algorithm or overcomplicating the implementation. Because of this the only course of action is to create a suitable large number of test points with a suitable small spacing to suit all cases.

The calculation of this problem revealed the limitations of using Excel as the foundation of a genetic algorithm solver. This calculation relies on the generation of values of sinusoids over a large number of data points (~3500 points). For each genome in every generation throughout the calculation, these points need to be calculated and are inserted into the spreadsheet and the sheet is updated which takes a significant amount of computation power. This computation is amplified based on the number of dependant cells to be updated and the relative complexity of the relevant cells. Initially, the Excel sheet being used for this test had cells calculating the individual input of each of the input sinusoids which ultimately proved impractical as the calculation would run at the rate of one generation every few seconds. By combining the calculation of the sinusoids at each test point to a single cell the calculation time was sped up to around 20 hours for 10,000 generations at the sacrifice of readability and a layout more prone to human error. Many of the solutions shown in the results above are cut before reaching the full 10,000 generations and instead were ended after remaining at a minimum for 1,000 generations or so. This problem showed the effect of minimising the numbers of updating cells in excel has on the speed of the calculation and shows the problem with calculating large numbers of data points for each genome.

The solutions generated by the algorithm were able to replicate the sets of expected results to a reasonable degree of accuracy. As the expected amplitude of a given sinusoid gets smaller, the effects of variations in its frequency are lessened and thus are more likely to diverge from the true solution. This was countered by altering the fitness function to place more emphasis on divergence of lower amplitudes. This approach kept the smaller sinusoids within a reasonable degree of accuracy and overall, enough of the sinusoids in each case were accurate enough for a trend to be observed.

3.3 – Multiple Objective Test Case – Pipe / Pump System

3.3.1 – Description

A pipe and pump system is a fairly standard engineering problem in which the design of a system specifies an operating point which in turn defines the characteristics of the system. The variables open to the design are the diameter of the pipe, the size of the pump, the operating speed of the pump and the material of the pipe. The pipe is a defined length and features a series of bends and valves which are accounted for as small losses. While this setup could represent a wide variety of engineering problems, in this case it will be used to represent the scenario of refilling a reservoir. A pipe system connects a reservoir to a pump a fixed height below it. The pump has a fixed amount of time to pump a specified volume of water up to the reservoir on a daily basis. For this to occur, the head generated by the pump must be equal to the head lost in the pipes during the transport of the water. Knowing this head, the operating point of a pump of a given specification can be found which in turn can be used to define the efficiency of the system. On top of this, there are number of economic concerns derived from the specification of the pipe and pump in terms of initial setup cost. The wide variety of output variables makes this suitable for a multiple objective solver which serves as good preparation for the tidal generator array problem approached later in this report. Initially the balance between setup cost and running cost will be looked at but later on, other configurations can be considered.

3.3.2 – Setup

The setup of this problem within Excel followed a set of standard formatting aimed at making the operating of the genetic algorithm solver easier. The input variables to be manipulated by the algorithm were grouped and their boundaries were listed for reference. All the fixed variables were grouped into two categories; the design variables (e.g. required water volume, maximum operation time) and the fixed variables (e.g. density of water, value of pi). The main calculation takes place over a series of points defined by the characteristics of the pump being used. Based on input variables, a set of flow rates, pump heads and pump efficiencies are calculated from the associated coefficients. Using each respective flow rate the flow speed can be found which in turn can be used to find Reynolds number which in turn can be used to calculate the friction factor and thereby the head lost in the pipes by a combination of the Darcy-Weishbeck, Bernoulli and small losses formulas[4]. Using these values, a set of data best represented by a graph similar to the one shown below in figure 5 can be generated.

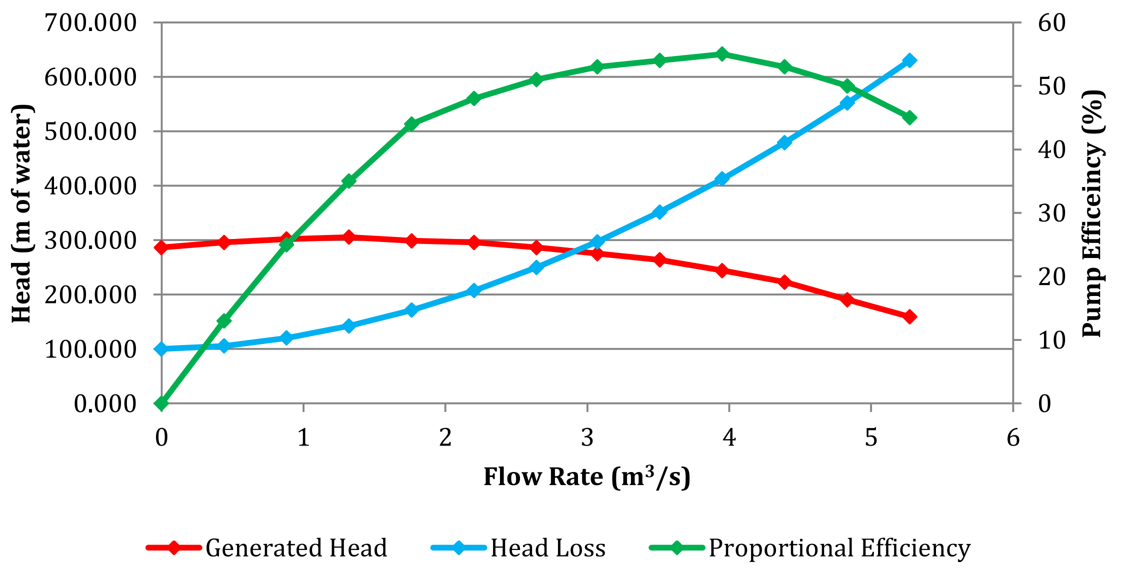

Figure 5: Representative graph of pump head, head loss in pipes and pump efficiency at varying flow rates

In order for output variables to be calculated, the flow rate at which the lines for head loss and generated head cross must be found along with the associated efficiency. The method used must be capable of operating autonomously and must account for all special cases such as potential cases where the lines do not cross.

In order to find this operating point, some form of interpolation must be used. The lines appear curved meaning it would be reasonable to use a second order interpolation. However, due to the nature of the implementation the formula would be overly complex and prone to error should anything greater than a linear interpolation be used. In a situation like this, the trade-off between calculation time and accuracy must be considered. On top of this, some method needs to be discerned to find out which points the crossover occurs between. The simplest method is to find the difference between the two lines at each point and look for a change in sign to indicate the lines cross. Special cases can be accounted for by applying a large negative weighting to solutions where the lines do not cross since it indicates there is no viable operating point for the system and thereby it should be excluded.

Once the operating point has been found some small calculations are used to generate a set of output values. In practise it is best to keep these tabulated in a table and have a separate section to hold the optimisation variables known to the genetic algorithm solver. This allows for the optimisation variables to be easily interchanged without altering the setup of the solver.

This problem introduces a series of constraints which every solution must fulfil. The only constraint put forward by the problem is that the reservoir needs to be filled within a certain time but more constraints are needed to get a good result. As mentioned earlier, a large weight is placed on there being an operating point in the system and there are also constraint placing negative weight on overly expensive (but still viable) configurations to steer the algorithm towards more reasonable solutions.

3.3.3 – Results

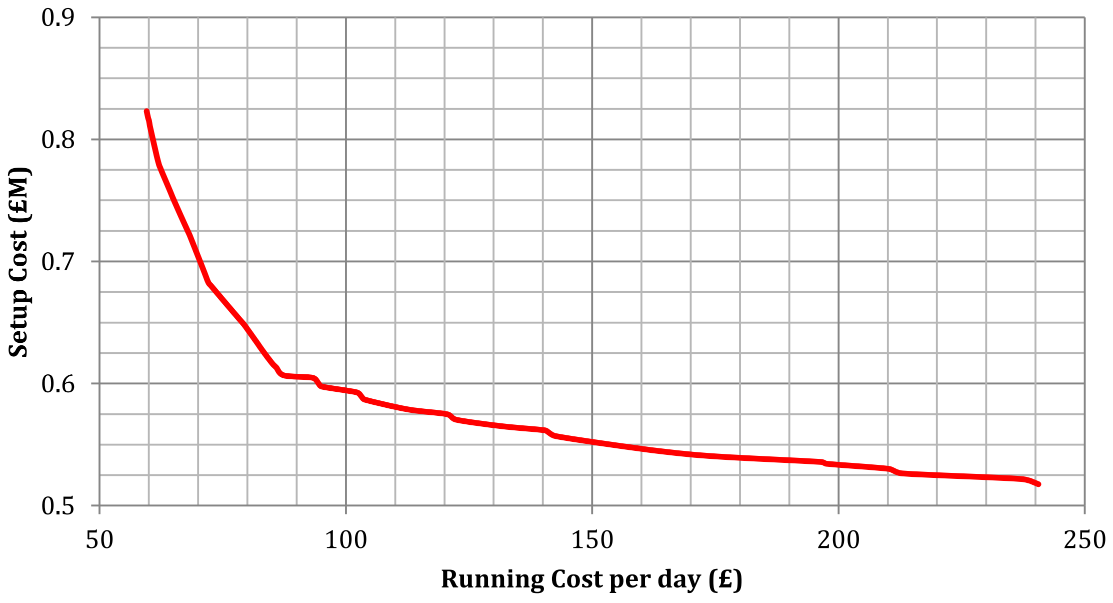

Figure 6: Pareto front for pipe/pump system in terms of setup and running costs

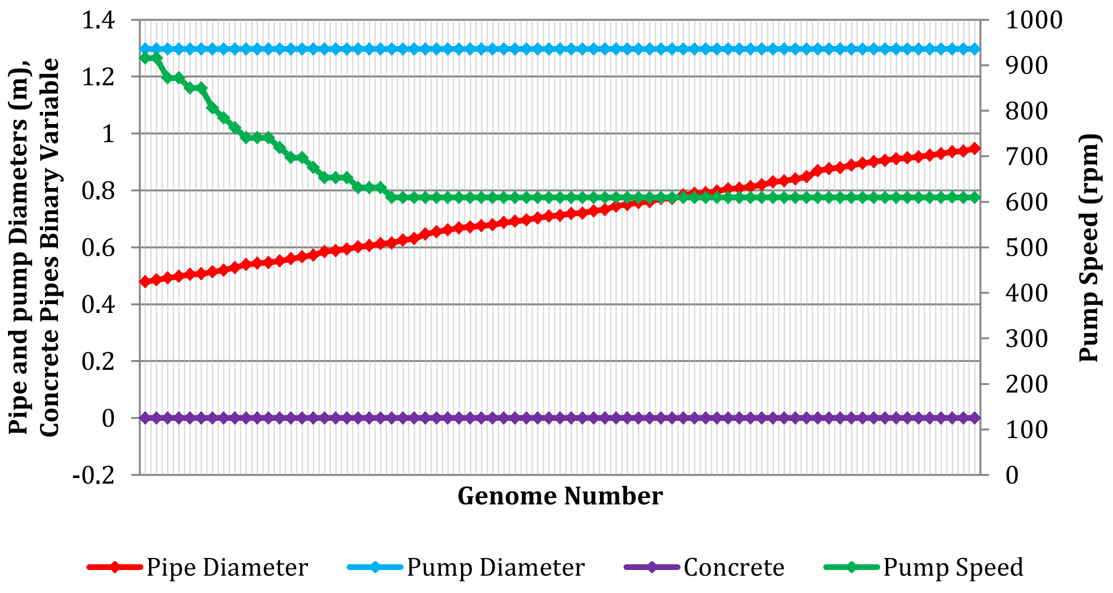

Figure 7: Variable map of Pareto front for a pipe/pump system marked against genome number

3.3.4 – Review

This test case showed how to deal with certain types of constraints. The addition of an infeasibility constraint in this problem meant that constraints had to be dealt with differently. Any solution that does not have an operating point cannot be used the aim should be to remove it from the gene pool. Ideally, the faulty genome should be replaced with a clone of another viable genome or rolled back to a previous generation where it is still in a viable range. However, these methods would require access to the genetic algorithm's programming. The solution of adding a large negative weighting on any genome that doesn't have an operating point prevents solutions from mutating far into the unviable region of the solution space.

The other constraints in this exercise can be dealt with in another way which allows for a negative weight to be applied without making the solution unviable. This is done by finding how far the solution is over the boundary specified by the constraint. This divergence can then be manipulated to produce a negative weighting for the solution. This has the benefit that small mutations close to the boundary do not suffer large negative weightings allowing them to operate close to the boundary to find ideal solutions. Also, if a solution is in a constraint region such as this a series of small mutations are more likely to bring the solution to an unconstrained range than a uniformly distributed negative weight as shown in the first test case. Since these constraints are managed in this way, the final generation should be checked for viability since it is possible for a final solution to exist past the boundary due to minimal weighting from the constraint.

Figure 6 is the Pareto front of the optimal solutions to the problem and gives an insight into the manner that multiple objective problems present results. Every solution on the line represents an optimal solution and selecting the chosen solution entirely depends upon the balance between the variables on each axis. This balance could be expressed by a line originating from the origin of the graph and with a gradient representative of the ideal relationship between the two decisive variables. This line could then be traced and the rating of each individual solution along the Pareto front could be ranked via a 'least squares' sort. Using a method such as this allows for the process of selecting the chosen solution from a Pareto front to be done autonomously.

It must be understood that the Pareto front is made up of discrete solutions that can vary greatly in terms of their genes and as such they cannot be interpolated between easily. It is possible that there is no viable solution between two known solution points lying on a Pareto front. However, a more common problem associated with attempting to select a point on the front as opposed to a known solution is that there is no direct relationship between the position on the front and the values of the solution's variables. As shown in figure 7, the variables making up each solution on the front vary along its length in a non-linear manner. In this case certain trends are apparent such as the locking of the pump diameter and the concrete binary variable to a singular value. The pipe diameter appears to increase linearly while the pump speed levels off to a constant value in an exponential manner. While these trends appear simple they can only be confirmed to be true within this sample range and such trends cannot be expected in more complex engineering problems.

The solutions presented along the Pareto front for this problem (shown in figure 7) have some interesting qualities and show some of the conclusions that can be drawn from the results of a genetic algorithm. The concrete variable can only exist in two states suggesting that either concrete pipes or plastic pipes should be used. In every case along the Pareto front plastic pipes are chosen which suggests they are categorically better than concrete pipes. While this conclusion may be true, a result like this may suggest the model being used does not accurately model the benefits of using concrete. Were this model to be improved, factors such as supply, transport costs, carbon footprint and installation could be expanded upon to more accurately represent the benefits of concrete pipes.

The pump diameter presents an interesting result in that it remains constant across the entire range. This suggests that in all cases a pump with a diameter of 1.30m is optimum in all cases. It is also known that this is not an anomaly of the algorithm since the result remains the same over different runs of the same problem. Were the factor excluded from calculation of the fitness function by an error in the spreadsheets setup, the variable would appear random since mutations of the gene would appear no less favourable. A result like this is ideal since it suggests there is a relationship to be derived between the fixed variables which define the problem and a specific output variable, allowing it to be possibly removed as a factor of the calculation. This combined with the knowledge that plastic pipes are categorically better means this problem can be reduced to only have two degrees of freedom. This would then allow for optimum solutions to be found by a simple analytical approach as opposed to the trial-and-error analogue of a genetic algorithm.

4 – Tidal Generator Array Design

4.1 – Overview

The design of a tidal generator array is a complex problem which can be optimised by a number of different standards. A project such as this would be often considered as an option among other power generation methods. For this purpose the optimisation of the design to minimise the cost while maximising the output in terms of both power and revenue is very important. On top of this, it is likely for a number of constraints to be imposed upon the design brief such as a maximum budget, a power generation quota and generalised efficiency. The aim in setting up this algorithm is to find a setup which will reliably produce a set of optimised results within given constraints in a reasonable amount of time.

4.2 – Problem Definition

The standard design to be modified by the algorithm is a set of tidal generators in a linear pattern. The region in which these generators are in is assumed to be a cuboid of water of a fixed depth. The exact width, length and depth are specified by the design brief along with the speed of the flow which is assumed to enter the region normal to one of the vertical faces. The turbines are to be facing towards the flow to have the wake restricted to a single axis. In reality the flow is likely to fluctuate and change within the region in terms of both direction and magnitude. This approximation is reasonable since it assumes that only the flow component in the same orientation of the turbines will generate power and assuming the region is chosen such that it faces the direction of prevailing flow the effects of lateral flow will be negligible. Of course, a more complex algorithm capable of handling a map of flow velocities as opposed to a generalised flow will provide more accurate results but that is not possible within the Excel and GANetXL infrastructure.

The design of a tidal generator for the purpose of this project is similar to that of a standard wind turbine. A turbine with a number of blades is supported by a monopile foundation. There is some limitation to the size of the blades defined mainly by the depth at the site. The sum of the length of two blades gives the disk diameter representing the swept locus of the turbine henceforth referred to as the disk size ‘D’. This cannot exceed 8m less than the total depth at the site. The 8m quantity is distributed above the disk to allow the turbine to operate with enough clearance for ships to pass by undisturbed. It is assumed also that the monopile can be any height necessary to support the turbine head at the required point. The height is also specified such that a small amount of space is left below the turbine disc to prevent turbulent or physical effects from the sea floor. Based on analysis of the power formula it has been determined the turbine should be as high as possible in the water stream to maximise its output. As such it will be placed at a height such that only 8m of space lies above it to the surface of the water. The cost of a given turbine is calculated using the characteristic height of the monopile and the size and number of the blades to calculate a volume on which a costing function is used.

The technical setup of this problem including the various formulas is adapted from a spreadsheet provided by Tidepod ltd, a company in the tidal power sector[5]. This was then modified and formatted for use within GANetXL.

4.3 – Setup

In order to specify the layout of the turbines and by association the foundations to the algorithm, a function needs to be used to generalise the array. In an ideal case, the coordinates of each turbine could be specified as a distinct gene allowing for the layout of the array to be optimised beyond a simple linear pattern. However, the number of genes available is limited making this method impossible with GANetXL. Instead the linear pattern will be assumed by the algorithm requiring only the spacing between the turbines to be specified. Due to the nature of the problem, the spacing between rows and the spacing between columns have different effects and as such are treated distinctly. Beyond this, there are two equivalent ways of specifying the spacing in the array. The direct spacing between turbines could be specified in terms of a distance or as a ratio of the disk size. Alternatively the number of turbines in a given row or column could be specified and the spacing could be calculated. While the second method allows for a more human readable value, the relationship between disk size and spacing would allow for the value to remain both valid and relevant should the disk size be changed making it more suitable for use in a genetic algorithm.

Other variables open to the genetic algorithm include the disk size of a turbine and the number of blades. As mentioned earlier, the disk size of the turbine is limited by the depth of the water at the site. Since the depth of the site is assumed to be constant across the site, the limit can be specified to GANetXL prior to running. It is worth noting that the minimum size of the turbine is only limited by their manufacture. The Tocardo T100 turbine is considered small with a diameter of 3.1m[7] so the minimum size will therefore be set at 3m. Were the site defined more specifically as a map of heights it is likely that the blade size would be limited to the shallowest depth to allow for all the turbines to be manufactured by the same process. Since this is an exercise in array optimisation, this factor will be ignored.

The calculation of the power generated by the turbines involves the application of a standard formula relevant to flow past a turbine. This is applied in two sections: the first row and the remainder of the array. The reason for this is that the first row receives the flow travelling at an undisturbed speed. However, the rest of the array receives a lower speed due to the effect of the wake of the turbine in front of it. This is simulated by a percentage decrease from the free flow speed calculated from a standard formula. Experimental data shows that the wakes of the turbines have a cumulative effect in terms of flow speed reduction. This effect however causes negligible changes past the second row. In the interest of calculation time and simplicity the array will only be divided into these two groups. If the wake spacing is set such that only 1 row can fit within the site the power calculation of the obscured rows in the array will cause an error if they are not excluded from the calculation specifically. As such, a check will be put in place to deal with this possibility.

These four variables are kept within defined ranges to focus the results to an expected range and restrict the results from falling out of the model. As mentioned earlier the disc size is limited to a 3m minimum per a known turbine design. The maximum is set as 8m less than the depth or 25m (from a known large turbine), whichever is lowest. The lateral spacing is set to a minimum of 2 times the disk size (2D) since experimental and theoretical data suggests below this yield point the turbines interfere with one another's wake and as such should be avoided. A similar limit is placed on the wake spacing with a minimum of 4D which is derived from experimental data from flume tanks. The data is only accurate for spacing above 4D but it was noted at the spacing of 4D the velocity deficit was significantly high. As such, the minimum will be set at 4D. Both the lateral and wake spacing have an arbitrarily high maximum since it is expected they will converge to a normal range. Due to the common industry practise, the blade numbers will be allowed to be either 2 or 3 since sourcing turbines with more blades would be problematic.

4.4 – Preliminary Results

After setting up the problem for use within GANetXL the case was run several times to gain an insight as to the nature of the problem. While these results are not suitable to draw conclusions on the optimal solution for a given set of input criteria, they allow a better understanding of the nature of the problem so constraints can then be added to reach a better solution.

By treating this as a purely business minded problem the goal of the algorithm would be to maximise the yield while minimising the CAPEX. This approach produces a Pareto front which allows the user to select a given CAPEX corresponding to a budget and thereby select a configuration to maximise the yield. The yield in question is evaluated as the power generated by a unit area. This is directly related to the revenue from a given array and can therefore be used to evaluate profit over timescales varying over the array's lifespan.

Initial results suggest trends towards smaller sized turbines in closer packed arrays. A design such as this would have certain benefits and drawbacks. Having more turbines means a statistical increase in the likelihood of a turbine failing. This must be considered alongside the reliability of smaller turbines which is in general higher than larger alternatives. An array with a larger number of turbines would likely take longer to install and as such may accrue more costs during its construction. However, were a singular turbine to fail, it would be easier to replace and less power would be lost overall as a result compared to a larger turbine.

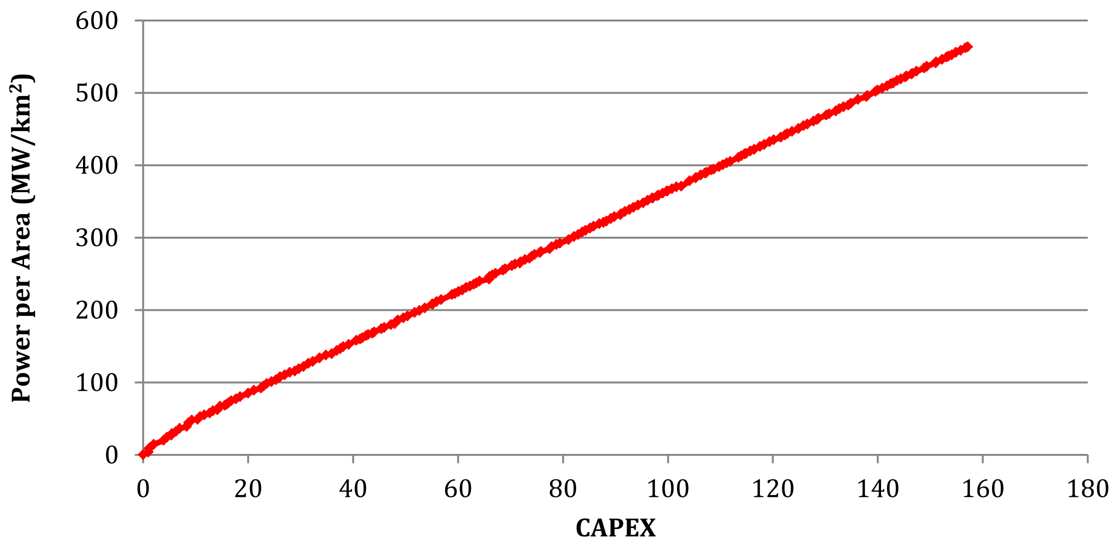

Figure 8: Pareto front of a tidal generator array charting Power per area against capital expenditure with a simplified model

As can be seen in figure 8 above, there is no significant change in the relationship between CAPEX and yield at this stage in the calculation within the range. Generally speaking this suggests there is no arbitrary change to an input criterion available to produce a significantly better configuration above a certain cost barrier. By considering the values of the input variables of the results along the pareto front, the possibility of fixed variables can be considered as well as the possible improvements to the range to which the variables are limited within. It is interesting that the Pareto front ends at a point close to £160M CAPEX despite having no arbitrary limit set by constraint. By considering the gene makeup of the solutions along the pareto front it can be possible to find why this was considered the upper bound for cost.

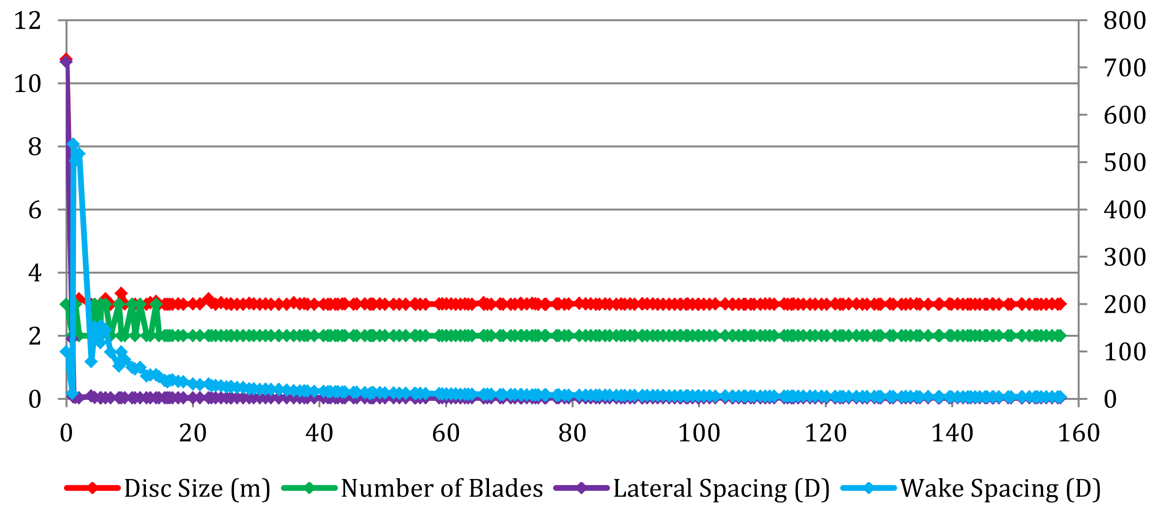

Figure 9: Variable map of Pareto front for a tidal generator array with simplified model

The data set from this initial test shown in figure 9 presents some promising results. The lateral spacing converged early on to the minimum value indicating the effects of the wake are being considered. The algorithm sees the benefit of closely packing rows since they do not affect one another whereas packing in the wake direction would have a negative effect.

Towards the end of the range, both the spacing variables converge to their minimum value maximising the number of turbines in the array. These results are promising since they seem to suggest the model is accurately representing the problem.

The disc size and the number of blades variables appear to be effectively locked at set variables throughout the range with some fluctuations at the lower end of the scale. The problem was run again with a constraint set such that results over £20M CAPEX would be excluded so these trends could be examined. These results are presented in appendix B. Looking at figure 39 suggests there is a relationship between the wake spacing and disc size likely caused by the fact wake spacing is presented in terms of the disc size. Nevertheless this means the disc size cannot be considered arbitrarily locked to an optimum variable. The variable of the blade number also appears to fluctuate at the bottom end of the range of figure 38 but the range does not match that of figure 9. This suggests the fluctuation may be a result of random mutation by the algorithm and the number of blades having a negligible effect on the result.

By looking at figure 38 it can be seen that there are some strange results below the yield of £4M CAPEX. The values of the wake spacing appear to suddenly drop after this point from a period of apparent random values. If this range proves problematic down the line it may be excluded from results to prevent a drain on genomes from calculating anomalous results.

By reviewing the individual genomes along the Pareto front and examining their secondary values some problems have been found. The formula calculating the velocity deficit appears to be producing negative values at some configurations which suggest the results being produced are not accurate. This will have to be considered before a final set of solutions can be generated.

4.5 – Solving Calculation Problems

The formula calculating the velocity deficit was found to be unsuitable at certain configurations and as such meant any solution offered in certain ranges would be unrealistically represented. This problem was not observed earlier since the formula was written primarily for manual use. Because of this it was kept within a standard range which was later extended for use by a genetic algorithm. To counter this, secondary variables must be managed during the running of the algorithm to mimic the judgement of a human operator. By excluding the range in which these impossible results occur in there would be no chance of the algorithm presenting a result that drops out of the model in this regard. However, this comes at the cost of the exploration of all the possibilities. In a case such as this the velocity deficit formula should be considered outside of the bounds of a genetic algorithm in favour of more stringent methods such as computational fluid analysis, wind tunnel simulation and field research. If after these approaches have been completed a model that can account for the effect of the flow velocity through arrays of these problematic configurations can be extracted, the range can then be extended to include them.

Finding a solution to this problem can be approached in a number of ways. An analysis of the formula can determine what regions the input variables will produce a non-physical result within. From this, the input variables can be restricted to never enter this range and thereby prevent any negative effects of receiving an unphysical answer. To build upon this concept of restricting the range of input variables, an approach to setting the range that input variables can mutate within can be considered. By allowing a variable to exist at any physically possible value (i.e. the turbine can exist in any height from microscopically small to the depth of the water) the algorithm is able to find solutions which may have been overlooked due to the unexpected nature of the design seeming incorrect to a human considering it. However, an approach such as this is only viable if the formulas governing it are consistent and accurate across the entire range. If this is not the case the variable should be restricted to a range that can be accounted for by the formula (i.e. flows past small turbines differing from flows past large turbines). Beyond this, a further level of restricting can occur to limit the variable within a pre-defined region that represents industry standards or expected values. This covers cases where the algorithm might suggest a value which would be ignored by the operator since it is known to not work in reality due to incompleteness of the model. This disparity with real world implementation and the accuracy of the model presents a serious problem in terms of using genetic algorithms in engineering design problems.

The formula for the velocity deficit which was producing values in an unexpected range is plotted in the figure below. The formula in question was a function of the wake spacing and the ratio of the turbine size to the depth. Therefore, the value was dependant on two input variables and as such the ranges where the velocity deficit would be within an expected range could not be simply calculated.

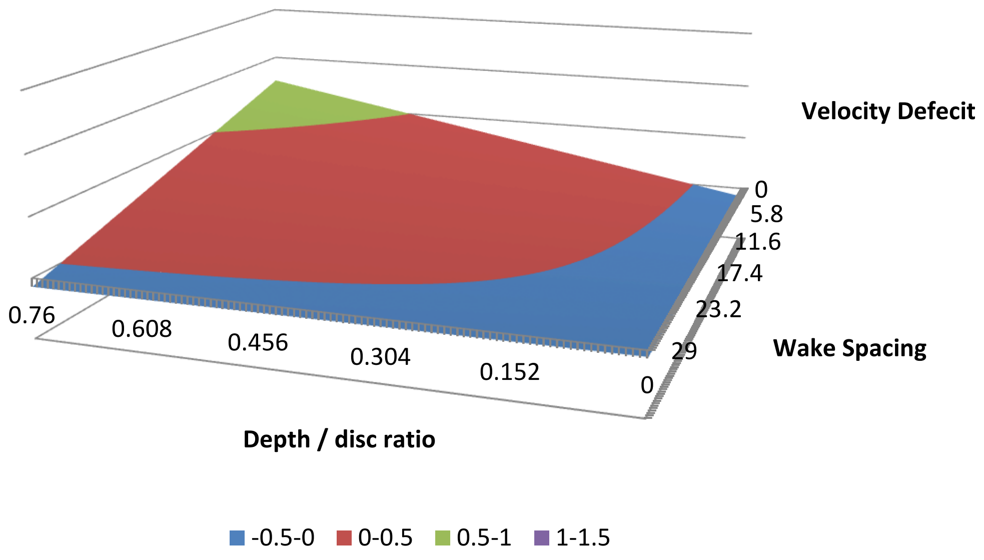

Figure 10: Graph of velocity deficit function in relation to its independent variables

Figure 10 (above) shows the region in which the accuracy of the function breaks down. The region in blue represents where the function acts as if the flow past the turbine increases in speed as it passes the turbine. Since this is physically impossible, this region must be excluded from use in the algorithm. Based on experimental data generated by Tidepod[5] it can be shown the smallest wake lengths occur when the depth/disc ratio is 0.25 and thereby the most preferable velocity deficits. This value of 0.25 was used in creating the formula this algorithm is using and any values below 0.25 can be considered unphysical and thereby excluded. In the interest of experimentation, no upper limit will be placed on the value although it is expected to converge to 0.25. This does however still leave an unphysical region open to mutation. While the wake spacing could be limited so it is not possible to enter this range, instead a constraint will be placed on the velocity deficit. The yield point represents the value at which the formula is no longer representative of the true behaviour of the design under these conditions. In this case it was chosen to be set at 0.1 since data provided by Tidepod suggests a velocity deficit of 0.1 is a viable value.

This approach was considered the best due to the shape of the unphysical region. At a specific wake spacing it is possible for the entire range of the depth/disc ratio to lie within the unphysical region and vice-versa. Therefore, arbitrarily restricting the gene variables so a value could not enter the unphysical region would either remove the whole range or otherwise remove potentially viable places on the front. For a similar reason, setting the industry standards as the limits would also not remove the problem and would only serve to prevent the analysis of the entire test space. On top of this, by adding a constraint to be evaluated at runtime less values will have to be manipulated prior to running making the algorithm easier to use.

5 – Results

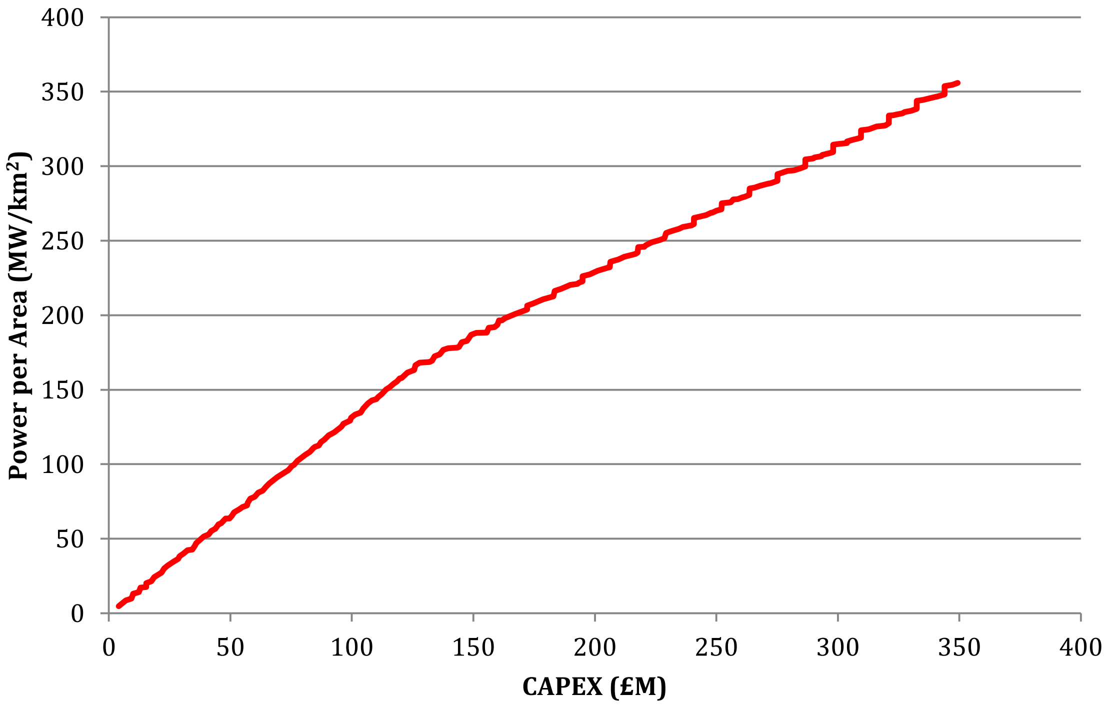

Figure 11: Pareto front of revenue against capital expenditure for a tidal generator array with velocity deficit held above 0.1

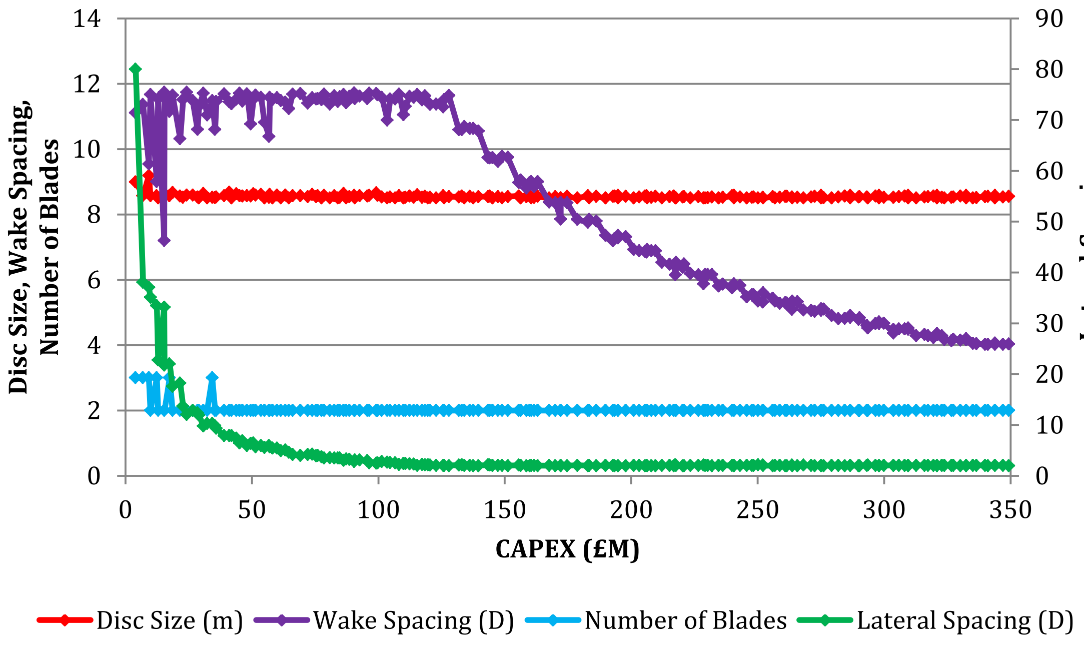

Figure 12: Variable map of Pareto front for a tidal generator array marked against capital expenditure of design with velocity deficit held above 0.1

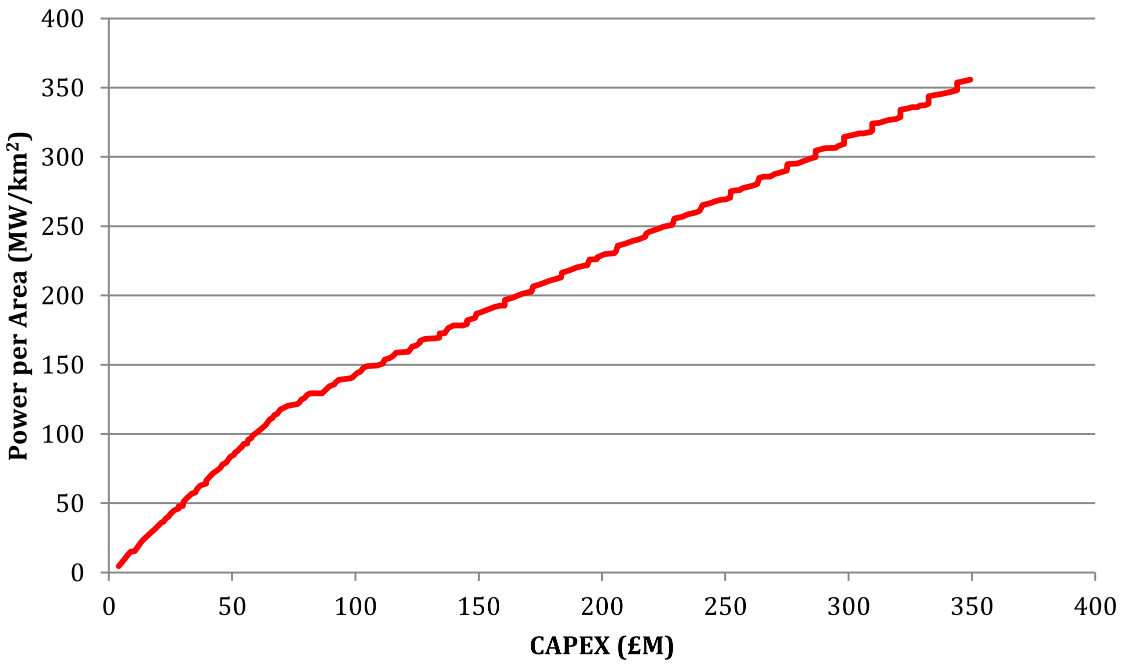

Figure 13: Pareto front of revenue against capital expenditure for a tidal generator array with velocity deficit held above 0.0

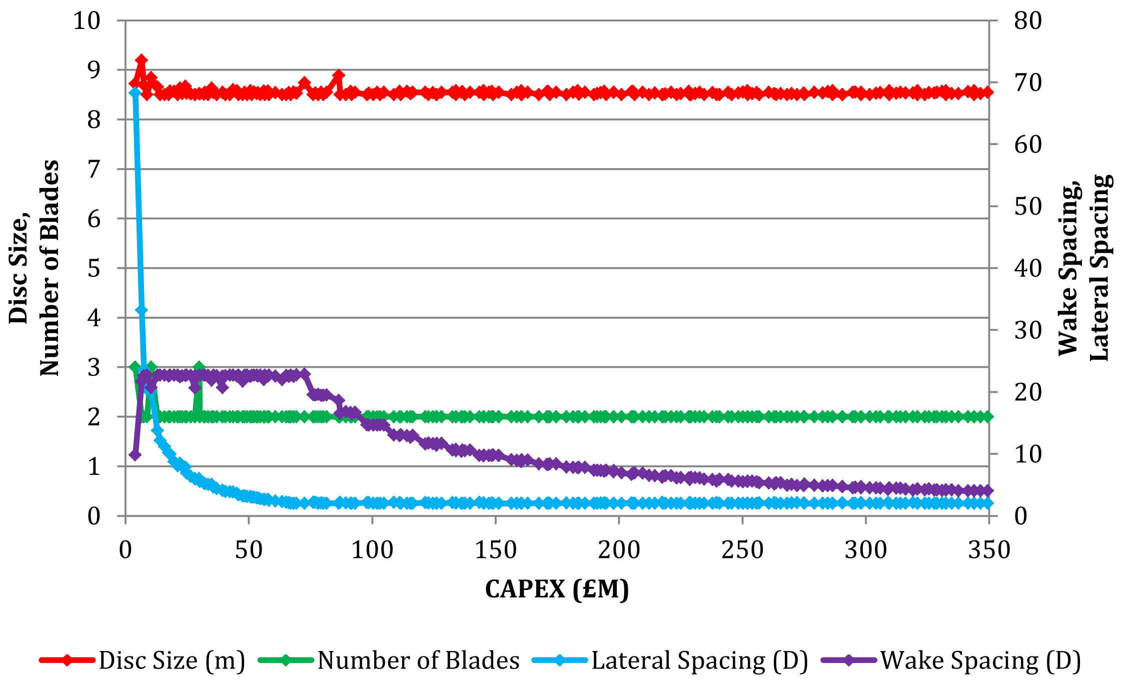

Figure 14: Variable map of Pareto front for a tidal generator array marked against capital expenditure of design with velocity deficit held above 0.0

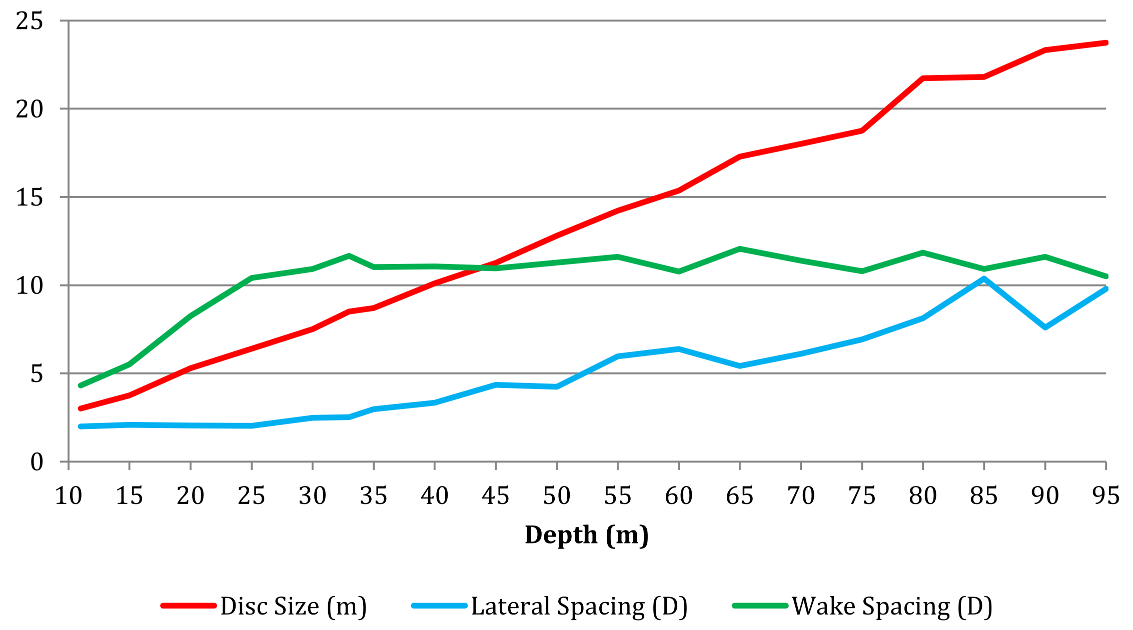

Figure 15: Graph of disc size, lateral spacing and wake spacing variation with depth at £100M CAPEX with velocity deficit limited to 0.1

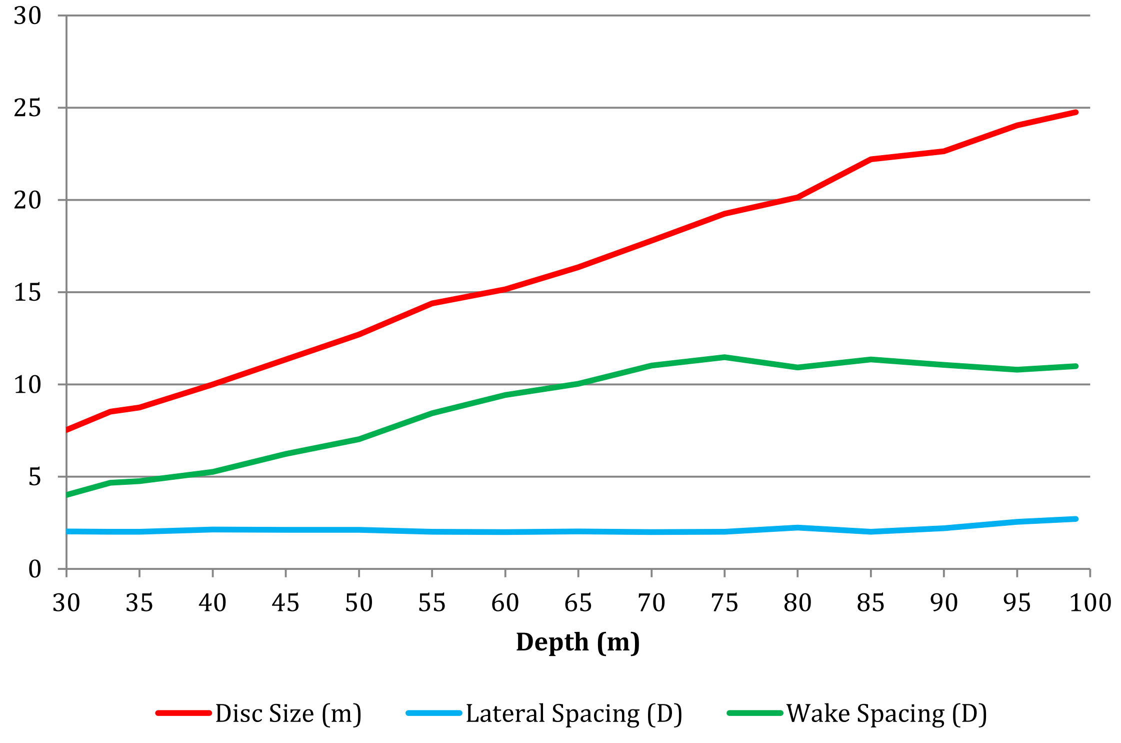

Figure 16: Graph of disc size, lateral spacing and wake spacing variation with depth at £300M CAPEX with velocity deficit limited to 0.1

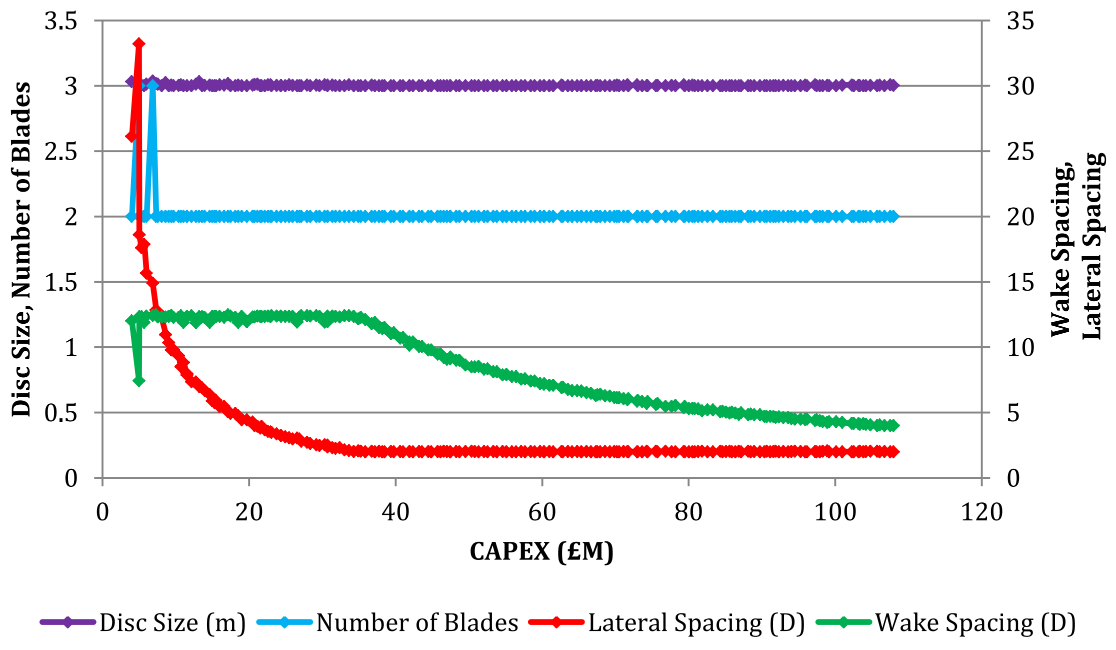

Figure 17: Variable graph of tidal generator array at minimum depth (11m) with velocity deficit limited to 0.1

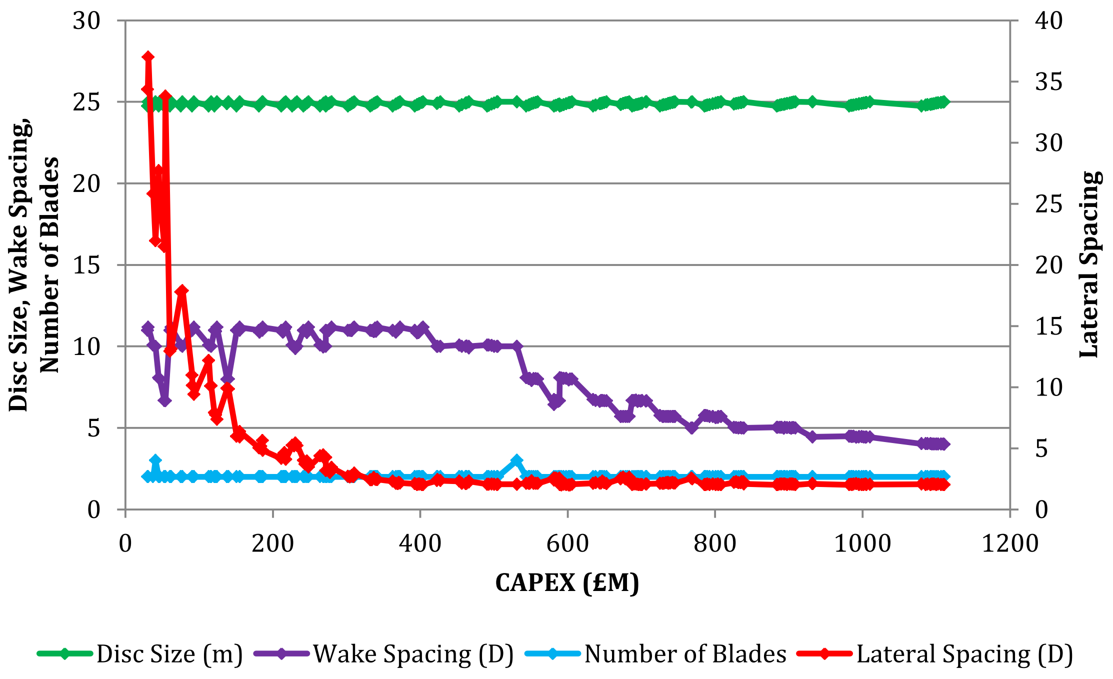

Figure 18: Variable graph of tidal generator array at marginally below maximum depth (99m) with velocity deficit limited to 0.1

6 – Analysis

Due to the low number of input variables along with the relative simplicity of the calculation, the algorithm was able to complete a run of 20,000 generations within an hour but had converged to a set of solutions suitable to draw conclusions from after 10 minutes. In this sense, the algorithm was able to find an optimum solution much faster than a human counterpart. However, in a lot of cases the results presented needed some refinement before put into practise. This was due to a number of reasons. Firstly, due to the fact the variables were described as floating point numbers the solutions were rarely close to a whole number which could cause problems during production and installation. Each variable has a practical tolerance associated with it: disc size is limited by the precision of the manufacturing and the turbine spacing is limited by the precision of the installation which is likely to vary on a per-turbine basis. Because of this, the algorithm cannot account for tolerance and will produce a set of variables that must later be rounded to a reasonable degree of precision.

The alterations to the velocity deficit formula and its use within the algorithm have been able to provide a useful set of results. The effect of this constraint can be seen within the results shown in the section above. When observing the Pareto front of the problem there appears to be a change in the trend at around the capital expenditure cost of £130M. A change like this suggests a limit has been hit on one side of this point within the gene variables preventing the true trend from being displayed. By looking at the variable graph in figure 12 it can be seen that at the point the trend changes in the Pareto front, a change occurs in the trend of the wake spacing. Prior to the point it appears locked to a value of around 11.5 but beyond that it changes its trend to a parabola levelling out to a value around 4. Further analysis of the data shows the results on the left are presenting solutions which are limited to a velocity deficit of 0.1 whereas the solutions to the right are not capable of reaching the limit and can therefore be considered optimum for their range. The factor to consider is that all the solutions shown here will work as intended since they remain in the model. The change in trend suggests there is the possibility for alternate solutions which at this stage would require an expansion of the model to use a design within this range.

Another set of results was generated using a constraint to limit the velocity deficit to 0 instead of 0.1. This represents the absolute minimum range the variable could be valid for since any value lower would represent the flow speeding up as a result of the turbines interference. The reason this was done is to see the effect this has on the set of data produced and how the position of the point at which the trend changes moves. It can be seen in figure 13 that there is a change in the trend of the Pareto front at the CAPEX of around £70M. This value is lower than that of the £130M point noted in figure 11 and indicates that the range in which velocity deficit formula is held to be correct in directly affects the Pareto front. The effect of this is that the values above this yield point produce two different values based on the limit of this formula. Therefore no region of the Pareto front is immune to the accuracy of this formula. Considering figure 14, there is a plateau in the wake spacing on configurations below £70M. The trend observed past this point is the convergence to the minimum value in an exponential manner which would be expected to continue in reverse into this plateau region. The plateau represents that the solutions that may have been generated following the previously described trend had negative velocity deficits and as such were not able to be displayed. The plateau instead represents a set of configurations that manage to have the equivalent CAPEX while keeping the velocity deficit coefficient above zero. Were more research done into the exact behaviour of the velocity deficit coefficient throughout the entire test space, it would probably be found to converge to a maximum as costs approach £0M and converge to a minimum as costs increase. While assumptions can be made to the shape of this trend, they would require experimental data to back the up which is beyond the scope of this project.

The values of the lateral spacing present an interesting trend. At low capital expenditure costs, the value of the lateral spacing is high and appears to increase towards the limit of a single turbine per row. The trend of the wake spacing appears to do something similar to the point it is limited by the model being used as discussed earlier. This is an expected result since the minimal cost would be a single turbine within the site. It is worth noting designs of less that £4M in cost were omitted from these results by a constraint since they were shown to diverge greatly in their results in the previous result set and as such were preventing a general trend from being calculated for the rest of the test space. Since the value of the wake spacing is locked, the cost is increased by decreasing the spacing between the columns of turbines. Although the trend appears to have an exponential component, the result corresponds to a linear increase in the number of turbines. This decrease in spacing converges towards it lower bound before the wake spacing of the turbines is allowed to decrease per the allowance of the model. It is likely that these variables would interact in a specific manner to maximise the number of turbines spaced laterally before adding another row of turbines in the wake. The presents a problem in the implementation of the algorithm since the formula calculating the effects of the wake does not take into consideration the close spacing of turbines in the lateral direction meaning there is no penalty to designs with closely packed rows where there should be. Also, due to the inflexibility of the data being specified to GANetXL, it is not possible to have different numbers of turbines in each row to capitalize on the close packing on the first row producing better results.

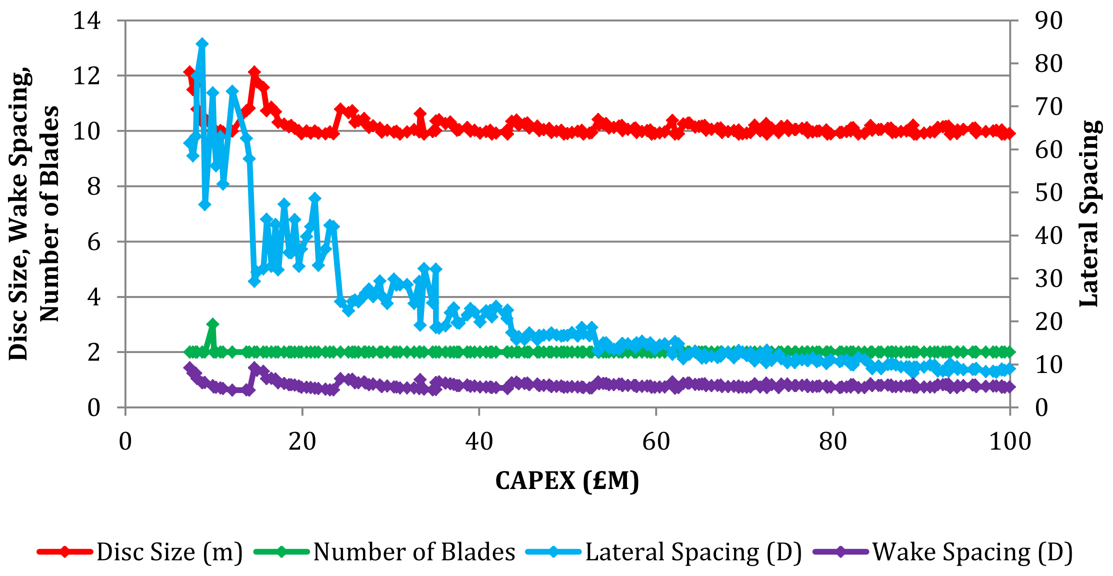

Figure 19: Variable graph of Pareto front for a tidal generator array system under several constraints

The graph above in figure 19 shows the trend expected to occur between the variables at low capital expenditure costs. This graph was generated by altering the constraint dictating the minimum allowed velocity deficit to be artificially high so as to allow both the lateral spacing and the wake spacing to interact together without altering the model. These constraints place a higher significance on the contribution of the array layout as opposed to the turbine characteristics and serves only to check if the relationship exists despite being negligible under regular constraints. It can be seen the lateral spacing has a general trend of decreasing exponentially which corresponds to a linear increase in the number of turbines in a given row. It can also be seen that the wake spacing shows a similar trend within a much more confined region. This indicates at a certain point it becomes more cost efficient to add more rows which are more sparsely packed and then increase the packing from there. The degree of this packing also appears to have an effect on the size of the turbine that is suggested since they are directly linked. Were the model improved so as to better account for the velocity deficit in closely packed arrays, trends such as those presented in figure 19 would also appear. Being able to modify the constraints in this manner is a major strength of using genetic algorithms for this problem.

The number of blades variable remains effectively constant throughout the range indicating that a two bladed turbine is ideal. The reasoning behind this is that the effect of more blades was negligible compared to the size of the turbine and the spacing with regard to the velocity deficit in the wake. Because of this the only factor affecting the fitness of solutions with more blades was the cost of more blades in terms of volume causing a minimisation of blades to be preferable. There are a few solutions suggesting three blade designs in various runs of the algorithm but they do not appear to represent a trend. As such if a configuration with 3 blades is selected, the effect of reducing the blade number should be considered.

The value specifying the size of the turbine also appears to be locked at a value of around 8.5m. While it is likely that a manual trial and error approach would be able to identify a turbine size such as this to be ideal when other variables are fixed, it would be unable to identify the general trend. Since the genetic algorithm is able to test the entire region of results, generalised trends as the results can be gathered with confidence. On top of this, the result is specific to use within a genetic algorithm since the trend is only representative of the Pareto front and not the entire result space. Without an exhaustive approach it is impossible to say with certainty that a point lies on the Pareto front and as such a result like this cannot be generated by standard methods. There is an indication within figure 19 that the optimal value of the turbine size is affected when the ratio of the lateral and wake spacing is changed. When dealing with solutions presented as a Pareto front it can be hard to discern if fluctuations in results are due to random mutation per the algorithm or are representative. In this case the value of the disc size was not locked but was changing slightly with the array spacing ratio. The only solutions to this are to allow the algorithm to run for longer to allow the results to converge towards the true Pareto front, minimising the error or to change the configuration so results are focussed on discerning the nature of the variable such as in figure 19.

One of the factors that are first specified when the design for a tidal generator array is planned is the depth of the site. The depth of the site can vary greatly depending on the location of the array. It is also likely that the height will vary to a degree on the site but this consideration is ignored for the sake of this project. As mentioned earlier, the monopile foundation is able to be any size required so that the turbine blades are as high as possible in the water stream but leaving 8m of space above them to maximise the speed of the incoming flow. However, although there may be plenty of space in deeper waters for larger turbines, there is a limit on the size of 25m diameter. This limit is based on the availability of existing tidal generators since in practical terms; the turbines making up an array would be bought from a company and not made specifically. As such, the availability dictates the upper limit. The exact height where this maximum size turbine can be used leaving only the 8m required above it is 33m depth which is the value used for all results discussed prior.

Figures 15 and 16 show the characteristics of tidal generator arrays of specific costs at varying depths. It is worth noting the blade number variable has been left out for clarity since it has been found to be equal to 2 in most cases. As the depth of the site increases, the height of the monopile required to hold it in the desired position increases and places a higher significance on the contribution of individual turbines to the yield since it costs more to produce each turbine. Based on the depth alone, the model has two limits which are presented in figures 17 and 18. The lower limit is the point where the turbine can be no smaller and has only 8m of space between the top of its swept area and the surface of the water. This case is rather simplified since the variable of the disk size is effectively locked at the value of 3m leaving only the spacing and number of blades variables open to the algorithm. The other extreme occurs at 100m depth (note figure 18 uses a depth of 99m since the model does not work at a depth of 100m). At this depth and below, the ratio of disc size to depth cannot reach a range where it is above 0.25 and thus is penalised heavily by the constraint holding it in the model discussed earlier. All results below this depth must be considered unaccounted for by the model. Figure 18 shows that at higher CAPEX values, the data begins to clump together as discrete ranges are changed between. This is indicative of the limitations upon the model. This is an example of a case where there are no viable solutions in the model at specific CAPEX costs. Although these results are reaching the limits of the model, the results presented are viable. Calculating these results manually would be very time consuming since much of the range is not possible. This shows the strength of a genetic algorithm when dealing with heavily constrained problems.

7 – Conclusion

This report is focussed on the implementation of these problems within the GANetXL plugin and the quality of the generated results. Problems encountered during this project can therefore be separated into three categories from which conclusions can be drawn. Some problems encountered represent limitations within GANetXL and its implementation within Microsoft Excel. Other problems arose due to the implementation of the problem and the model used to simulate it. Finally, some problems can be attributed to the nature of genetic algorithms as a method for solving complex mathematical problems.

Throughout this project limitations were in place by GANetXL, namely the limitation to 30 genes, 10,000 generations and population maximum size of 100. Because of this, the scale of several of the problems had to be limited. This was most noticeable during the Fourier synthesis problem which utilised all of the genes available but easily could have used many times more. The population size was reasonably high but the limit placed a higher significance on the contributions on mutations. Considering the first test case using standard test functions, a higher population size would have allowed the solutions to converge much sooner since more points in the solution space could be considered at once. In this way the population size is linked to the maximum number of generations when considering the quality of the solution possible to be generated. Throughout this project when generating final results the entire run of 10,000 generations was used. While the solutions found were to a suitable degree of accuracy, more generations would have meant a more accurate result and can therefore be considered a limitation.

The most major problem with GANetXL and its use of Excel was found during the running of the Fourier Synthesis case. This problem used all the genes available and mapped a complex function across thousands of test points. It was this set of test points which showed the limitation of using Excel as a basis of a genetic algorithm. Based on observations made while this test case was running, certain assumptions can be made about the operation of GANetXL. For each genome in every generation of the algorithm the set of genes associated with it were input into the problem worksheet from a separate storage worksheet. At this point the problem worksheet was updated which in this case caused thousands of cells containing sinusoidal formulas to be updated. These formulas are typically quite taxing on the processor and a slowdown was even observed during the first test case in which only one test point was used. Once the sheet had updated, the fitness formula and constraint formula had their values compiled and stored, prepared for the next genome with the occasional update displayed to the screen. On a full size run this would amount to 1,000,000 updates with minor subtractions for elitism levels. Based on the time taken for a single generation to run it can be estimated a full run of this solution would take a few days which is far beyond the time used for the other solutions. This slowdown can be attributed mainly to the infrastructure of Excel since it is designed for manual input as opposed to other programs such as MATLAB more suited for the manipulation of data on this scale. When dealing with a problem such as this the management of data is very important and trade-offs between time and accuracy must be considered. In this case, the population size was lowered and the algorithm was halted once a suitable level of accuracy had been gained which in the final case still took several hours.

The second factor to consider is the effectiveness of the models being used throughout this project. Certain differences occur between calculation methods for problems solved manually and solved computationally. A primary concern is designing a robust model and henceforth keeping any proposed designs within the model. When manipulating a problem manually this is done automatically as the operator intuitively does not test cases that are obviously not possible and they will quickly notice any unexpected behaviour in the model. It is considered a strength of genetic algorithms in that they can examine the entire range of solutions within defined boundaries and if the line between possible and impossible solutions is correctly drawn, unexpected solutions can be shown to be optimal. Within the first test case and third, the variable boundaries were specified by the problem. Within the Fourier synthesis test case and the tidal generator array problem the limits were placed such that any physical value could be chosen. This meant the Fourier synthesis problem was difficult to manage due to the relationship between amplitude and frequency ideally requiring no limits; but being forced to exist within fixed limits. The limits specified during the generator array calculation caused problems in regard to the solutions falling out of the model. Both the pipe/pump problem and the generator array were constrained to keep them within the model. The pipe/pump system checked for a valid operating point and checked the solution met requirements of the problem. The generator array problem had to be much more greatly constrained. The model used was considered to only be accurate at representing solutions within a common range since it originated from a manually controlled calculation. Even though the common range was interpreted to limits of the genes for use with the algorithm the range still included unwanted results. Certain configurations were possible within this range that would not be chosen by a human operator since they are obviously wrong but the algorithm needed a way to identify this. By monitoring the velocity deficit and various quantities throughout the run the results generated can be considered accurate but the benefit of examining the entire solution space was lost. This represents a key problem with the conversion of a manual calculation to a computational program.

In some cases the model was limited by the nature of Excel and the problems implementation within it. The Fourier synthesis fitness function relied on the difference between two data sets representing curves. Since the data was spread out over thousands of cells an integrating function such as Simpson’s rule was tricky and time consuming to implement. A function such as this would also greatly increase computation time. The chosen fitness function used a sum of per-datum difference which allowed a result to be gained to an acceptable degree of accuracy. When dealing with the pipe/pump system problem the fitness function relied on the accurate finding of the operating point of the system. The implementation used for this project used liner interpolation between the points either side of the crossover. Due to the nature of Microsoft Excel this implementation was overly complicated and prone to error since it requires a lot of conditional statements to determine where the exact crossover lies. Although a primarily human concern the implementation of problems for use with a genetic algorithm typically complicates them and makes them more prone to error. While suitable review can present this it should be considered a weakness to genetic algorithms.

The final source of problems would lie in the method of genetic algorithms in general. Generally speaking, common engineering problems should be capable of manipulation by a genetic algorithm if a method exists to solve them manually. Deciding on a point where a genetic algorithm is incapable of solving a problem is directly related to the system it is running on. Factors such as float precision and computation power and memory storage can define the upper limits of a genetic algorithms capability. While in theory a genetic algorithm could automate the design of a tidal generator array by analysing results from a CFD program the sheer amount of data and computation time is beyond that available on commonplace systems. As genetic algorithm software improves and models become more complex, the problems associated with them will shift from the implementation to the infrastructure requiring more and more powerful computers to operate them.

In terms of establishing how useful genetic algorithms can be to the design of tidal generator arrays and associated engineering problems, the results are very successful. Consider if the solutions were not optimised, the algorithm was able to test 20,000,000 combinations of the four variables used within the period of an hour making it dramatically faster than a human when a brute-force approach is applied. A human analysing this problem would in practise be using an educated approach of trial and error, improving the solution periodically by noting patterns between the changing of variables and the qualities of the end result. This human approach becomes fundamentally flawed as the range of possible solutions becomes more complex, offering multiple minimum regions. In order to find all these minima, a standard approach would be reduced to a brute force method to map the entire result space reducing the effectiveness of the method greatly. It can however be seen that the solutions generated by the algorithm are not random and are able to present a set of data useful to the practical design of a tidal generator array. The results were able to converge to a solution set displaying expected trends such as the relationship between lateral and wake spacing at low capital expenditure costs shown in figure 19. Also, none of the variables appeared random showing they were successfully carried through the model and were considered in its optimisation. This same manner of success was shown across all the test cases as well with clear and useful results despite encountering problems during their setup.

In the design of tidal generator arrays as with a number of industry scale engineering problems, it is likely that there will be new research and developments in the field which will lead to alterations to the design process. It I also likely that a re-evaluation of the design is likely to be costly in terms of time and money so any method used to solve such problems should be considered on such criteria. First the expansibility of a genetic algorithm is a key benefit to it. At any point the model that a genetic algorithm uses can be built upon to provide a more accurate result. Beyond this, the cost of finding a solution hinges on the time taken for the algorithm to complete a given number of generations or to converge to an accepted degree of accuracy. It has been shown throughout these tests that a common timescale for a problem such as the tidal generator array problem to present a set of potential solutions is around an hour which places it in a timescale conducive to its use in industry. Were to calculation time longer, say a week the process could be considered a tool similar to CFD analysis; reserved for use until the nature of problem is fully understood. As it stands, the timescale suggests genetic algorithms could be used as part of a design process and can be iterated upon throughout the life of a project to provide iteratively better results.

8 – Further Work

This report has dealt with a number of problems and their implementation within the GANetXL genetic algorithm solver. However, looking into alternate genetic algorithm solvers would allow for a possible expansion of the scale on which these algorithms were run. Some alternates possible include MATLAB plugins such as the global optimization toolbox[6], standalone solvers and custom-created solvers made for solving specific problems. The greater the control available over the exact operation of the GA solver, the better quality the solutions offer will be. I would be interested in testing the effects of varying types of constraints on the quality of a solution. Generalising the effect of a constraint such as in GANetXL makes their nature easier to understand but allowing constraints control to directly manage members of a population (i.e. remove a genome exceeding a constraint boundary to be replaced by a clone of a better genome) could improve the efficiency of a given algorithm.

The fitness functions and the related methods of calculation of the problems presented in this report are accurate at representing the goal of the problem. However, expanding on the complexity of the fitness functions used would allow for a much higher quality solution. The generator array problem was simulated with only four genes which allow much room for expansion. The functions could be expanded to allow different spacing to be used between each row due to the cumulative effect of velocity deficits left in the wake of the various rows of turbines. The fixed variables describing the site could be expanded from a simplistic representation of the site as a cuboid with a linear flow could be replaced by a map of heights and corresponding velocities from survey data. In order to properly use this data, the turbines would have to be represented individually which would be much more memory intensive and would require significantly more genes. However, an approach like this would allow for arrays beyond that of a linear pattern to be tested. This project was primarily focussed on the use of genetic algorithms but beyond this, a study of fluid dynamics would be pertinent to expanding the model. A study of velocity deficits and the nature of a wake at small spacing both in lateral and perpendicular directions would help in expanding the model. Links with industry could also help in developing costing functions to take account of all the factors associated with the design such as legislature, taxes, employment and other human aspects. This design problem has a large potential in terms of expansibility if the limitations of GANetXL and that of the model can be overcome.

9 – References

[1] Savić, D. A., Bicik, J., & Morley, M. S. (2011). A DSS Generator for Multiobjective Optimisation of Spreadsheet-Based Models. Environmental Modelling and Software, 26(5), 551-561.

[2] Koza, J. R. (1992). *Genetic Programming: On the programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. ISBN: 0-262-11170-5. Ch. 6.

[3] Mishra, Sudhanshu (2006). *Some new test functions for global optimization and performance of repulsive particle swarm method.* Available at: http://mpra.ub.uni-muenchen.de/2718/.

[4] Douglas, J. F., Gasiorek, J. M., Swaffield, J. A., & Jack, L. B. (2011). *Fluid Mechanics Sixth Edition*. Pearson Education Limited. Harlow. ISBN: 978-0-273-71772-0.

[5]Tidepod Ltd. Newquay, Cornwall. Source of unpublished experimental data and mathematical basis for tidal array model.

[6] The MathWorks Inc.. (2014). *Global Optimization Toolbox.* Available: http://www.mathworks.co.uk/products/global-optimization/. Last accessed Apr 2014.

[7] Tocardo. (2014). *T100.* Available: http://www.tocardo.com/products_and_services/test_product.html. Last accessed Apr 2014.

[8] Bahaj, A.S. and Myers, L.E. (2013) Shaping array design of marine current energy converters through scaled experimental analysis. *Energy*, 59, 83-94.

[9] Chamorro, L., Arndt, R. Sotiropoulos, F. (2011). *Turbulent Flow Properties Around a Staggered Wind Farm*. Boundary-Layer Meteorology, Dec2011, Vol. 141 Issue 3, p349-367. 19p.

[10] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. First MIT Press. ISBN: 0-262-63185-7.

[11] Bevelacqua, P. (2010). *Fourier Series.* Available: http://www.thefouriertransform.com/series/fourier.php. Last accessed Apr 2010.

Appendix A

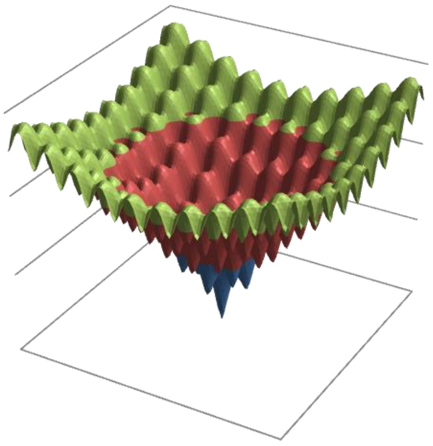

Figure 20: Graphic plot of Ackley's function

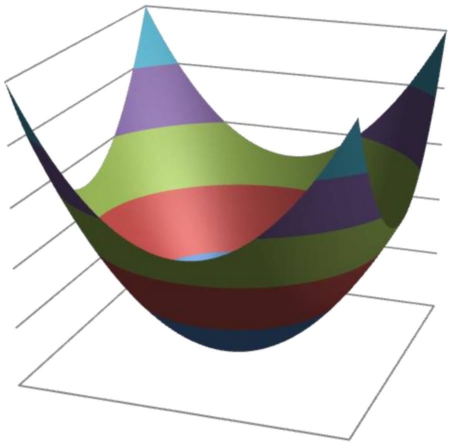

Figure 22: Graphic plot of a sphere function

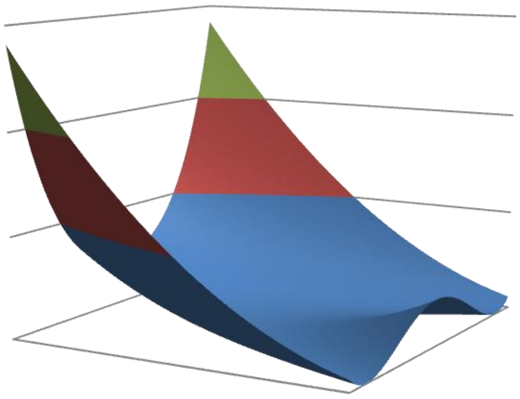

Figure 21: Graphic plot of the Rosenbrock function

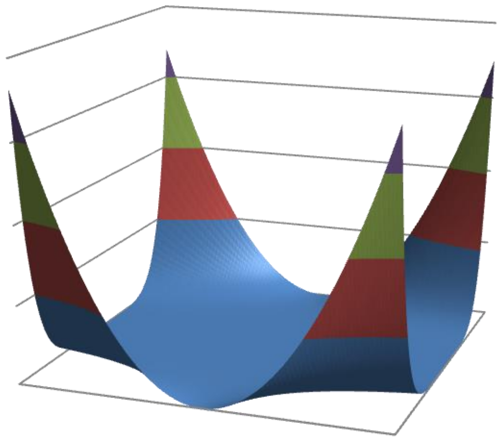

Figure 23: Graphic plot of Beale's function

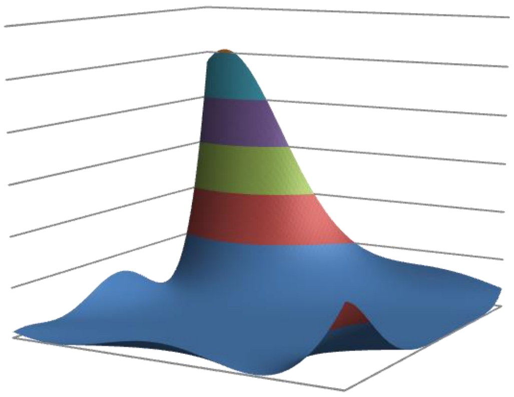

Figure 24: Graphic plot of the Goldstein-Price function

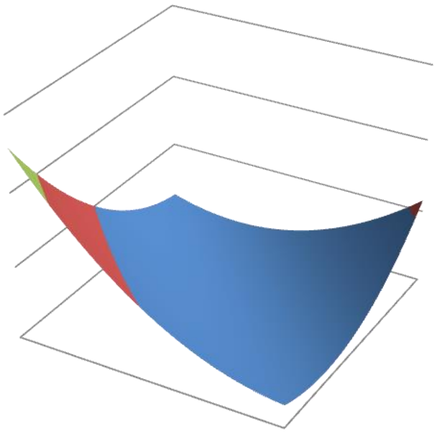

Figure 24: Graphic plot of the Booth function

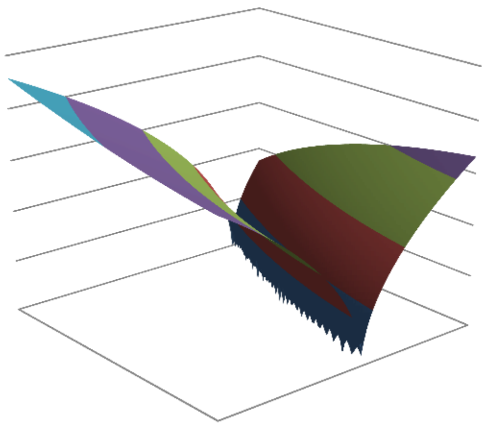

Figure 25: Graphic plot of the Bukin function N.6

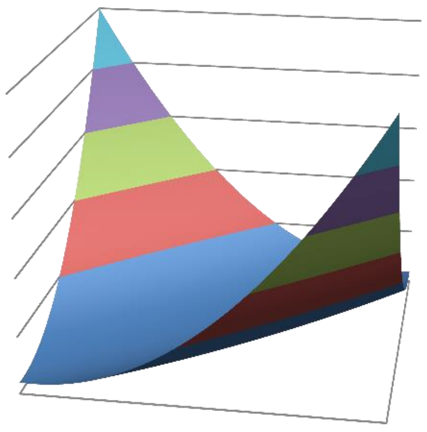

Figure 277: Graphic plot of the Matyas function

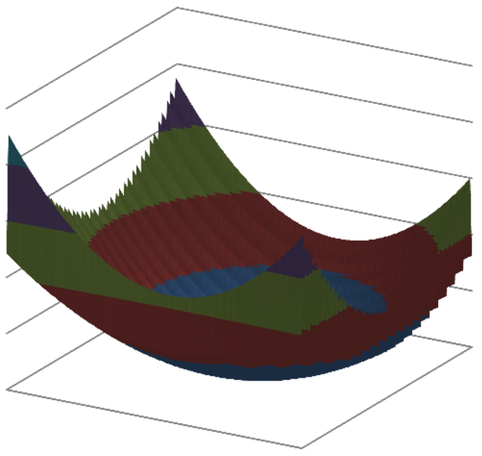

Figure 268: Graphic plot of the Levi function N.13

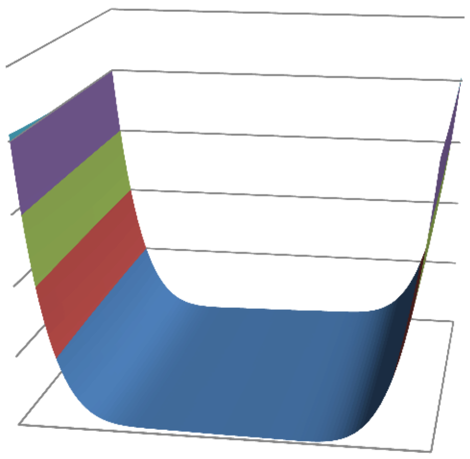

Figure 29: Graphic plot of the Three-hump Camel function

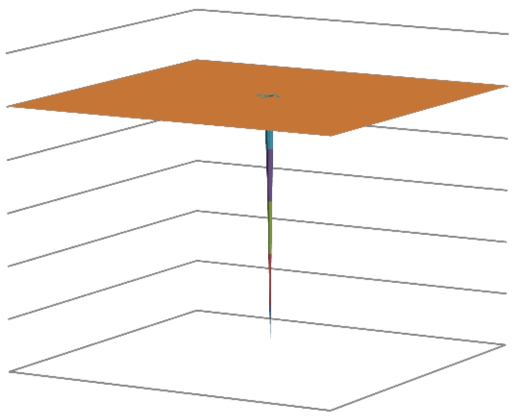

Figure 30: Graphic plot of the Easom function

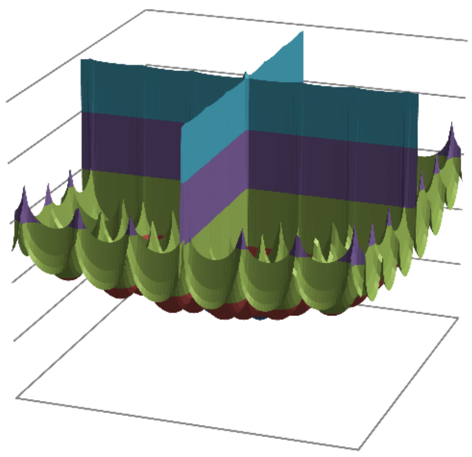

Figure 28: Graphic plot of the Cross-in-tray function

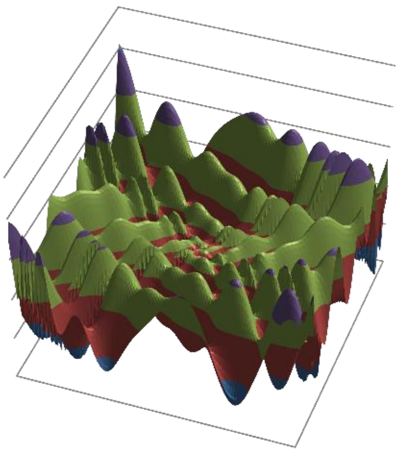

Figure 29: Graphic plot of the Eggholder function

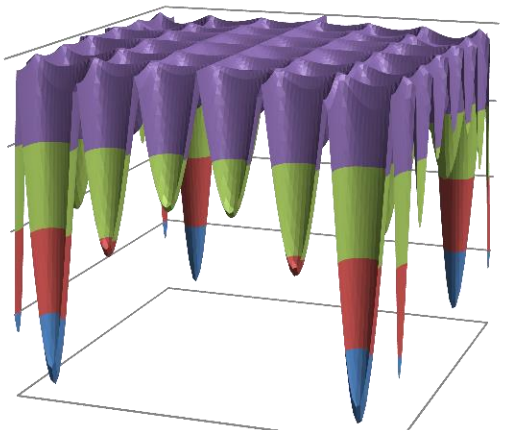

Figure 323: Graphic plot of the Holder Table function

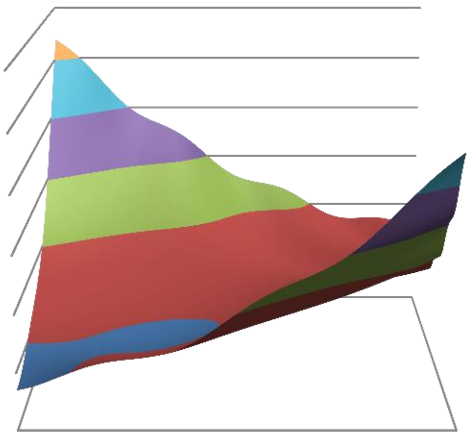

Figure 304: Graphic plot of the McCormick function

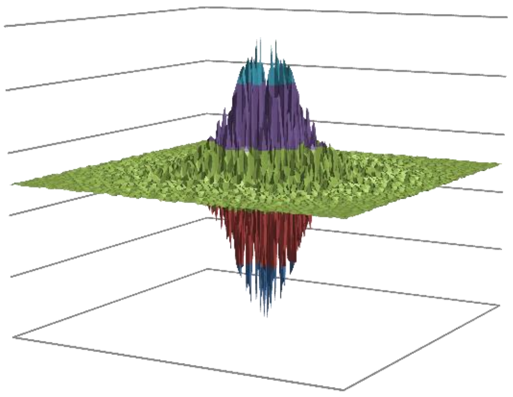

Figure 335: Graphic plot of the Schaffer function N.2

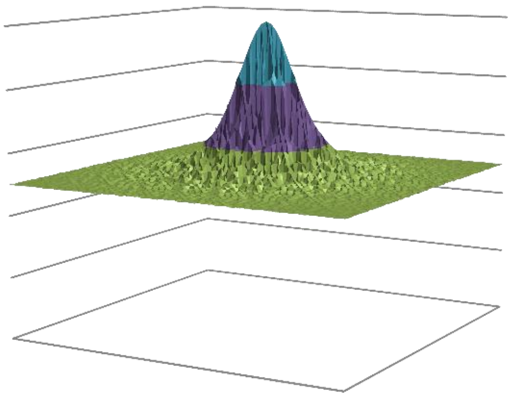

Figure 316: Graphic plot of the Schaffer function N.4

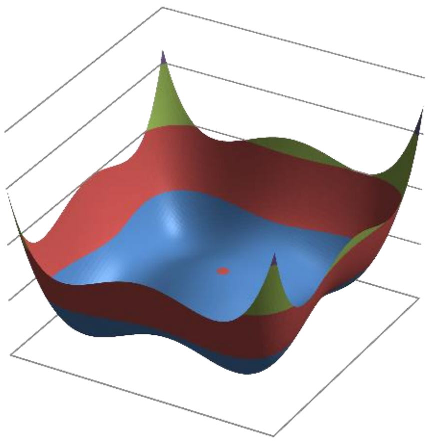

Figure 34: Graphic plot of the Styblinski-Tang function

Appendix B

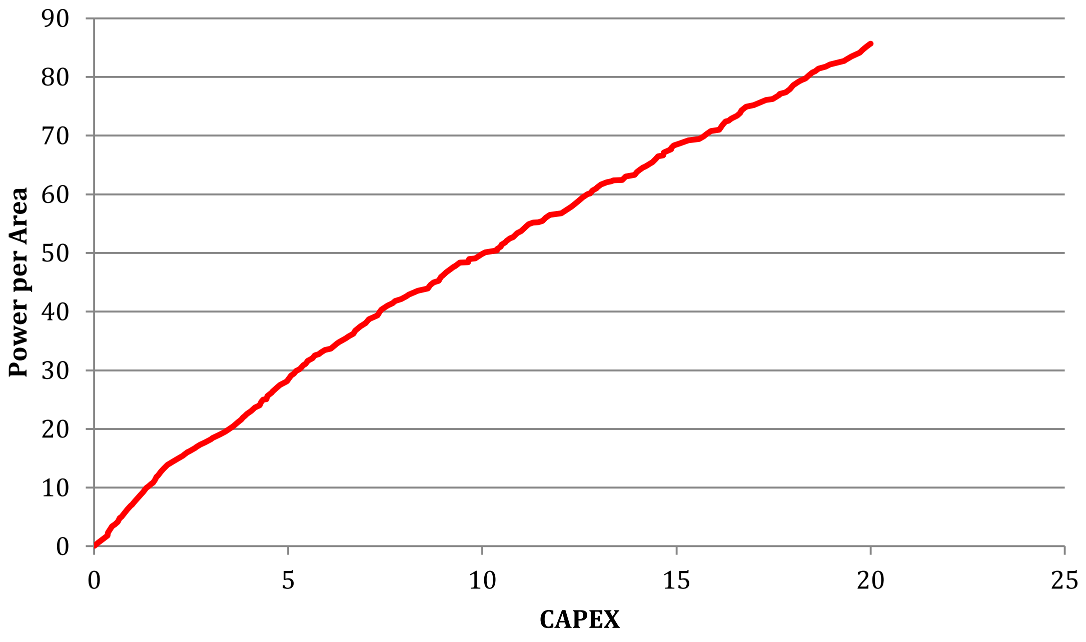

Figure 35: Pareto front for tidal generator array with simplified model charting power per area against CAPEX limited to lower CAPEX range

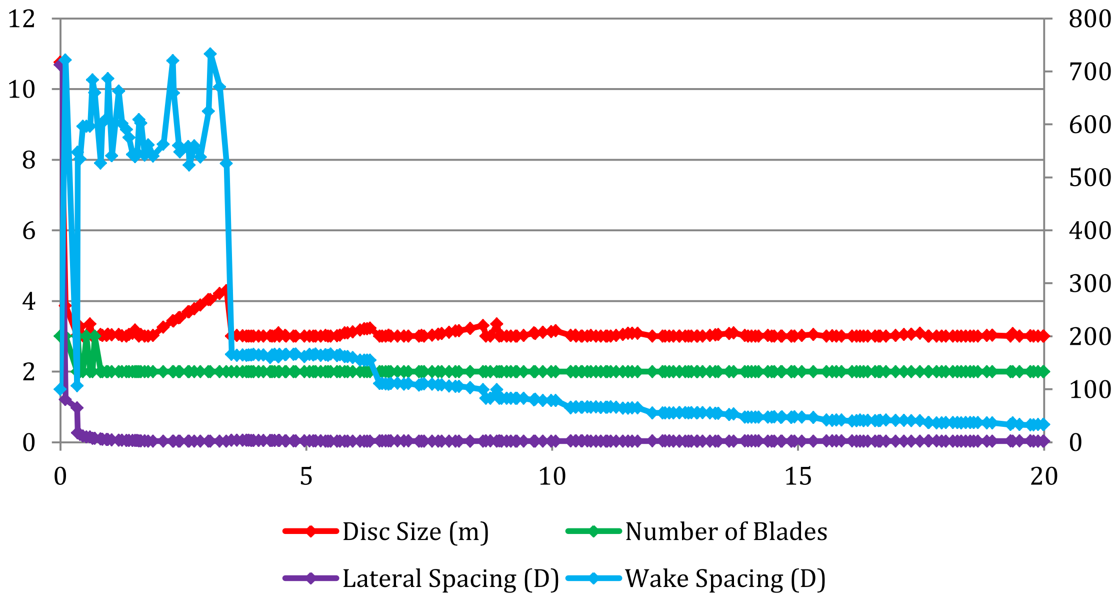

Figure 36: Variable map for tidal generator array Pareto front with a simplified model limited to lower CAPEX range

Appendix C - Preliminary Report