The Behaviour of Evolutionary Algorithms for the CFD-Driven Design Optimisation of Aerofoils

Author:
Hannah R Ditchburn

Supervisors:
Dr Ben Evans
Dr Sean Walton

Submitted to Swansea University in fulfilment of the requirements for the Degree of Master of Science by Research

Department of Aerospace Engineering,
Swansea University

September 2022

Abstract

This thesis utilises aerodynamic shape optimisation software AerOpt and FLITE2D, to explore the behaviour of three Evolutionary Algorithms, Differential Evolution (DE), Modified Cuckoo Search (MCS), and Particle Swarm Optimisation (PSO), to optimise a 2D nonsymmetric aerofoil, providing an evaluation of their aerodynamic optimising capabilities.

The aerofoil used in test cases is the NACA21120, where a variation of control node approaches are utilised to alter the aerofoil's geometry. In the first set of test cases, a control node is placed on the upper surface to allow the thickness to be altered, and in the second set of cases, six control nodes are arranged along the boundary of the aerofoil, to examine the overall shape change.

A mesh convergence study helped to determine the best mesh settings for the given problem. Each algorithm is tested in a subsonic, transonic, and supersonic flow regime to ensure the test cases fulfil the CFD aspect of the research. All flow regimes were treated as viscous with the relevant Reynolds number applied. To provide an analysis on how tuning the input parameters affects the algorithm's behaviour, the number of agents were inputted were varied from 10 to 50 to 99. The generations number was set to 99, and the fitness objective was to optimise for the lift-drag ratio (L/D), throughout all optimisations.

The first set of results (one control node) found that fitness improvements were largest in the transonic cases, increasing the L/D by an average percentage of 213%. The aerofoil's L/D at Mach 0.5 was improved by an average of 80%, and Mach 1.5 by 33%. Each algorithm showed a similar trend in which the control node was positioned at the final generation in the design space, this varied depending on the Mach number being optimised for, either resulting in an increase or decrease in the aerofoil thickness. Varying the number of agents inputted, had a more significant effect on MCS, whereas DE and PSO showed more consistent results regardless of the number of inputted agents. Generally, PSO displayed fastest convergence of all the agents, shortly followed by DE, followed by MCS.

The second set of results (six control nodes) were optimised for identical input parameters but for simplicity, at a single flow regime, Mach 1.5. Differing from the first set of results showing similar control node placement within the design space, the second set of results showed the algorithm's position some of the control nodes in different locations within the design space. Despite the similar fitness improvement values seen between DE and PSO, the final geometries were observed to be somewhat varied, where DE reduced the thickness of the trailing edge, but PSO increased it. MCS displayed similar geometry change to PSO but with more conservative control node movement.

Declarations

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed: [redacted]

Date: 04/05/23

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed: [redacted]

Date: 04/05/23

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed: [redacted]

Date: 04/05/23

The University's ethical procedures have been followed and, where appropriate, that ethical approval has been granted.

Signed: [redacted]

Date: 04/05/23

Keywords

Aerofoil, Agent, Computational Fluid Dynamics, Control Node, Evolutionary Algorithm, Fitness, Generation, High Performance Computing.

Nomenclature

Symbol/abbreviation	**Name/description**
L	Lift (Newtons, N)
C_L	Coefficient of Lift
D	Drag (Newtons, N)
C_D	Coefficient of Drag
f	Fitness
l	Chord length
M	Mach Number
T	Temperature (K)
R	Gas Constant (J/K-kmol)
γ	Adiabatic Ratio
c	Speed of Sound (m/s)
ρ	Density (kg/m^3)
μ	Dynamic Viscosity (kg/ms)
F	External forces (Navier-Stokes) (N)
I	Identity Tensor (Navier-Stokes)
u	Flow Velocity (Navier-Stokes) (m/s)
Re	Reynolds Number
EA	Evolutionary Algorithm
GA	Genetic Algorithm
DE	Differential Evolution
MCS	Modified Cuckoo Search
PSO	Particle Swarm Optimisation
CFD	Computational Fluid Dynamics
ASO	Aerodynamic Shape Optimisation
CN	Control Node
BL	Boundary Layer

Acknowledgments

The author would like to thank primary supervisor Dr Ben Evans, and second supervisor Dr Sean Walton, for the continuous support, guidance, and expertise provided throughout the project.

The author would also like to thank research peer Ben Smith, and RSE’s at SA2C, Dr Ed Bennett and Dr Michael Pei for the continuous support provided on computational challenges encountered throughout the project.

Swansea Academy of Advanced Computing (SA2C) [45] is a team of research software engineers (RSEs) dedicated to enhancing the use of HPC systems for research within the university. Collaboration with the team has enabled SA2C to provide software development support throughout this research project.

Table of Contents

1. Introduction

Despite promising results when implemented into the design optimisation processes, Evolutionary Algorithms (EAs) hold limited use within industry and academia. Improving efficiency is a focus for engineers globally and is rapidly becoming a necessity due to climate change. Reducing emissions across aircraft fleets is achieved at scale by efficiency increases, an area in which EA's show enormous potential. Research carried out at Brno University of Technology, showed that morphing wings, designed using EAs, have the potential to reduce fuel consumption in-flight by 44%. This was achieved by altering the aerofoils camber, thickness, and overall shape to determine the fuel consumption improvement [1]. With the development of robust optimisation studies using relevant software frameworks, such as AerOpt [2] embedded with the FLITE2D solver [3], developed at Swansea University, EA's optimising capabilities are demonstrated.

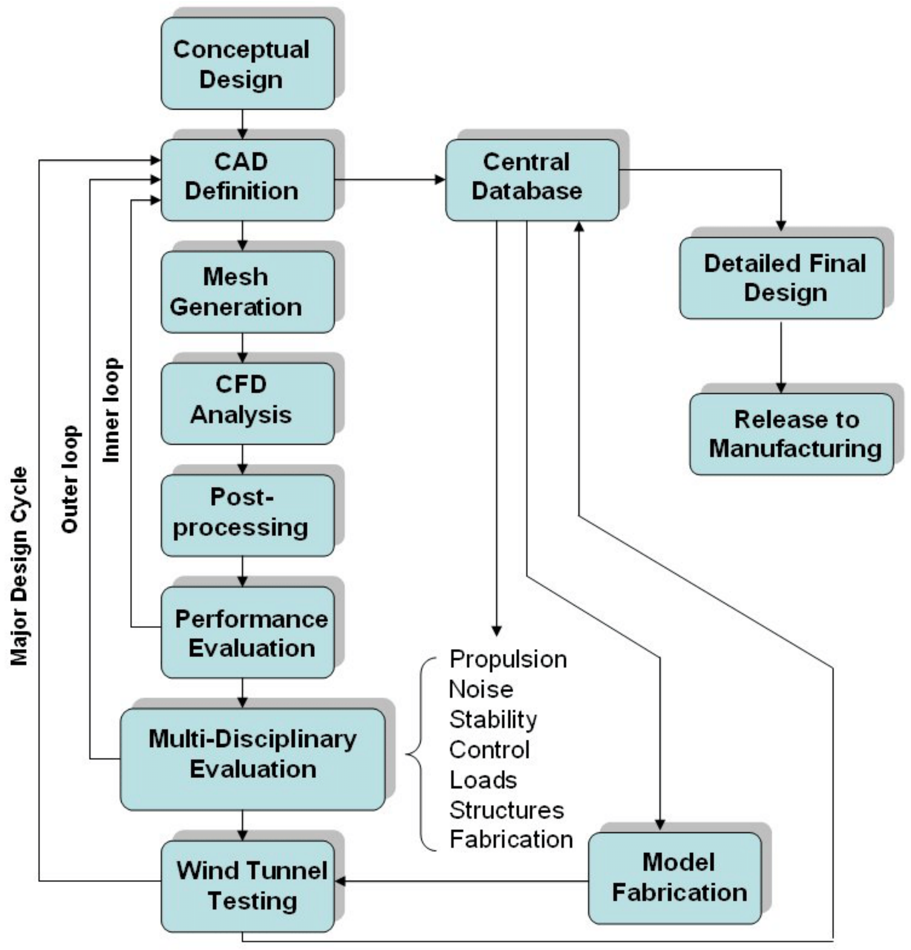


Figure 1 A Generalised multi-disciplinary aerospace design cycle [27]

The flow chart above shows a generalised summary of an aerospace engineering design process, where the middle section, from CAD development to the final design evaluation, is a section that an applied EA with CFD solver such as FLITE [3] is capable of streamlining.

1.1. Research Aims

This thesis aims to demonstrate how the variation of the input parameters can be used to control the direction and result of Evolutionary Algorithms. Research objectives include:

- Analyse the algorithm's behaviour and result based on the number of agents inputted.
- Examine the algorithm's behaviour and result based on the number of control nodes inputted.
- Develop the general usability and HPC connection in Swansea University's AerOpt software.
- Conclude and make recommendations for future work based on observations made in the input parameter analysis. This may consist of recommended number of agents optimal for the EA's operation, or an insight into number of degrees of freedom that each algorithm optimally operates at.

It is important to note, the EA's included in this thesis have been applied to a very specific optimisation problem within the scope of CFD. Conclusions and recommendations stated within this paper should be broadly interpreted, and results will vary depending on the application.

1.2. Optimisation

Optimisation finds the best solution for a given problem with regards to its resources. There are a range of optimisation methods utilised depending on the problem being applied to. In aerospace engineering, aerodynamic shape optimisation (ASO) has become an essential part of any robust aerodynamic design, and an essential part of designs seen in everyday life, such as, cars, trains, aircraft, wind turbines, pipes, and many more [4]. Developments in computational methods and resources has allowed for lengthy design processes to become faster and easier to test, attracting a range of academics to not only focus on solving the problem itself but analysing the process of the optimisation method.

Typically, an optimisation problem involves an objective function, its variables, and constraints. The variables and constraints defines the set-up of the problem, called the initial modelling, sometimes considered the most important step in optimisation. If the model is too simplistic, it may not be efficient in finding a solution to the problem, too complex, and it may compromise what the optimisation variables and function are able to solve. Once the model is developed, the chosen optimisation process can be applied, usually in combination with computational resources, to find a solution. There is no 'best' optimisation method, rather different techniques that can be utilised to suit the problem.

1.3. Gradient-based and Non Gradient-based Techniques

Gradient-based optimisation methods search the design space by identifying the gradient of the objective function at particular stationary points to find the direction of what it believes to be the optimal solution. If the objective function of a problem is stationary and gradient information is available, gradient based optimisation algorithms present a robust set of tools for solving a problem. Gradient-based methods such as, Newton's method [6], Quasi-Newton method [7], Levenberg Marquardt algorithm [8], and the conjugate direction method [9], are popular optimisation methods suited to continuous optimisation problems, and are sometimes favoured over evolutionary algorithms due to their robustness and computational efficiency [11][12]. However, in some problems, gradient information may not be available. Even if gradient information is available, it can be unreliable, therefore, dependent on the problem's characteristics, non-gradient methods can be used as effective optimisation tools instead [9].

Unlike gradient-based optimisation methods, non-gradient based methods do not require gradient information to find a solution. Instead, non-gradient methods use evaluations of the user-defined objective function. They are predominantly used when gradient information is not available. The majority of non-gradient optimisation methods focus on heuristic methods, including evolutionary algorithms. These methods find a global optimum, whereas a gradient-based method finds a solution locally, making either suited to the type of optimisation problem applied to and the desired outcome [13]. There are advantages and disadvantages to both gradient and non-gradient based methods. Since gradient-based methods find a solution locally, exploration of the design space can be limited, and these methods sometimes settle at a local optima, (an optimal solution within that area of gradients) which may not be the true optimum [11][28]. Also, these methods tend to be affected by noise sensitivity, making them not as well suited to solving aerodynamic design problems accurately [22].

1.4. Evolutionary & Genetic Algorithms

Evolutionary optimising techniques, or Metaheuristics in practice, are implemented through use of an EA. Genetic Algorithms (GAs) are considered a sub-class of EA's [14][15], along with other sub-classes of evolutionary algorithms such as genetic programming (Banzhaf et al.) [16], Evolution Strategies (Beyer and Schwefel) [17], PSO (Kennedy and Eberhart) [18] and many more.

EAs produce excellent results for design optimisation and can be used for a variety of applications [1][32][34]. DE [20] and MCS [21][22] are considered types of GA's, whereas PSO takes an iterative-type approach, where each iteration is defined by a new position in the design space. Generations are to signify offspring created at the next generation of via breeding. PSO does not include this nature of breeding, making it less a GA and rather its own form of EA.

EA’s use a non-gradient descent-based search method to determine the best candidates within a population, in an attempt of achieving an optimal solution. During this process, mutation, and recombination (using a crossover function) are applied to the candidates displaying the highest fitness regarding the defined optimisation function. The objective function is defined by the user and is the objective to be optimised. These candidates produce a candidate solution, or off-spring that tends to the target optimum, repeating for each iteration or generation. GA’s typically are the most widely used for optimisation problems and emphasise on the recombination of pairs, whereas genetic programming and evolution strategies tend to focus more on mutation [1].

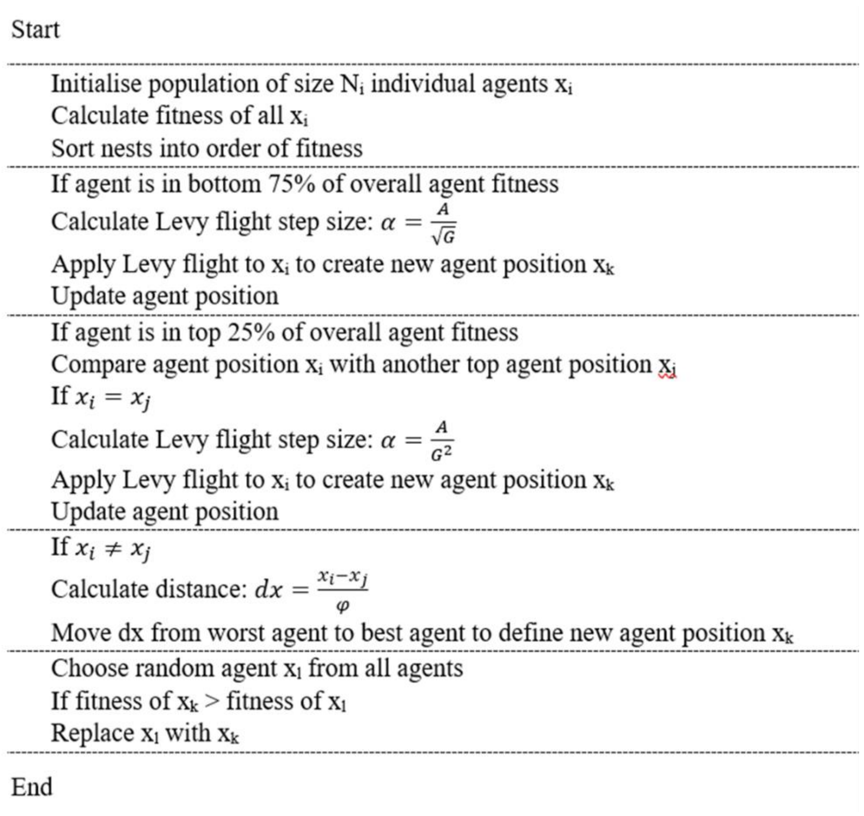
An EA generally begins with initialising a population size of N agents, these are points within user-set design constraints the algorithm uses to improve the design. The agents are expected to evolve in fitness through a set number of generations based on the objective function defined by the user. Each generation defines a new possible position for the agent to evolve from. Therefore, the more agents and generations included in the optimisation, the higher probability the algorithm will find a result in high fitness. With this comes the requirement of computer power, particularly since many GA’s are run in combination with additional solvers, like Computational Fluid Dynamics (CFD) solvers, making computing resources costly, and outputted results cumbersome for the user to interpret [30] (see section 1.7).

1.4.1.Cuckoo Search & Modified Cuckoo Search

The Cuckoo Search (CS) algorithm is a genetic algorithm developed by Yang and Deb [23][24]. The MCS algorithm developed by S. Walton et al [21], is an enhanced version of the CS algorithm, including the addition of agents in the top search can interact.

The CS algorithm aims to mimic the invasive reproductive behaviour of the Cuckoo bird. Cuckoos lay their eggs in a host birds’ nest, usually of a different species. To convince the species that the eggs are their own, these eggs must display similar characteristics to the host birds’ eggs. If the host bird were to discover a difference between its eggs and the Cuckoos’, it will destroy the contrasting eggs or even abandon the entire nest. To avoid this, Cuckoos’ eggs evolved to resemble the host birds’ eggs. This strategy forms the basis in which the CS and MCS algorithm are built on. The eggs are referred to as agents, and each generation produces a new egg or offspring.

A design space is set by the user and defines the space in which the agent can search for new positions. Within the algorithm process, an offspring is the equivalent of an agent taking a new position inside the design space. A summary of MCS for one generation can be seen in Figure 2.



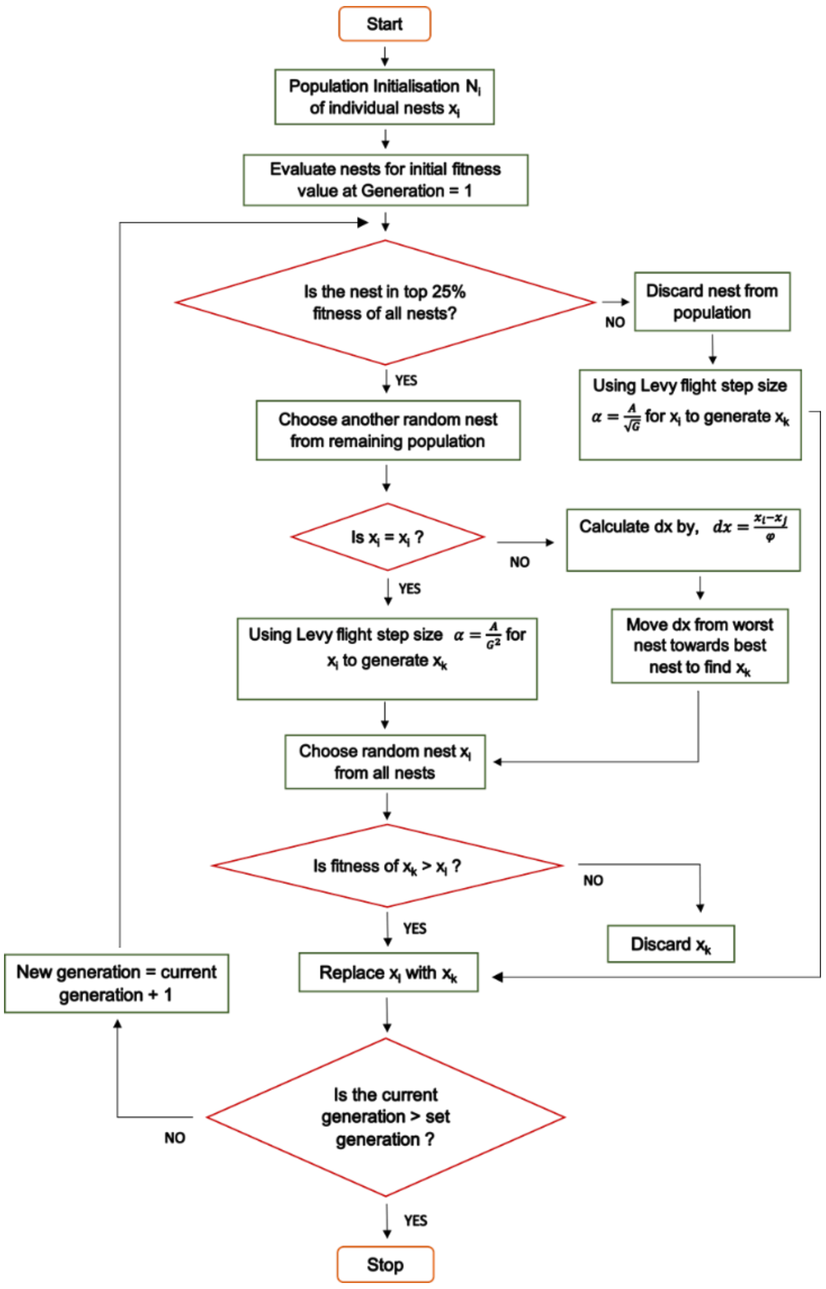
```
Start
------------------------------------------------------------
    Initialise population of size $N_i$ individual agents $x_i$
    Calculate fitness of all $x_i$
    Sort nests into order of fitness
------------------------------------------------------------
    If agent is in bottom 75% of overall agent fitness
    Calculate Levy flight step size: $\alpha = \frac{A}{\sqrt{G}}$
    Apply Levy flight to $x_i$ to create new agent position $x_k$
    Update agent position
------------------------------------------------------------
    If agent is in top 25% of overall agent fitness
    Compare agent position $x_i$ with another top agent position $x_j$
    If $x_i = x_j$
    Calculate Levy flight step size: $\alpha = \frac{A}{G^2}$
    Apply Levy flight to $x_i$ to create new agent position $x_k$
    Update agent position
------------------------------------------------------------
    If $x_i \neq x_j$
    Calculate distance: $dx = \frac{x_i - x_j}{\varphi}$
    Move dx from worst agent to best agent to define new agent position $x_k$
------------------------------------------------------------
    Choose random agent $x_l$ from all agents
    If fitness of $x_k$ > fitness of $x_l$
    Replace $x_l$ with $x_k$
------------------------------------------------------------
End
```

$A = Max\ Levy\ Flight\ Step\ Size$

$\varphi = Golden\ Ratio$

Figure 2 Summary of one generation of Modified Cuckoo Search

The fitness of the randomly generated positions is evaluated in the first generation. Aligning statistically, approximately half the agents reduce in fitness in the first generation, and half increase. This defines the path in which they will follow for levy flight implementation. Figure 2 and Figure 3, an agent in the bottom 75%, will have a broader search scope applied, where as an agent in the top 25%, which already holds a respectable fitness, lacks the need for diversity, hence the faster narrowing rate of the search scope in the levy flight calculation. After the Levy Flight step size is applied accordingly, and the agents' positions are updated, the positions are compared to each other. To increase interaction between the top agents, distance dx, is calculated to crossover the agents with a mutation constant referred to as the 'golden ratio'. This combines the vectors of the two agents whilst still implementing an aspect of randomness via the golden ratio. If two agents hold the same position, they are assumed to have high fitness, therefore they are applied to the narrower Levy Flight step size.



Start
Population Initialisation N_i of individual nests x_i
Evaluate nests for initial fitness value at Generation = 1
Is the nest in top 25% fitness of all nests?
NO
Discard nest from population
YES
Choose another random nest from remaining population
Using Levy flight step size $\alpha = \frac{A}{\sqrt{G}}$ for x_i to generate x_k
Is $x_i = x_j$?
NO
Calculate dx by, $dx = \frac{x_i - x_j}{\varphi}$
YES
Using Levy flight step size $\alpha = \frac{A}{G^2}$ for x_i to generate x_k
Move dx from worst nest towards best nest to find x_k
Choose random nest x_l from all nests
Is fitness of $x_k > x_l$?
NO
Discard x_k
YES
Replace x_l with x_k
New generation = current generation + 1
Is the current generation > set generation ?
NO
YES
Stop

Figure 3 Flowchart of an entire Modified Cuckoo Search optimisation [29]

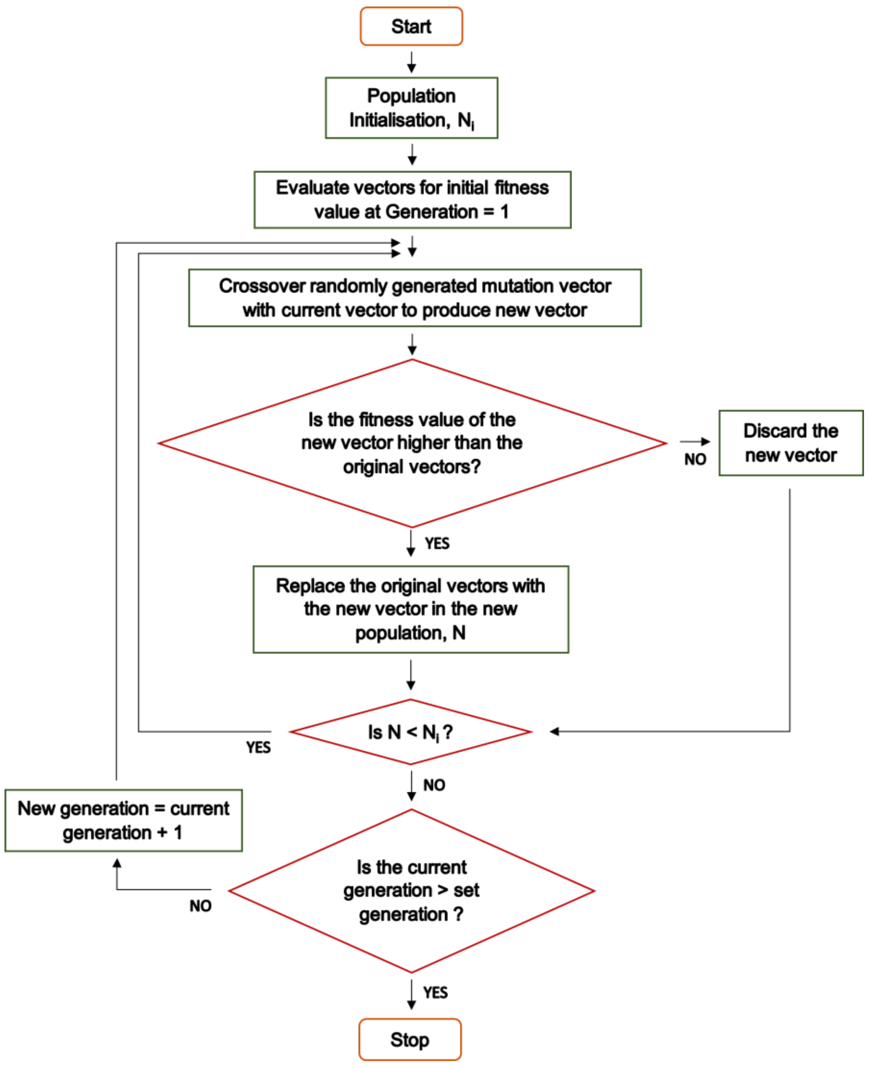
The interaction between agents was implemented into the MCS algorithm and did not exist in the original CS algorithm. The CS algorithm would only implement Levy flights to each individual agent, abandon the worst, to then continue mutating via Levy flights, evaluating each agent until the set conditions were met. This method relies on the chance of a random selection having high fitness and therefore may take longer to converge to an optimum. The interaction of separate agents in MCS aims to increase convergence rate. It still contains the main features of the cuckoo search however mimics a 'survival of the fittest' scenario more closely, by allowing the breeding of the fittest with the fittest [21].

1.4.2.Differential Evolution

The DE Algorithm was first developed by Storn and Price in 1995 [20]. The process of one generation is shown in Figure 4.

```
Start
--------------------------------------------------------------------------------
    Initialise population of size $N_i$ individual agents $x_i$
    Calculate fitness of all $x_i$
    Generate mutation vector *
    Apply mutation vector via crossover function to create new $x_j$
    If $x_j > x_i$
    Replace $x_i$ with $x_j$
    Else discard $x_j$
--------------------------------------------------------------------------------
End
```

Figure 4 Summary of one generation of Differential Evolution



Start
Population Initialisation, N_i
Evaluate vectors for initial fitness value at Generation = 1
Crossover randomly generated mutation vector with current vector to produce new vector
Is the fitness value of the new vector higher than the original vectors?
NO
Discard the new vector
YES
Replace the original vectors with the new vector in the new population, N
Is $N < N_i$?
YES
NO
New generation = current generation + 1
Is the current generation > set generation ?
NO
YES
Stop

Figure 5 Flowchart of an entire Differential Evolution optimisation [29]

Generating a mutation vector* requires three randomly selected vectors, **A**, **B**, and **C**. By calculating the difference between **B** and **C** and adding this to vector **A**, multiplied by a mutation constant (defined by the user), a mutation vector is formed [25]. Crossover between the mutation vector and current agents' vectors produces a new fitness. If the resulting fitness is higher than the original agent's fitness, it will be replaced. The larger the mutation constant, the more diverse a search within the design space. This can lead to slower convergence rate but a more thorough and varied exploration [26].

1.4.3.Particle Swarm Optimisation

Represented in Figure 6, Figure 7, and Figure 8, is the PSO algorithm, inspired by the natural swarm formation seen in groups of organisms, such as bird flocking or fish schooling [18]. The main difference with PSO, is that the population of agents possess knowledge of their own positions, their historical best positions, and the current global best, defined by the swarm.

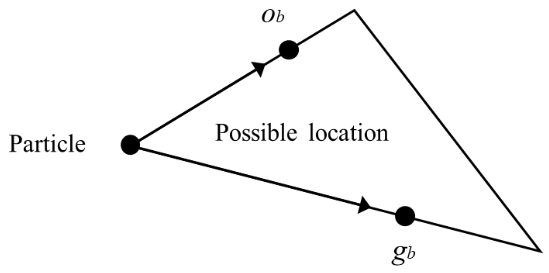


Figure 6 Diagram showing the possible best location of best fitness based on the best fitness particle velocity vector and the best global fitness velocity vector [10].

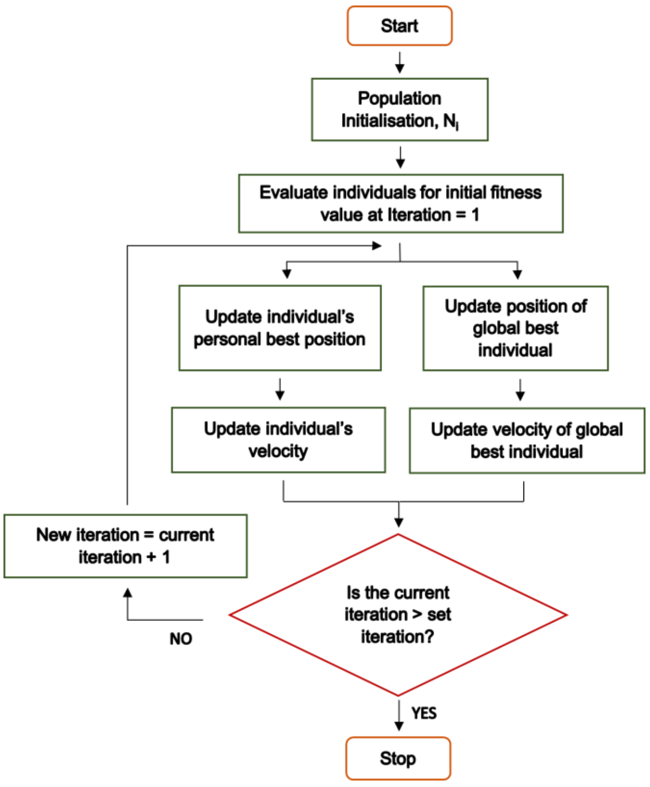
For each iteration, an associated fitness of the individuals is defined. Based on this, their velocity is updated to draw it towards the possible location of best fitness, marked in figure 4. This velocity may not be of the ideal magnitude and direction to begin with, but through each iteration, the updated velocity will become more accurate and as a result, narrow in on the area deemed optimum. To update the velocity in each iteration, a series of conditions and coefficients are set by the user to determine the rate in which the solution will converge. How far an individual can move in each iteration, the degree of pull towards the local memory of its best position, and the degree of pull on an individual towards the global best position, these are all factors that can be controlled by the algorithm user and are set according to the requirements of the problem [21].

PSO involves no 'breeding' of the agents, so the algorithm is not seen as generational, instead it uses iterations. PSO is not considered a genetic algorithm like DE and MCS, instead more a process inspired by biological events. In general, each iteration it focuses on the velocity between two agents, these two neighbouring agents are closest in distance. Once identified, the agent closest to the global best has its current X and Y velocities assigned to the other agent, leading both agents in the direction of the global best.

This method was successful but lacked diversity in the search, resulting in agents prematurely settling on the same path, posing the risk of receiving a 'false' optimum. To avoid this, a factor was randomly assigned to some X and Y velocities, called 'craziness', introducing variation into the process, giving a more lifelike appearance to the algorithm.

```
Start
------------------------------------------------------------
    Initialise population of size $N_i$ individual agents $x_i$
    Calculate fitness of all $x_i$
    Update agents' $x_i$ position (applying 'craziness factor') and global best position
    Update agents' $x_i$ velocity and global best velocity
------------------------------------------------------------
End
```

Figure 7 Summary of one iteration of Particle Swarm Optimisation



Start
Population Initialisation, N_i
Evaluate individuals for initial fitness value at Iteration = 1
Update individual's personal best position
Update position of global best individual
Update individual's velocity
Update velocity of global best individual
New iteration = current iteration + 1
Is the current iteration > set iteration?
NO
YES
Stop

Figure 8 Flowchart of an entire Particle Swarm Optimisation cycle [29]

1.5. Parameter Tuning

Parameter tuning defines the adjustment of input parameters used by optimisation methods such as evolutionary algorithms. EA input parameters consist of a design space, number of agents, number of generations, the algorithm itself and so on. An EAs optimisation process varies, notable from the three flowcharts detailed above (see sections 1.4.1, 1.4.2, 1.4.3). With these differences comes characteristics in its optimisation process, such as MCS having a 'percentage of agents discarded' feature or DE allowing the agents to interact. These characteristics become evident when the EAs are used for identical optimisation problems but output a variation of results. It can be argued that the randomising element in EAs drive the difference in results, however when identical problems are executed multiple times to produce an average, variation between the EA's results still remain.

Parameter tuning can greatly affect the performance of an algorithm, one of the main challenges is evaluating which parameters lead to these affects in performance [31]. Typically, an EA has its own sensitivity parameters to incorporate randomness into the process. For example, DE has a 'mutation constant' within the algorithm, MCS includes a 'Levy flight' step size, and PSO applies a 'craziness factor' into calculating the next agent position, all parameters which can be adjusted via the user through some disassembling of the algorithm code. Further parameters include inputs before running the optimisation. Increasing the number of agents inputted into the algorithm increases the number of search points and interactions between the agents, this ensures more of the design space has been searched. The larger the design space, the larger the area is necessary to explore, increasing the number of agents can achieve this. There are disadvantages to increasing the agent's parameter, one of which is computational resource. If the number of core processors available is less than the number of agents inputted, the agents will be required to be processed in series, where usually they are capable of parallel processing, so the optimisation wall-clock time increases. Increasing the number of generations inputted into the algorithm increases the number of cycles the agents search for, in theory, producing a more thoroughly searched result. Sometimes this may be the case, however EAs interactive nature tends to converge all the agents to the final solution sooner than the stopping criteria (maximum generation number), so unless full convergence of the agents is not observed, increasing the generation number is not seen as necessary. Furthermore, generations are run in series, meaning regardless of the computational resource available, the optimisation wall-clock time will increase if the number of generations is increased.

There is a fine line between improving the effectiveness of an EA through its input parameters and making its performance worse, and it is heavily dependent on the algorithm characteristics, availability of computational resources and the problem itself. All of which require extensive testing and analysis to begin suggesting frameworks for applying parameter tuning.

1.6. Application

Evolutionary algorithms are versatile, they require only initial parameters to evolve a design. From engineering design to optimising manufacturing and supply chain processes. EA's are a powerful tool for optimisation, and this is demonstrated through their effectiveness in a range of applications, from design optimisation of a cantilever beam to creating a complex aerodynamic design [32][33][34].

Implementing EA's into the Engineering design field would be a major step in simplifying the design process. Currently optimisation methods can be cumbersome for designers and won't always provide a single optimum solution. One focus of genetic algorithms within engineering is wing design, used to optimise the aerodynamic performance of aerofoils. 2D aerofoils are commonly tested on but lack integrity within application. 3D aerofoils make what is theoretical, more applicable to real-life.

The single stage to orbit conceptual Skylon spaceplane [35], designed by Reaction Engines Ltd, is an example where 3D aerodynamic surfaces undergo ASO, with the potential of utilising several techniques to do so. The objective function (sometimes referred to as the fitness function) is a dependent variable and will fit the given problem. For example, optimising a surface part on the Skylon concept design may involve an objective function to maximise lift whilst minimising drag, to ensure optimum efficiency through its flight phases.

An optimisation study depends on the goal dependant objective function. As an example, in the manufacturing industry, every product needs to be manufactured to a high quality, at as fast and cheap rate as possible. Manufacturability and materials selection must include the required mechanical properties whilst being cost efficient and most importantly include robust safety systems. Including multiple aspects of a design process into a single optimisation is regarded as a multi-objective optimisation.

1.7. Visualisation

Evolutionary algorithms (EA) generate data on a mass scale, where aside from the converged final values, extraction and comprehension of the whole data is often intractable. Visualisation is an essential segment in the understanding and implementation of data gathered through algorithms, and it refers to a variety of techniques for displaying and interpreting EA data, the most common of these being graphical analysis.

During research, algorithms are commonly used to optimise processes or designs. This could be in a monitory, efficiency or practicality sense. However, to assess, accept and implement findings, it is important for the data to be clearly and concisely understood by individuals of varying knowledge on the algorithms.

Xue and Liu determined that the inclusion of visualisations aids in the overall workload of the user to be lessened and subsequently their participation was of greater value [36]. The potential of this being utilised was confirmed by Packham and Parmee who displayed that the combination of visualisations with EAs greatly reduce the quantity of data given to the user and instead streamline the essential information [37]. There has been much research on the development and evaluation of visualisation techniques alone, with the majority of these focusing on the foundations of visualisation, the various methods and their applications[38]

There have been various techniques appraised for visualising an EA, such as exploring generational results to understand the fitness progression of the algorithm. Taking the optimal value from each generation provides clear approximations of the convergence, while evaluating all values at a given generation constructed a diagram able to display the convergence of optimal results throughout an optimisation [39]. Collins additionally analysed visualisation within EAs, adopting a software approach, aimed to identify and examine the solutions given by EA's and the design space they investigated through software visualisation, suggesting a 'HENSON' framework [40]. This framework, along with a series of other developed frameworks such as 'VIZ' software and more recently, 'MRtrix3' allow for clear and concise data provision while targeting specific needs of the EA community. With technological advances this has been developed from 2D plots such as charts and graphs to 3D design space plots [41]. This advancement enhances and deepens potential, as seen in an empirical study on students, which concluded that, visualisation that was more diverse and readily interactable resulted in an overall better understanding [42].

1.8. Computational Fluid Dynamics

Computational Fluid Dynamics (CFD) has revolutionised methods within aerodynamic design [27][28]. Prior to the introduction of CFD methods, simulating the flow around an aerodynamic surface required simulation of atmospheric conditions in a wind tunnel or full scale flight tests [43]. These methods of flow analysis are costly and time consuming, hence CFD along with computer aided design (CAD) together have become an essential tool for streamlining an aerospace engineering design process.

The governing equations forming Newtonian CFD today are the Navier-Stokes equations, used to implement complex calculations to simulate the fluid flow interaction with the defined aerodynamic surfaces. This allows the prediction of aerodynamic performance through the lift, drag, and pressure coefficients, calculated via the sum of the integrated pressure values around the geometry [28].

More recently CFD has been combined with ASO, leading to an automated loop between initialising a design, applying it to a CFD solver, and analysing the results, to follow by re-initialising a new design for further optimisation. The CFD in this research is combined with ASO through AerOpt, an aerodynamic optimising software that utilises FLITE2D as a backend CFD solver. FLITE is an edge-based, vertex-centred finite volume discretised solver of the compressible Reynolds-Averaged Navier-Stokes (RANS) equations [50]. The applicable governing equations, the compressible Navier-Stokes equations, are commonly denoted as,

$$\rho\left(\frac{\partial u}{\partial t} + u.\nabla u\right) = -\nabla_p + \nabla.\left[\mu(\nabla u + \{\nabla u\}^T) - \frac{2}{3}\mu\{\nabla u\}I\right] + F \qquad (1)\ [27]$$

where the left hand side of equation (1) is the total inertial forces. The first part of the sum on the right hand side is the pressure forces, the second part is the viscous forces, and the last part, F, is the external forces. All flows assessed in this research were treated as viscous and compressible within the CFD solver. The CFD aspect of this research is used on the basis of an application for the algorithm's behaviour to be analysed; the CFD approach used is described briefly in the methodology.

1.9. Exploitation of High Performance Computing

EAs in particular are well suited to the exploitation of high performance computing due to the nature of their input parameters. The two main input parameters required of an EA is the number of agents, which are run in parallel, and the number of generations, which are run in series. Given the required computing power is available, this allows for any number of agents to be inputted without affecting the length of the optimisation. For example, if 100 agents are inputted, and 100 cores are within processing limits, the optimisation would span the number of generations inputted and that only. Providing a reliable prediction of the computing resources required before any testing has been done. This also benefits the algorithm's requirements in a parameter tuning sense, depending on an EAs characteristics, some perform better when the number of agents is increased, given the computing resources are available, this would not change the optimisation run time.

AerOpt Development with SA2C ***October 2021 - June 2022***	
Created a Supercomputing Wales account for HPC use	
Identified AerOpt's current limitations	
	Graphical user interface unable to display outputted data during and after optimisation on HPC
	HPC connection tempremental
	Portability of AerOpt between PCs
Enquired SA2C for support on the project	
Collaboration with SA2C ensured current version of AerOpt had a functional HPC connection	
Enquired SA2C for further guidance on improving the portability of AerOpt	
SA2C currently able to to build AerOpt via Docker	
SA2C's RSE (Dr Michael Pei) assigned to project to support with the portibility and building of AerOpt	
Success in enabling the build process on windows to be executed via QT Creator, without necessitating the use of Docker in the previous build process	
AerOptGUI now buildable, transferable and deployable on other windows machines	
RSE further implementations	
	Increased frequency of updates between the local machine and cluster to ensure a necessary connection remains & user is aware of the progress of the optimisation
	When lack of updates between the local machine and cluster within a set time-frame are detected, the optimisation process is terminated. This prevented AerOpt hanging on the cluster unnecessarily, as it may have done previously.
	Adjustment of directories in which results are transferred into from the cluster
	Adjustment of the set time-frame to ensure the stopping criteria did not terminate the optimisation prematurely
	Adjustment of files transferred to the local machine from the cluster
A usable, portable and buildable version of AerOpt achieved	

Figure 9 A timeline of changes made through the development of AerOpt with Swansea Academy of Advanced Computing

The AerOpt software was the focus of the support given from SA2C in this project. AerOpt was used to run local optimisations using an option of one of the three evolutionary algorithms described in sections 1.4.1, 1.4.2, 1.4.3. There were two main objectives, to develop the buildability and transferability of the software from one pc to another, and to implement a functional HPC connection into the software, to allow larger optimisation cases to run.

Initially, the AerOpt software had only been utilised for local optimisations limiting the scale in which the EAs could be tested to. At planning stages, it was decided that the projected work in this thesis was to require HPC resources, therefore this was a priority to integrate into the AerOpt software, to allow larger optimisation cases to be performed. The HPC resources were provided by Supercomputing Wales which an account and project was created to facilitate the research. Optimisation cases run through AerOpt connected to the cluster, and used the Sunbird HPC system. The Sunbird system is compromised of 126 nodes, totalling 5040 cores, and is mostly used by Swansea and Aberystwyth University researchers [46]. Once a successful HPC connection had been established, there were some minor adjustments made, such as the relay of information back to the local machine, and some stopping criteria detailed in Figure 9 to ensure the optimisation didn’t hang on the cluster if it was terminated locally.

It was also necessary to ensure the software could be transferred between windows machines for the benefit of future researchers. The build process was streamlined by facilitating the use of Qt Creator and GitHub, guaranteeing that a non-computer science based user would be able to navigate and execute the build process with some further simple inputs for deployment. Once a user has built the software on their windows machine, it is now possible to compress the file and send to other users to facilitate.

2. Methodology

2.1. Mesh

2.1.1. Generating a Mesh

Mesh parameters such as the number of elements, number of points and number of boundary points, were created using the mesh generator. The mesh parameters were inputted through a defined element size and radius in a background file, and the boundary layer properties in a geometry file. All meshes were executed using the FLITE2D package.

To ensure the correct formatting was inputted into the pre-processor, a MATLAB script was written to alter the mesh file format in which it states the mesh parameters. The pre-processor created a mesh solver file in which the solver executable could input to assess the convergence of the mesh, along with the solver input file, defining the necessary flow conditions, from FLITE2D [27]. The solver iteratively solves the Navier Stokes flow equations, outputting the coefficient of lift and drag as the residual reduces to the convergence criteria.

The solver executable failed for element sizes less than 0.02 at Mach 0.5, this was due to a combination of finer element size and a subsonic Mach number. To maintain consistent results, an element size less than 0.02 applied to the entire domain, was not considered in the result analysis for the transonic and supersonic Mach speeds.

Convergence criteria was defined inside the input file entered into the solver executable. The primary convergence criteria consisted of the residual value reaching -3. If this condition was not fulfilled, the solver continued iterating until the maximum iteration number of 50,000 iterations was reached.

2.1.2. AerOpt's Default Mesh Options

There are many approaches to mesh generation, the aerofoil demonstrated on in this study was the NACA21120 in which a two-dimensional unstructured triangular mesh was implemented using the Swansea University FLITE 2D system [27]. Within the boundary layer, the advancing layers technique provided a suitable mesh for displaying boundary layer behaviour, with a mesh convergence study assessing results regarding mesh setting variations in section 3.1. Within FLITE, a Radial Basis Function (RBF) is used to apply the parameterisation onto an existing surface mesh, creating a morphed mesh with the same topology as the original mesh.

AerOpt included three meshes integrated within the code. Default AerOpt mesh options consisted of a 'coarse', 'medium', and 'fine' mesh setting shown in Figure 10, Figure 11, and Figure 12.

Table I Default mesh settings provided in AerOpt

AerOpt default meshes		Radius					Total		
Setting name	Element size	Inner	Outer	First layer size	Expansion rate	No. of layers	Elements	Nodes	BL nodes
Coarse	0.15	0.4	0.6	0.001	2.72	10	538	284	30
Medium	0.07	0.4	0.6	0.001	2.72	10	1468	757	46
Fine	0.02	0.3	0.7	0.001	2.72	10	14814	7467	120

Visible in Table I, as the mesh setting becomes finer, the total number of elements, total nodes, and nodes in the boundary layer increase. The inner and outer radii are parameterised from a line source which in this case, is defined by the aerofoil chord. The inner radius defines the area in which the element size for the particular mesh is applied. Outwards of the inner radius, the element size increases until it reaches the outer radius. For the fine mesh setting, the inner and outer radii are slightly different parameters, having a smaller inner diameter meant the smaller element size is applied to a smaller domain, reducing computational power required to produce the mesh. The first layer size defines the first layer from the surface of the aerofoil boundary included in the boundary layer, the expansion rate then defines the rate in which the layer size increases, starting from the first layer. As suggested in the name, no. of layers is the number of layers included inside the boundary layer, and the number of nodes inside the boundary layer can be seen in the last column of Table I.

Figure 10, Figure 11, and Figure 12 are the default mesh options available in AerOpt, with their parameters detailed in Table I.

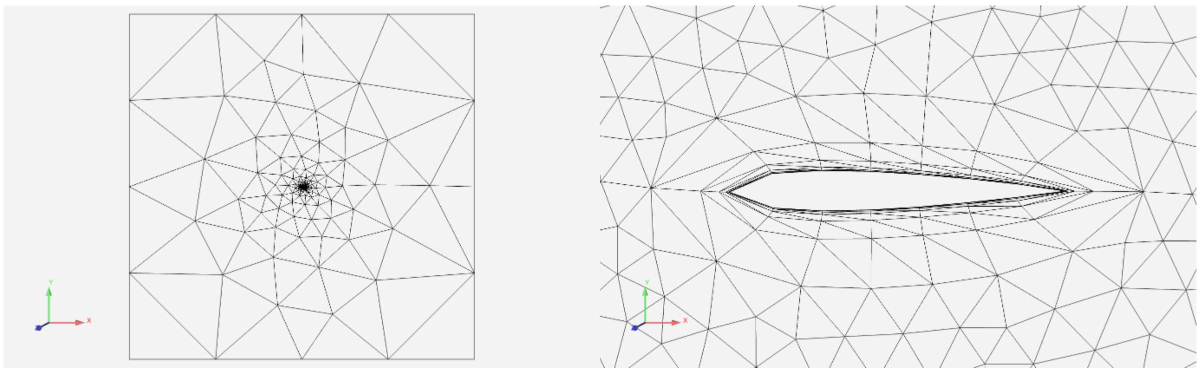


Figure 10 Left: Entire domain of 'coarse' mesh setting on the aerofoil. Right: Zoom of 'coarse' mesh.

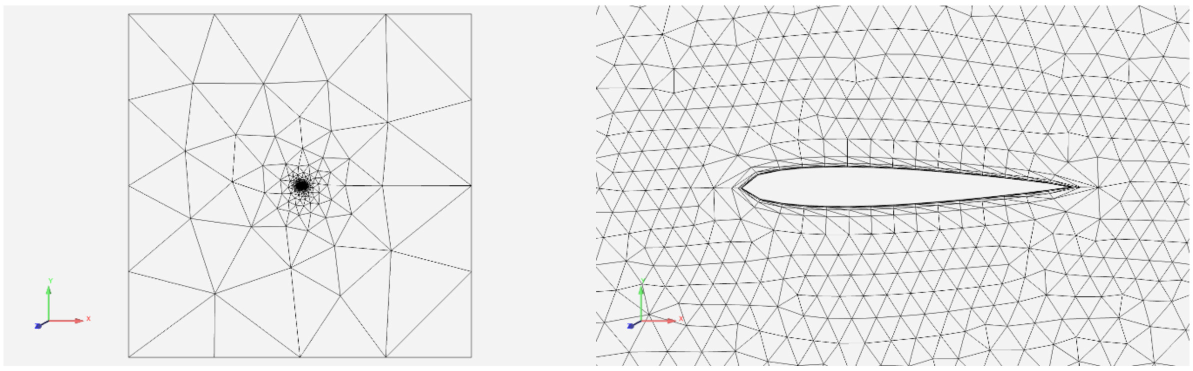


Figure 11 Left: Entire domain of 'medium' mesh setting shown on the aerofoil. Right: Zoom of 'medium' mesh.

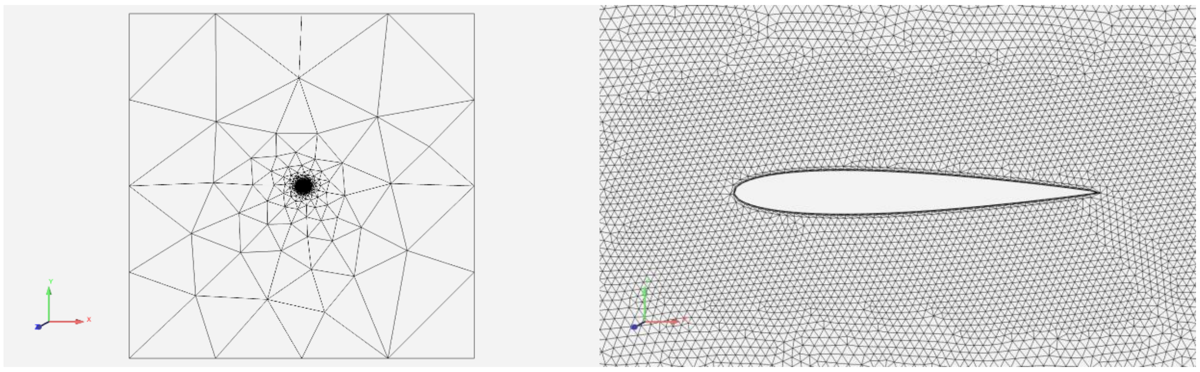


Figure 12 Left: Entire domain of 'fine' mesh setting shown on the aerofoil. Right: Zoom of 'fine' mesh.

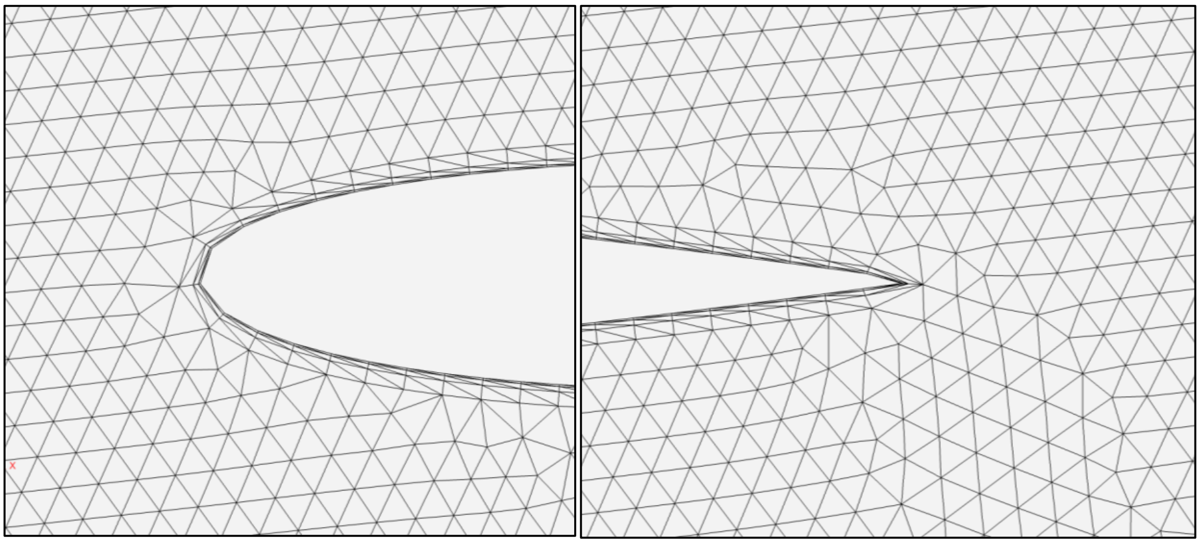


Figure 13 Left: Zoom of aerofoil leading edge 'fine' mesh. Right: Zoom of aerofoil trailing edge 'fine' mesh

Seen in Figure 13, the fine mesh setting represents the aerofoil leading and trailing edge poorly, a point source of smaller element size was added to achieve higher resolution at the aerofoil leading edge. Further mesh studies to understand how the addition of a point source alters the CFD results are plotted in the results section.

2.2. Control Node Approach

2.2.1. One Control Node

Control nodes in this research, utilise points that make-up the aerofoil's boundary, in which the geometry of this point has an allowable movement within a defined coordinate design space.

Cases A – C present optimisations executed with one control node on the NACA21120 aerofoil upper surface. This control node was chosen since it is the highest point on the upper surface. It is moveable in the x-axis and y-axis, giving the optimisation a total of two degrees of freedom.

The leading and trailing edge points were treated as fixed nodes by setting the design space as zero. This ensured the angle of attack was fixed through all optimisations, making the analysis wholly dependent on the geometry of the aerofoil.

Figure 14 shows the control node in red with a fixed design space, and fixed nodes in blue at the LE and TE of the aerofoil.

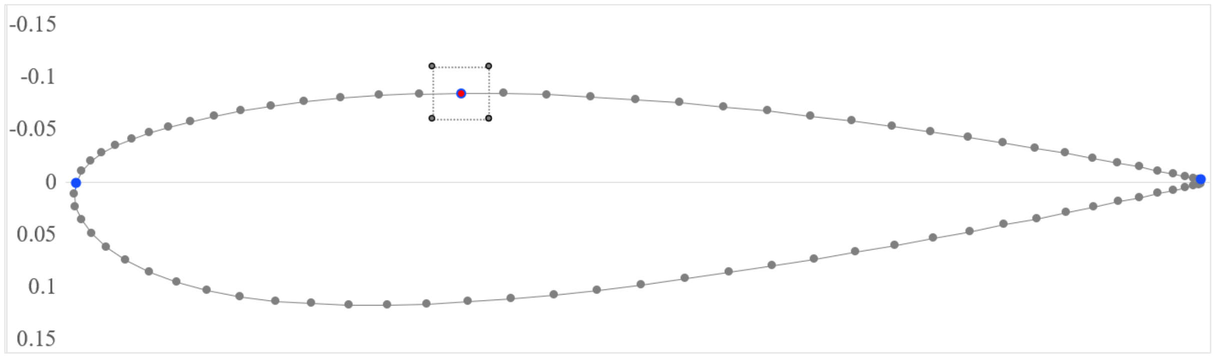


Figure 14 NACA21120 aerofoil. Upper surface control node marked in red and fixed nodes in blue. Upper surface control node coordinates: (0.341355, -0.0843716).

2.2.2.Multiple Control Nodes

Larger degree of freedom cases consisted of six moveable control nodes, the leading edge and trailing edge remained as fixed nodes. Each control node was moveable in the x-axis and y-axis, giving the problem twelve degrees of freedom.

The upper surface control node locations were calculated from the midpoint between the original upper surface control node (CN2) and the leading and trailing edge, and vice versa for the lower surface. Similar to the upper surface control node, the mid-lower surface control node was chosen as it held the lowest y-coordinate on the lower surface of the aerofoil. Coordinates of the control nodes can be seen in Table II.

Table II Six control nodes including their corresponding coordinates and allowed design space.

Control Node		*Coordinate*		*Design Space*			
		x	*y*	*x min*	*y min*	*x max*	*y max*
Upper Surface 1	CN1	0.724279	0.052994	-0.025	-0.025	0.025	0.025
Upper Surface 2	CN2	0.341355	0.084372	-0.025	-0.025	0.025	0.025
Upper Surface 3	CN3	0.101488	0.057589	-0.025	-0.025	0.025	0.025
Lower Surface 1	CN4	0.064806	-0.085317	-0.025	-0.025	0.025	0.025
Lower Surface 2	CN5	0.275952	-0.116665	-0.025	-0.025	0.025	0.025
Lower Surface 3	CN6	0.691375	-0.066595	-0.025	-0.025	0.025	0.025

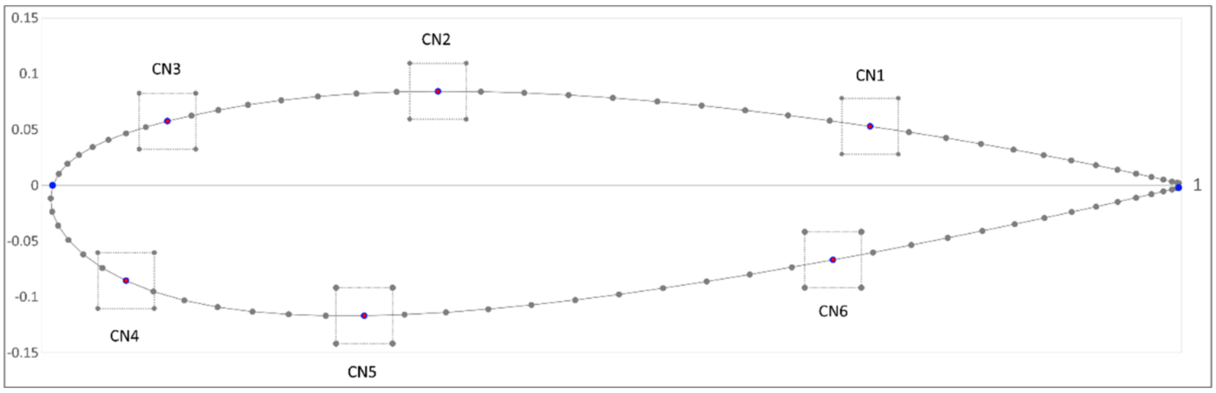


Figure 15 Six chosen control nodes marked on the aerofoil.

Apart from the control nodes, and the leading edge and trailing edge single points, that have been fixed to the axis during optimisation, the remaining nodes making up the aerofoil boundary are able to move in accordance with adjustment of the control nodes. The main movement originates from the selected control nodes, movement from other nodes is for the purpose of boundary smoothing.

2.3. Fluid Dynamics

In the FLITE 2D solver, the fluid was treated as steady-state and viscous, therefore the Reynolds number for the corresponding Mach number were included, utilising the Navier-Stokes solutions. For the subsonic, transonic, and supersonic flow, Reynolds number values were calculated using the fluid velocity, dynamic viscosity, and density seen in Table III. For simplicity, the air density was left as its default (sea-level air density) shown rounded to one decimal place in Table III.

Table III Flow conditions covering all optimisation cases.

Parameter	**Symbol**	**Unit**	**Mach**		
			0.5	**0.8**	**1.5**
Chord	*l*		1	1	1
Temperature	*T*	K	303	303	303
Gas constant	*R*	J/K-kmol	287	287	287
Adiabatic ratio	*γ*		1.4	1.4	1.4
Speed of sound	*c*	m/s	349	349	349
Density	*ρ*	kg/m^3	1.2	1.2	1.2
Dynamic viscosity	*μ*	kg/ms	1.73E-05	1.73E-05	1.73E-05
Reynolds number	*Re*		1.21E+07	1.93E+07	3.62E+07

2.4. Optimisation Parameters

2.4.1. Objective Function

The objective function for all optimisation cases was the maximise the L/D, taken from the coefficient of lift, C_L, (eq. (2)), and the coefficient of drag, C_D (eq. (3)), combined into eq. (4).

$$C_L = \frac{L}{q_\infty cb} \tag{2}$$ [44]

$$C_D = \frac{D}{q_\infty cb} \tag{3}$$ [44]

$$Lift - \ Drag\ ratio = \ \frac{L}{D} = \frac{C_L}{C_D} \tag{4}$$ [44]

$$f = Lift - Drag\ ratio \tag{5}$$

In the subsequent results section, values listed as the fitness directly correspond to the L/D calculated for a particular geometry. The L/D value gives an estimate of the aerofoil's aerodynamic performance with respect to minimising drag and maximising lift, achieving the largest ratio possible for the given parameters.

2.4.2. Exploring the Number of Agents & Generations

Within the separate evolutionary algorithms, three agent variations were tested. Optimisations with 10, 50, and 99 agents were all given a stopping criteria of 99 generations, AerOpt allowed 99 agents and generations as the maximum number input for each, for a single optimisation. This meant the maximum stopping criteria was given for all cases, this acted as a precautionary measure to ensure a tolerated final fitness conversion was seen in the results. Following this, the primary aspect of the results was to analyse any changes due to varying the number of agents; generations remained unchanged regardless of the agent number. The algorithm's performance along any point of the optimisation was independent of the inputted generation number, for example, an optimisation set to 99 generations still allows for the user to interpret results at a given generation, such as 60. If the optimisation was set at 60 generations beforehand, the result at generation 60 will be the same as it would be if the optimisation was inputted to run on 99 generations. The corresponding case numbers are listed in section 2.5.

2.5. Summary

There were 21 variations of the input parameters, therefore 21 test cases looked at in this research. Any test case marked with a starting letter of A, were run on DE, B means it was run on MCS, and C, PSO. The test cases were then broken down further into the number of agents inputted into the optimisation, 1 for 10 agents, 2 for 50 agents, and 3 for 99. Each number representing a set number of agents was separated into three Mach speeds, 1 indicated Mach 0.5, 2 for Mach 0.8, and 3 for Mach 1.5. For example, an optimisation run on MCS with 50 agents at Mach 1.5 will be labelled test case B.2.3.

Table IV Summary of cases with one control node

Case	*Optimising Algorithm*			*Agents*	*Control Nodes*	*Mach*	*Re*
A	*Differential Evolution*	A.1	A.1.1			0.5	1.21E+07
			A.1.2	10	1	0.8	1.93E+07
			A.1.3			1.5	3.62E+07
		A.2	A.2.2			0.8	1.93E+07
			A.2.3			1.5	3.62E+07
		A.3	A.3.2			0.8	1.93E+07
			A.3.3			1.5	3.62E+07
B	*Modified Cuckoo Search*	B.1	B.1.1			0.5	1.21E+07
			B.1.2	10	1	0.8	1.93E+07
			B.1.3			1.5	3.62E+07
		B.2	B.2.2			0.8	1.93E+07
			B.2.3			1.5	3.62E+07
		B.3	B.3.2			0.8	1.93E+07
			B.3.3			1.5	3.62E+07
C	*Particle Swarm Optimisation*	C.1	C.1.1			0.5	1.21E+07
			C.1.2	10	1	0.8	1.93E+07
			C.1.3			1.5	3.62E+07
		C.2	C.2.2			0.8	1.93E+07
			C.2.3			1.5	3.62E+07
		C.3	C.3.2			0.8	1.93E+07
			C.3.3			1.5	3.62E+07

Table V Summary of cases with six control nodes. Three optimisations included in the higher degree of freedom test cases. Each with the same input parameters but with a different Evolutionary Algorithm.

Case	*Algorithm*	*Mach*	*Reynolds no.*	*Agents*	*Generations*
D	Differential Evolution	1.5	3.62E+07	10	99
E	Modified Cuckoo Search	1.5	3.62E+07	10	99
F	Particle Swarm Optimisation	1.5	3.62E+07	10	99

3. Results & Discussion

3.1. Mesh Convergence Study

3.1.1. Introduction

The scale of optimisations deemed a mesh dependency study necessary to ensure higher accuracy in results, the fine mesh thus far had no evidence it could provide this. This assessed the most suitable available mesh for application to the specific flow parameters in the listed optimisation cases above.

The study was built off the existing default mesh parameters, finer meshes were tested as well as incremental element sizes within the default element sizes. Visual analysis of the complete mesh using EnSight indicated the finest mesh option of element size 0.02 was mostly sufficient to represent the shape of the aerofoil accurately without over-requiring computational resources.

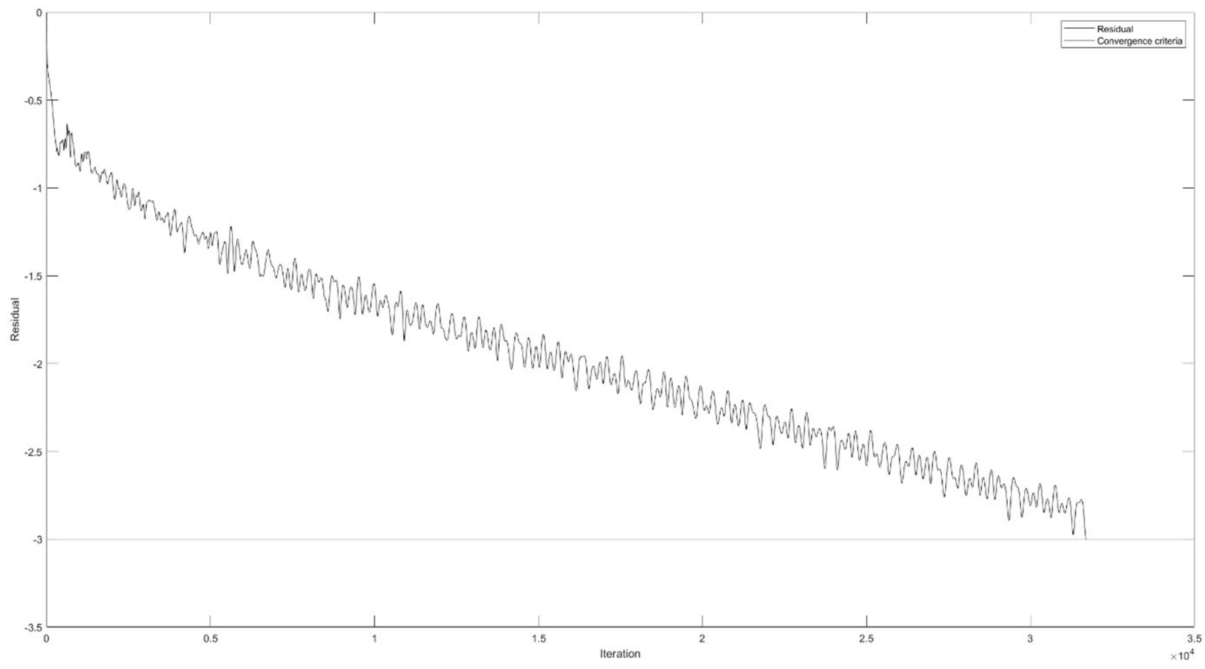


Figure 16 An example of the residual value reducing to the convergence criteria of -3 over approximately 32000 iterations. Taken from a mesh of element size 0.02, converged for Mach 0.8.

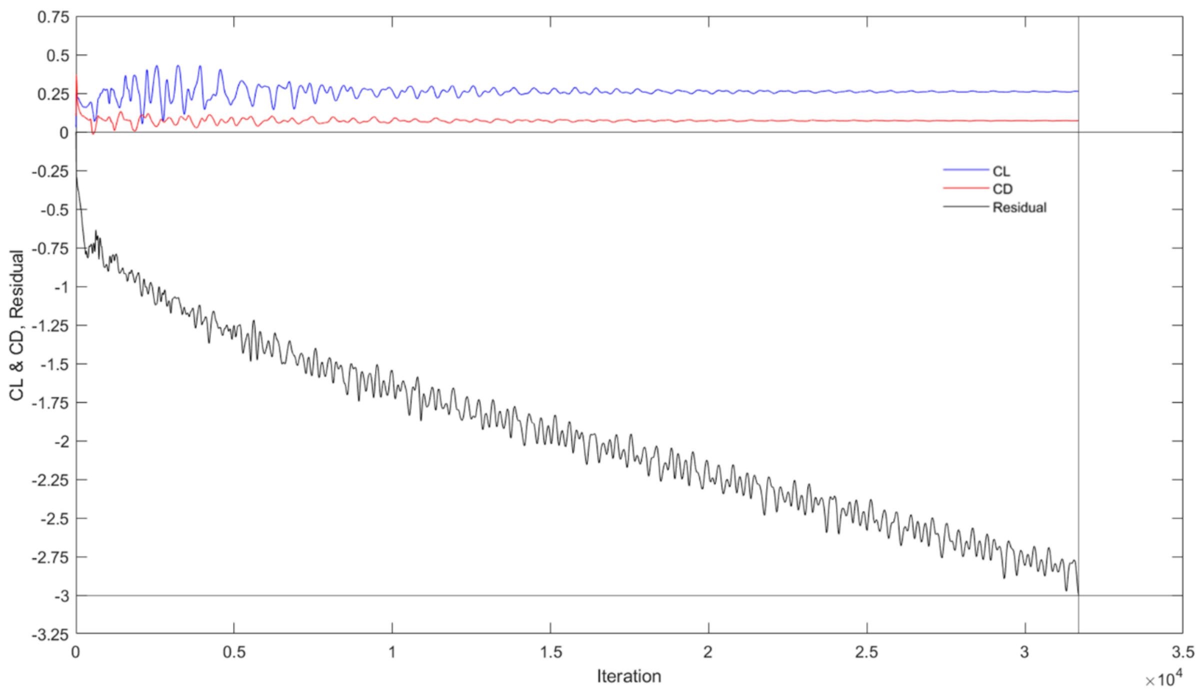


Figure 17 Example residual value seen same as above, with the addition of the C_L and C_D value observable over the number of iterations. Where the C_L and C_D value can be seen stabilising as the mesh converges.

Figure 24 shows the residual value decreasing towards the stopping criteria of value -3. The residual value is produced via the Navier-Stokes equations to a set stopping criteria mentioned previously. The FLITE CFD solver is an edge–based, vertex–centred finite volume discretisation for solution of the compressible Reynolds Averaged Navier–Stokes equations [49][50]. In figure 25, the lift and drag coefficient values can be seen plotted simultaneously as the residual value converges. Figure 16 and Figure 17 are an example taken from the convergence of a fine mesh and provide a visual of the plotted convergence values that every considered mesh was tested for.

3.1.2.Element Size Variation

Table VI Set of tables displaying the coefficient of lift and coefficient of drag values at each element size. The element sizes were varied with number of layers in the boundary layer, seen in the first column – AerOpt default element sizes are highlighted in green lists the coefficient of lift and coefficient of drag values obtained from testing varied mesh element sizes.

Table VI Set of tables displaying the coefficient of lift and coefficient of drag values at each element size. The element sizes were varied with number of layers in the boundary layer, seen in the first column – AerOpt default element sizes are highlighted in green. First table: Mach 0.5. Second: Mach 0.8. Third: Mach 1.5.

Mach 0.5		**Element Size**													
No of layers in BL	First layer size	0.010		0.015		0.020		0.025		0.030		0.070		0.15	
		CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD
10	0.001000	-	-	-	-	0.2615	0.0722	0.2640	0.0970	0.2565	0.1249	0.2262	0.3617	0.1552	0.8732
12	0.000250	-	-	-	-	0.2545	0.0725	0.2509	0.0999	0.2491	0.1274	0.2147	0.3555	0.1786	0.8524
14	0.000063	-	-	-	-	0.2563	0.0738	0.2431	0.0987	0.2336	0.1266	0.2078	0.3457	0.1787	0.8344
16	0.000016	-	-	-	-	0.2509	0.0728	0.2327	0.0990	0.2301	0.1240	0.2045	0.3419	0.1754	0.8428
18	0.000004	-	-	-	-	0.2382	0.0723	0.2299	0.0965	0.2244	0.1234	0.1945	0.3371	0.1789	0.8122
20	0.000001	-	-	-	-	0.2391	0.0738	0.2188	0.0990	0.2163	0.1244	0.1925	0.3354	0.1806	0.8044

Mach 0.8		**Element Size**													
No of layers in BL	First layer size	0.010		0.015		0.020		0.025		0.030		0.070		0.15	
		CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD
10	0.001000	0.4051	0.0062	0.4027	0.0059	0.4002	0.0560	0.3816	0.0743	0.3444	0.0945	0.2267	0.2463	0.1957	0.5172
12	0.000250	0.4335	0.0080	0.4272	0.0078	0.3685	0.0574	0.3316	0.0768	0.2570	0.0979	0.2026	0.2439	0.1918	0.5043
14	0.000063	0.4369	0.0079	0.4032	0.0071	0.3654	0.0579	0.3067	0.0762	0.2699	0.0964	0.1880	0.2372	0.1904	0.4908
16	0.000016	0.4266	0.0070	0.4002	0.0072	0.3534	0.0580	0.2985	0.0761	0.2494	0.0954	0.1842	0.2359	0.1813	0.4809
18	0.000004	0.4156	0.0078	0.3962	0.0076	0.3313	0.0573	0.2815	0.0754	0.2429	0.0949	0.1719	0.2334	0.1768	0.4723
20	0.000001	0.3708	0.0075	0.3352	0.0080	0.2939	0.0598	0.2159	0.0777	0.1790	0.0975	0.1597	0.2330	0.1706	0.4645

Mach 1.5		**Element Size**													
No of layers in BL	First layer size	0.010		0.015		0.020		0.025		0.030		0.070		0.15	
		CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD	CL	CD
10	0.001000	0.1583	0.1012	0.1488	0.1058	0.1419	0.1232	0.1514	0.1348	0.1370	0.1486	0.0855	0.2425	0.0694	0.3777
12	0.000250	0.1528	0.1020	0.1476	0.1072	0.1409	0.1248	0.1443	0.1369	0.1284	0.1497	0.0903	0.2449	0.0619	0.3607
14	0.000063	0.1407	0.1025	0.1621	0.1084	0.1412	0.1258	0.1409	0.1382	0.1263	0.1511	0.0931	0.2407	0.0683	0.3550
16	0.000016	0.1608	0.1037	0.1583	0.1092	0.1457	0.1266	0.1376	0.1389	0.1293	0.1527	0.1044	0.2412	0.0648	0.3465
18	0.000004	0.1664	0.1038	0.1596	0.1092	0.1513	0.1284	0.1366	0.1394	0.1199	0.1520	0.0964	0.2379	0.0649	0.3384
20	0.000001	0.1466	0.1035	0.1568	0.1103	0.1377	0.1287	0.1316	0.1410	0.1226	0.1546	0.0895	0.2361	0.0609	0.3310

Element sizes of 0.01 and 0.015 applied to the entire domain were unsuitable for the subsonic Mach regime, as stated in section [3.1.2]. The CFD solver (FLITE2D) used by AerOpt is suited to transonic and supersonic flow due to the purpose in which it was created, as a result, Mach 0.5 is the lowest subsonic speed the CFD executed without exceeding computational power resources.

Element sizes 0.01 and 0.015 are included in Table VI for Mach 0.8 and Mach 1.5, however were not included in the graph data seen in Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, Figure 23 to aid in consistent analysis of results, in line with Mach 0.5.

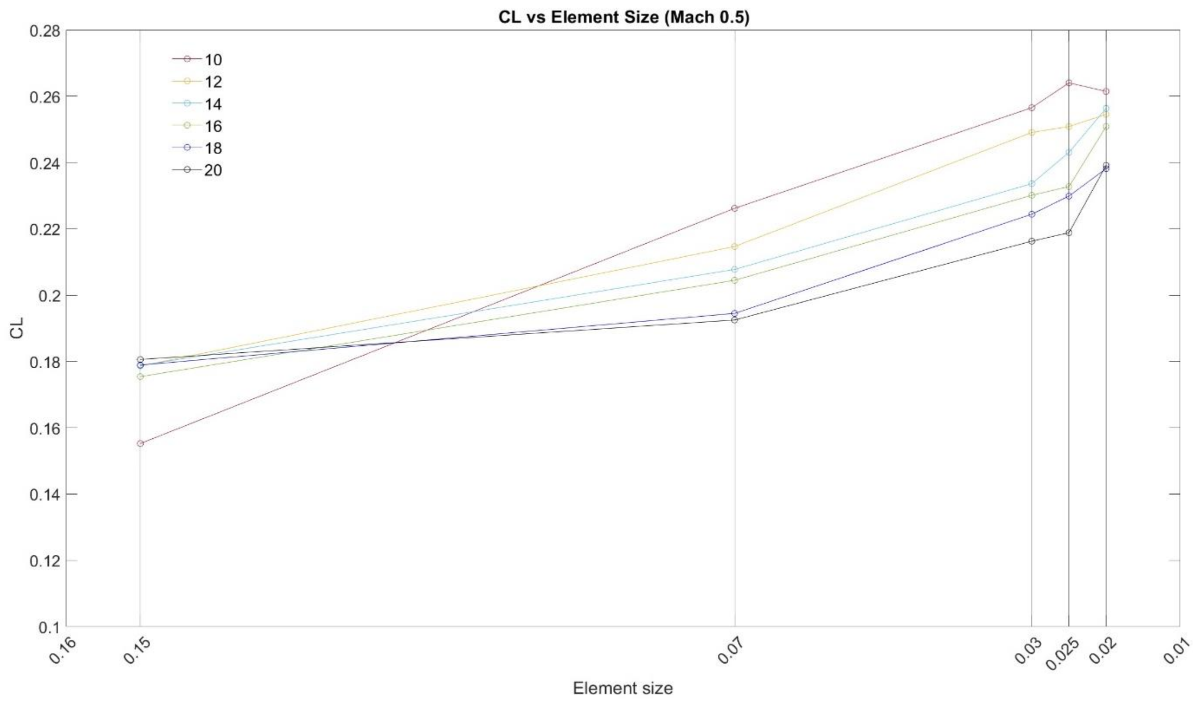


Figure 18 Graph showing the CL (y-axis) against element size (x-axis) from meshes tested at Mach 0.5. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

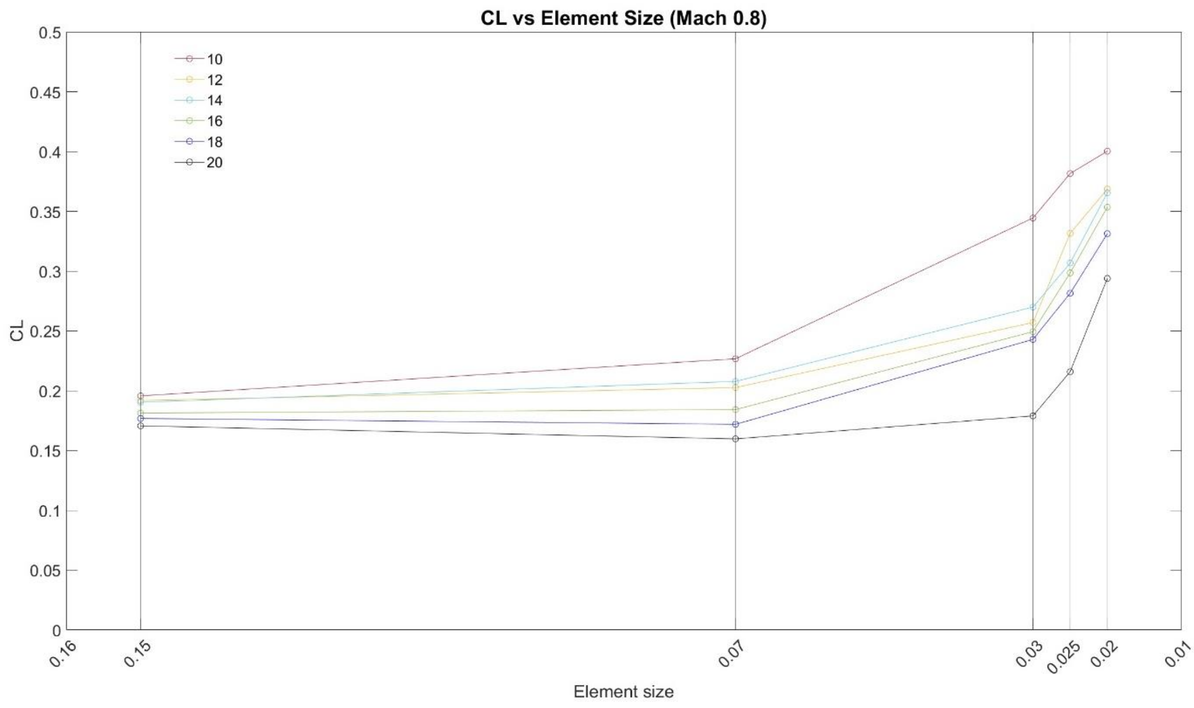


Figure 19 Graph showing the CL (y-axis) against element size (x-axis) from meshes tested at Mach 0.8. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

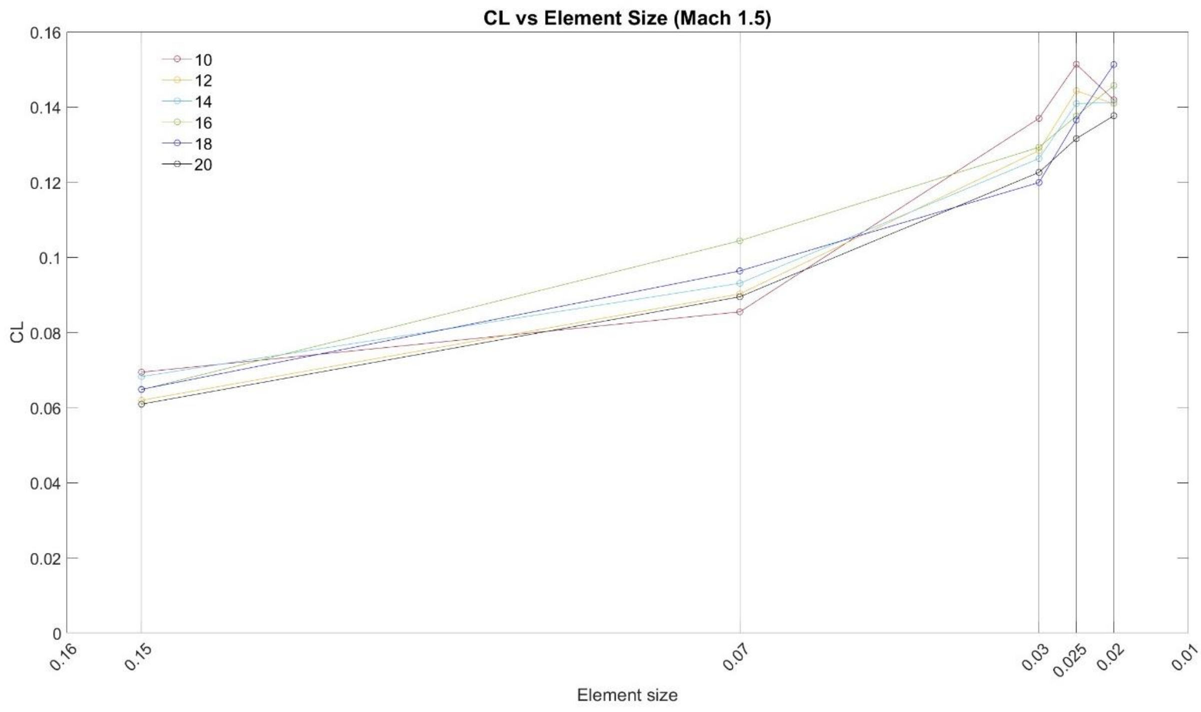


Figure 20 Graph showing the CL (y-axis) against element size (x-axis) from meshes tested at Mach 1.5. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

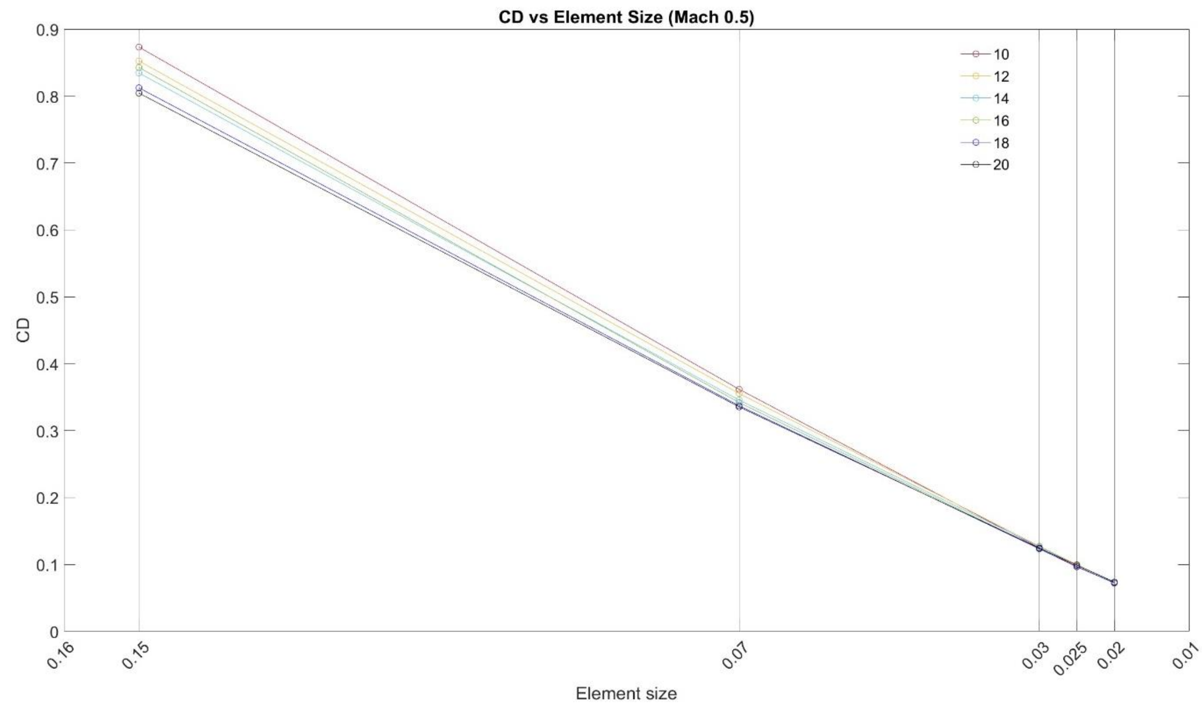


Figure 21 Graph showing the CD (y-axis) against element size (x-axis) from meshes tested at Mach 0.5. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

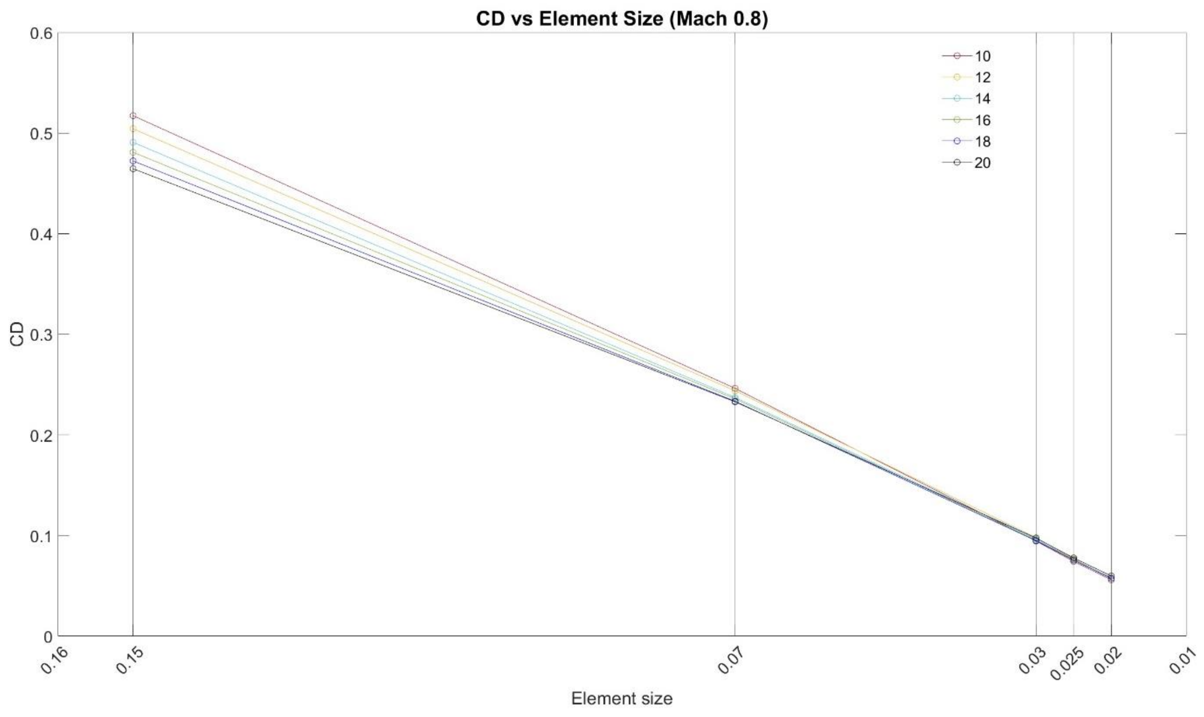


Figure 22 Graph showing the CD (y-axis) against element size (x-axis) from meshes tested at Mach 0.8. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

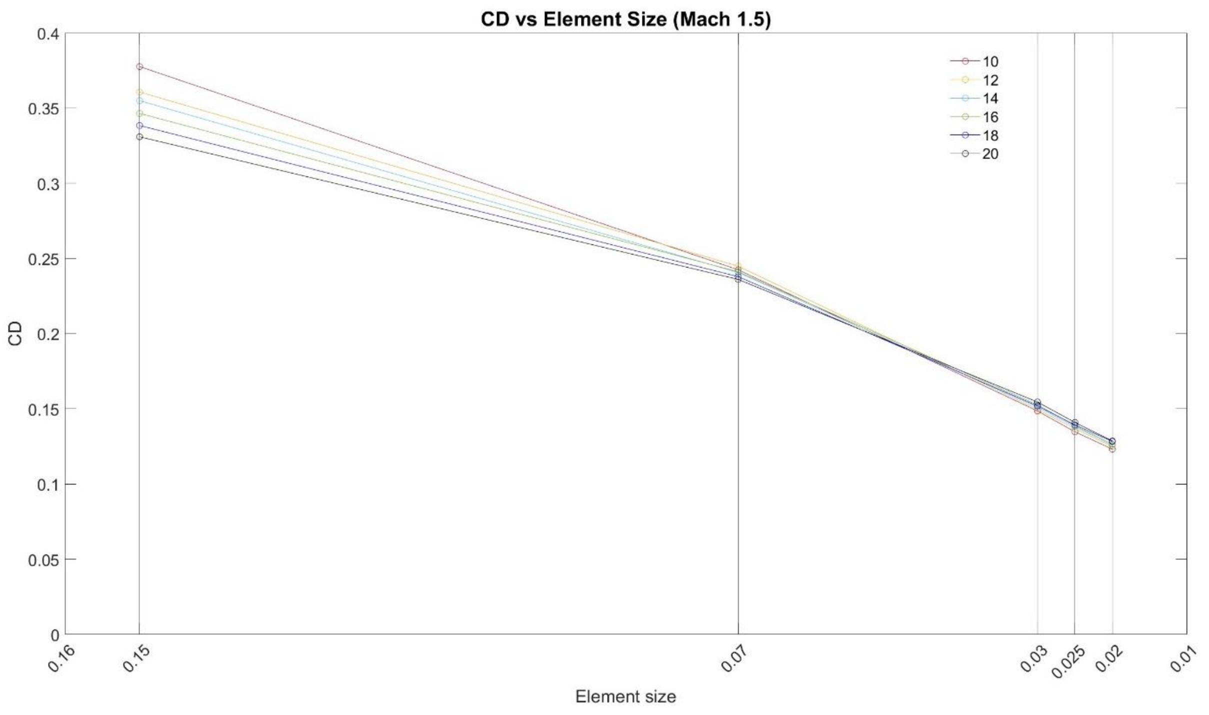


Figure 23 Graph showing the CD (y-axis) against element size (x-axis) from meshes tested at Mach 1.5. Each colour represents a number of layers in the boundary layer, red=10, yellow=12, light blue=14, green=16, dark blue=18, and black=20. Marker lines indicate the element size plotted on the x-axis.

Overall, between 0.025 and 0.02, marked along the x-axis (Figure 18, Figure 19, Figure 20), a partial plateau within the CL values is seen in the lower number of boundary layers. There was no clear conclusion following the CL results and finer mesh testing were not possible due to computing resource limits.

CD values (Figure 21, Figure 22, Figure 23) all follow a similar trend, with the finest mesh of element size 0.02, displaying the lowest drag value. All element size options showing a successful convergence within computing resource limits, but again, only the initial plateau can be seen, therefore no clear conclusion could be drawn. As a result, the chosen mesh was of element size 0.02, the smallest element size option within computing resource limits.

3.1.3.Boundary Layer Exploration & Point Source Implementation

Boundary layer (BL) exploration was increased via:

- Increase of number of layers within the BL
- Alteration of the first layer size
- BL expansion rate

Parameters of the BL included in graphs above can be seen in Table VII.

Table VII Details of boundary layers tested in the mesh convergence study. First row: number of layers in the boundary layer. Second: the size of the first layer in the boundary layer. Third: The default size of the first layer in the boundary layer. Fourth: Rate in which the layers increase in size if larger than layer 0.001. Fifth: Rate in which the layers increase in size if less than layer 0.001.

Layers	10	12	14	16	18	20
First layer	0.001000	0.000250	0.000063	0.000016	0.000004	0.000001
Starting layer	0.001	0.001	0.001	0.001	0.001	0.001
Expansion rate (> 0.001)	2.72	2.72	2.72	2.72	2.72	2.72
Expansion rate (< 0.001)	2	2	2	2	2	2
Layer 1	0.001000	0.000250	0.000063	0.000016	0.000004	0.000001
2	0.002720	0.000500	0.000125	0.000031	0.000008	0.000002
3	0.007398	0.001000	0.000250	0.000063	0.000016	0.000004
4	0.020124	0.002720	0.000500	0.000125	0.000031	0.000008
5	0.054736	0.007398	0.001000	0.000250	0.000063	0.000016
6	0.148883	0.020124	0.002720	0.000500	0.000125	0.000031
7	0.404961	0.054736	0.007398	0.001000	0.000250	0.000063
8	1.101494	0.148883	0.020124	0.002720	0.000500	0.000125
9	2.996065	0.404961	0.054736	0.007398	0.001000	0.000250
10	8.149297	1.101494	0.148883	0.020124	0.002720	0.000500
11		2.996065	0.404961	0.054736	0.007398	0.001000
12		8.149297	1.101494	0.148883	0.020124	0.002720
13			2.996065	0.404961	0.054736	0.007398
14			8.149297	1.101494	0.148883	0.020124
15				2.996065	0.404961	0.054736
16				8.149297	1.101494	0.148883
17					2.996065	0.404961
18					8.149297	1.101494
19						2.996065
20						8.149297

Following preliminary assessments of results gathered from varying number of layers in the boundary layer including the addition of point sources. It was evident, the addition of point sources showed favourable results over the addition of layers in the boundary layers.

The addition of the point sources showed a more significant change within the C_L and C_D values over the addition of layers in the boundary layer. This was the case for all the Mach numbers tested. As a result, the decision was made to maintain the default boundary layer settings, this ensured computational cost was being increased where deemed necessary. Graphs included in Figure 24, Figure 25, and Figure 26, show C_L and C_D data gathered from point source testing, at a varied number of boundary layers.

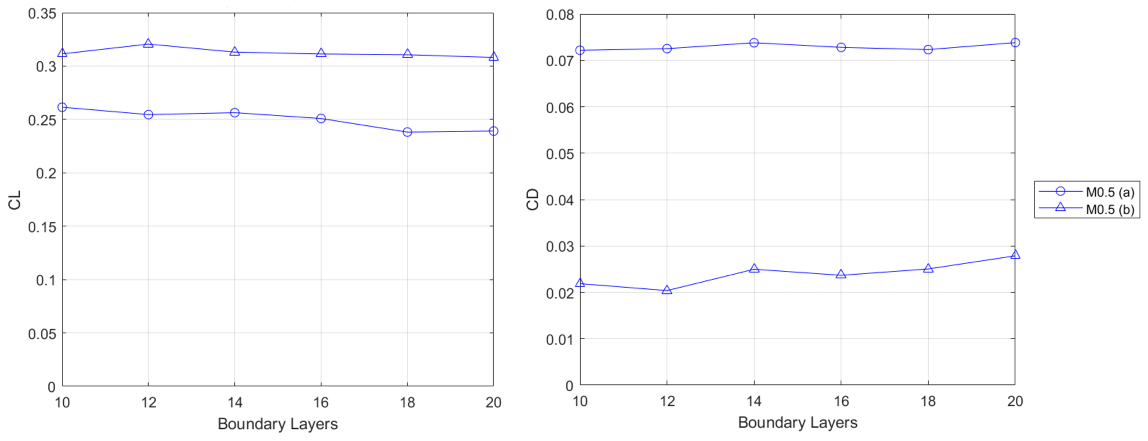


Figure 24 Graph showing CL & CD difference with addition of point source at the leading edge at Mach 0.5 (a) without point source (b) with point source.

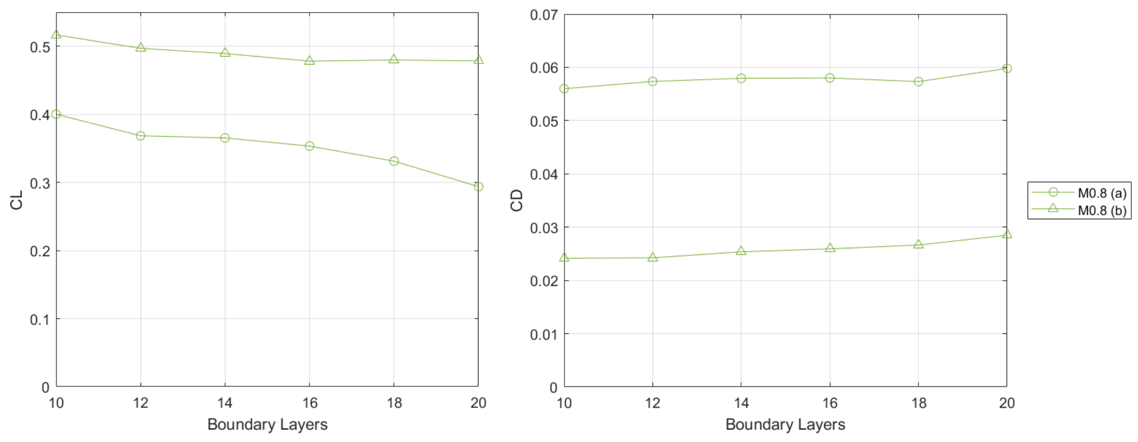


Figure 25 Graph showing CL & CD difference with addition of point source at the leading edge at Mach 0.8 (a) without point source (b) with point source.

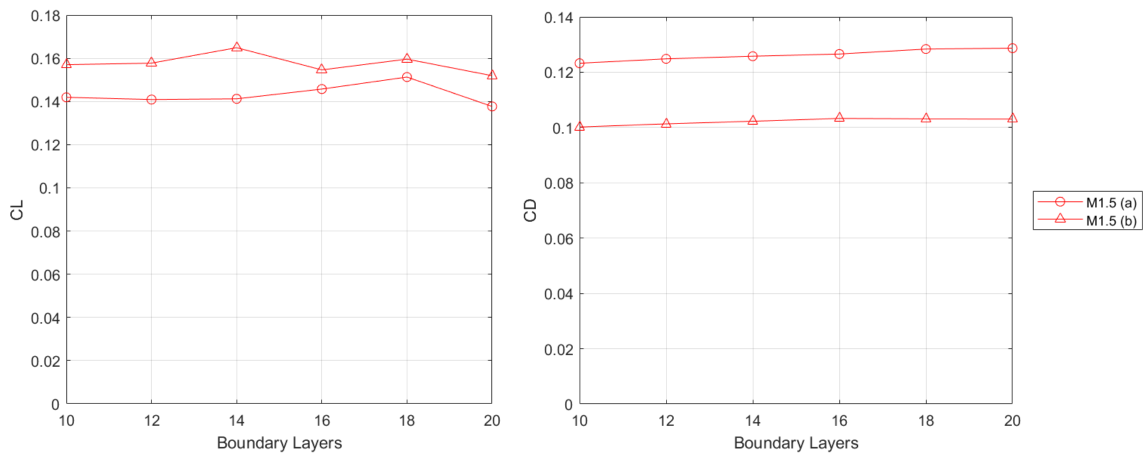


Figure 26 Graph showing CL & CD difference with addition of point source at the leading edge at Mach 1.5 (a) without point source (b) with point source.

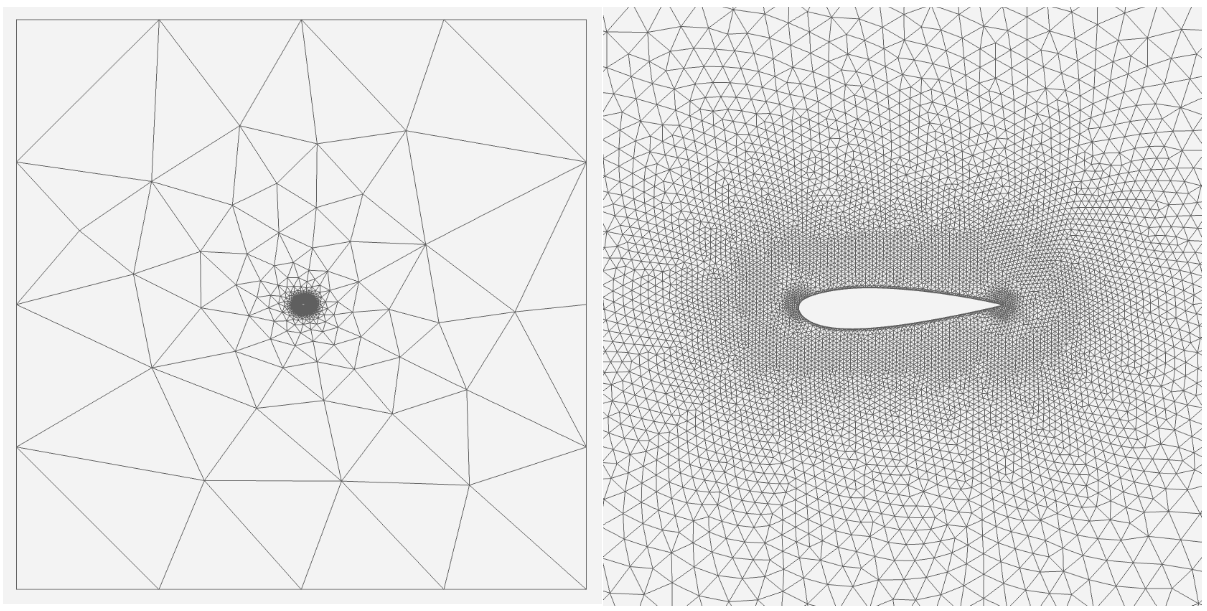


Figure 27 Left: View of entire domain of final mesh. Right: Zoom of final mesh

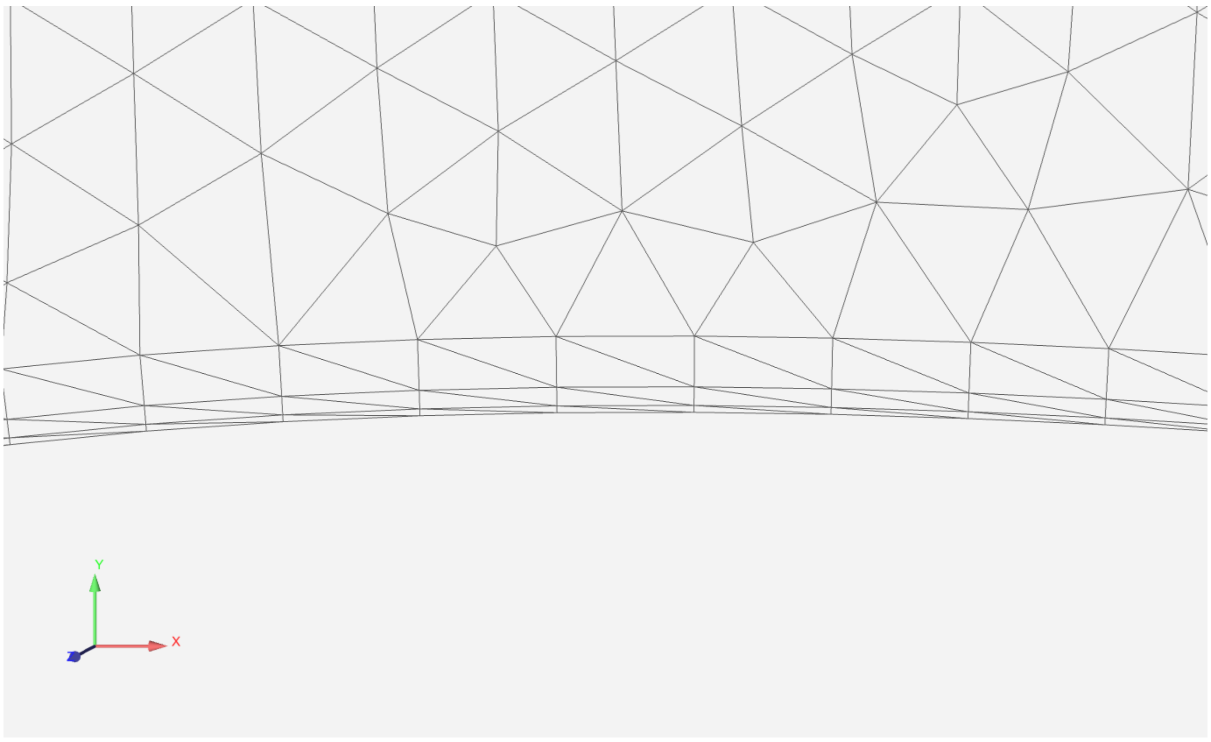


Figure 28 Boundary Layer of final mesh (upper surface snapshot)

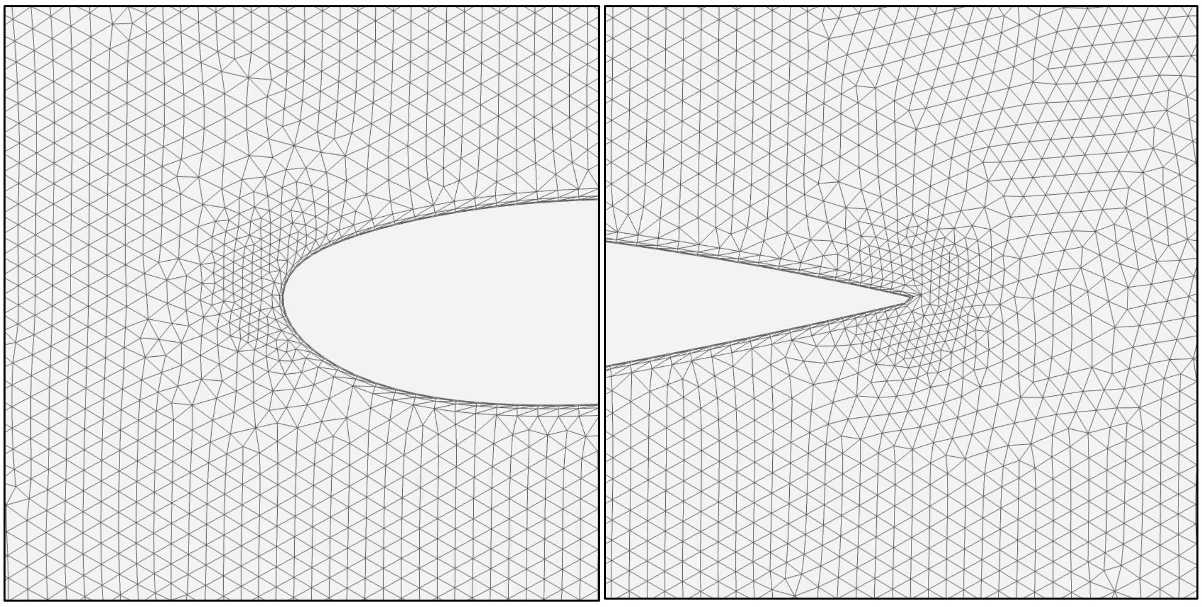


Figure 29 Left: Leading edge final mesh with point source. Right: Trailing edge final mesh with point source

A mesh of element size 0.02 was the smallest mesh resolution sufficient for the entire mesh, applied via a line source following the aerofoil chord line, with the addition of point sources at the leading edge and trailing edge of the aerofoil. These point sources consisted of element size 0.01, providing a higher resolution in locations considered most important for exhibiting larger pressure differentials on the aerofoil, without requiring a larger radius to apply a finer mesh to and exceeding computer resource limits.

Table VII Final mesh parameters

	Element size	0.02
Radius (line source - aerofoil chord line)	*Inner*	0.3
	Outer	0.7
	First layer size	0.001
	No. of layers	10
Point source (leading edge & trailing edge)	*Element size*	0.01
	Radius	0.05
Total no. of	*Elements*	15263
	Nodes	7699
	BL nodes	135

3.2. Optimisation Case Studies

3.2.1. Algorithm Convergence Behaviour

The algorithm's general convergence behaviour is shown through the fitness progression of all the agents inputted into the optimisation. Generally, the best agent is most relevant to results collection, however by looking at the total number of agents, the algorithm's characteristics are noticed, for example how the agents interact with each other and their effectiveness in this, as well as features such as the percentage of agents discarded seen in MCS.

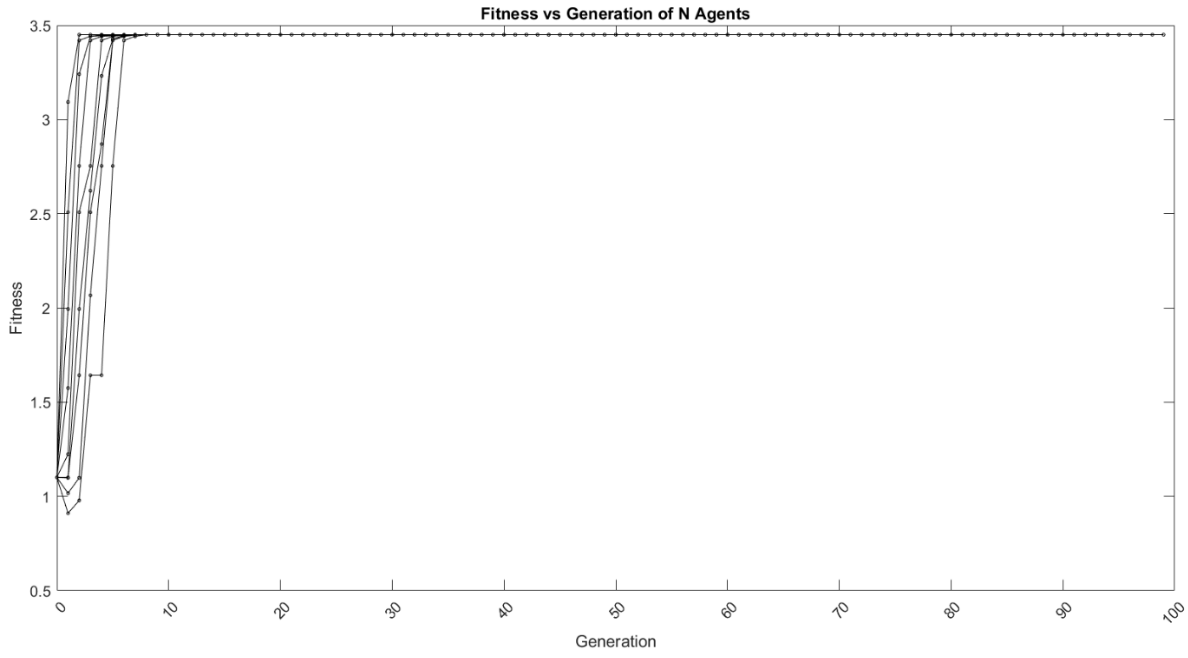


Figure 30 Fitness progression (y-axis) of all 10 agents seen over optimisation length, 99 generations (x-axis). EA - Differential Evolution (case A.1.2)

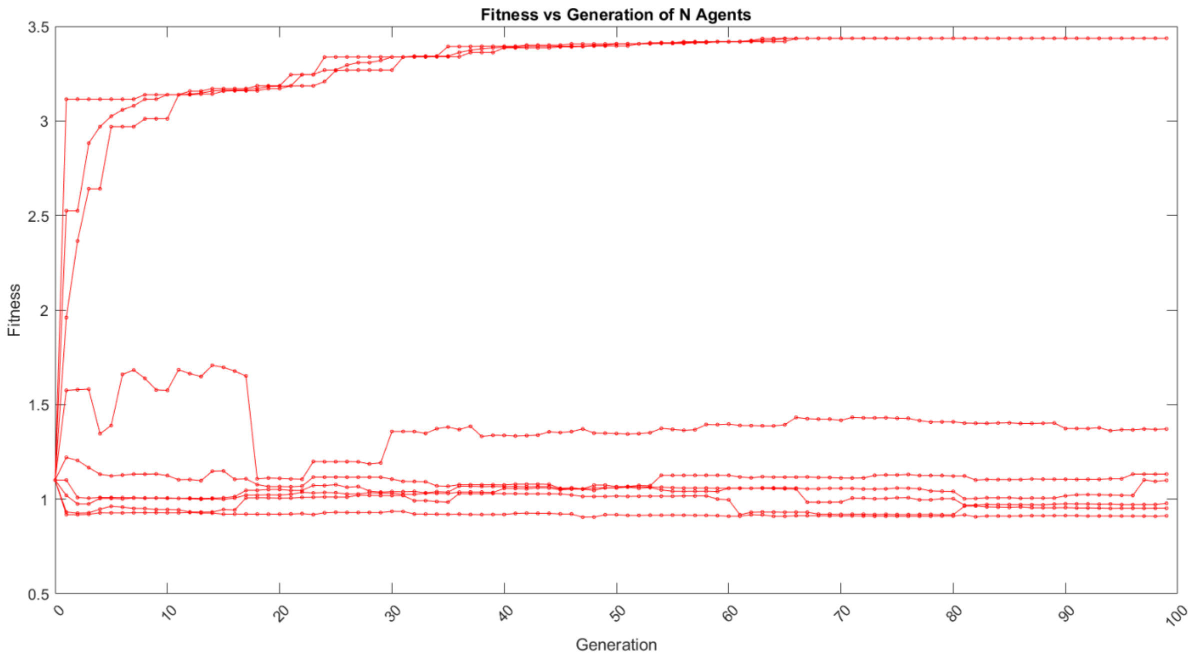


Figure 31 Fitness progression (y-axis) of all 10 agent seen over optimisation length, 99 generations (x-axis). EA - Modified Cuckoo Search (case B.1.2). (Separation seen between the agent's is discussed in the text).

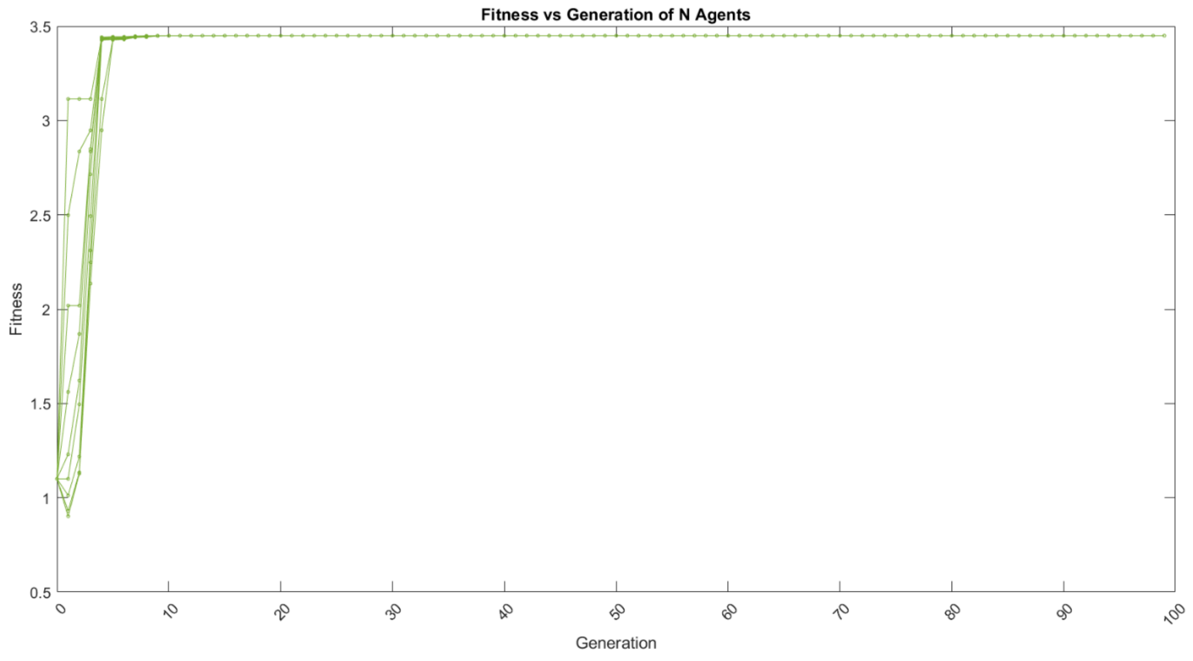


Figure 32 Fitness progression (y-axis) of all 10 agents seen over optimisation length, 99 generations (x-axis). EA - Particle Swarm optimisation (case C.1.2).

Figure 30, Figure 31, and Figure 32 show each agent's fitness increase at every generation, highlighting the overall converging behaviour of the algorithms. From generations 0 to 1, approximately half the agents decrease in fitness, and half increase, this is due to the random sampling of the design space, initialising the optimisation prior to any fitness evaluation.

Note that MCS in figure 43 has 75% if its agents at a significantly lower fitness than the top 25%. This is due to the agent discarding feature described in section 1.4.1. The feature chooses agents within the top 25% in the first generation, after the random sampling. From there, the top agents undergo a different optimising procedure to the agents in the lower 75% fitness category. Since the top agents already possess a higher fitness, the scope in which they search further is narrowed at a faster rate than the lower 75% of agents, this is defined by the Levy Flight step size. The top agents are also allowed to interact with each other, through a process of crossing over their parameters.

Figure 42 and figure 44 show a complete convergence of all agents by generation 10, figure 43 shows a slower convergence and reaches its converged final fitness at generation 76. This slower convergence may be due to its lower number of agents participating in the top search. The top search includes interaction between the agents, when there are less agents, not only is there less chance of finding a higher fitness in the search scope, there is also less information crossed over between the agents. 100% of DE and PSO's agents are interactive, so a faster convergence is usually expected.

Due to the fast convergence seen in each of the three graphs, most significantly in DE and PSO, every point plotted beyond convergence is N number of CFD simulations. Considering the algorithms have successfully converged on a final fitness within the first 10% of the total optimisation time, the remaining optimisation time is identical CFD simulations being run over and over. A commonly-associated aspect across most CFD users is the computational power needed to produce a CFD simulation. When observing the large number of CFD simulations essentially 'wasted' once the algorithm is converged, suggests a stopping criteria may be introduced. This could end the optimisation process once a fitness has been maintained for example 10-20% of the current optimisation time up to that point, preventing further costly simulations being executed for no real benefit to results.

3.2.2.Best Agent Analysis

Figure 33 demonstrates convergence within 5 generations of all agent variations. The top agents were chosen on the terms of their fitness at the final generation was the highest, and the highest final fitness of the three runs was chosen.

All optimisations converged to a margin of 0.003% difference between the top agent and the lowest agent fitness value at the final generation. All optimisations reached a minimum fitness of 3.45 by generation 5, with only minor adjustments to the fitness value in subsequent generations. Minor adjustments consisted of a total of 0.03%, slight adjustments such as this, were considered background noise caused by the CFD solver and are considered insignificant.

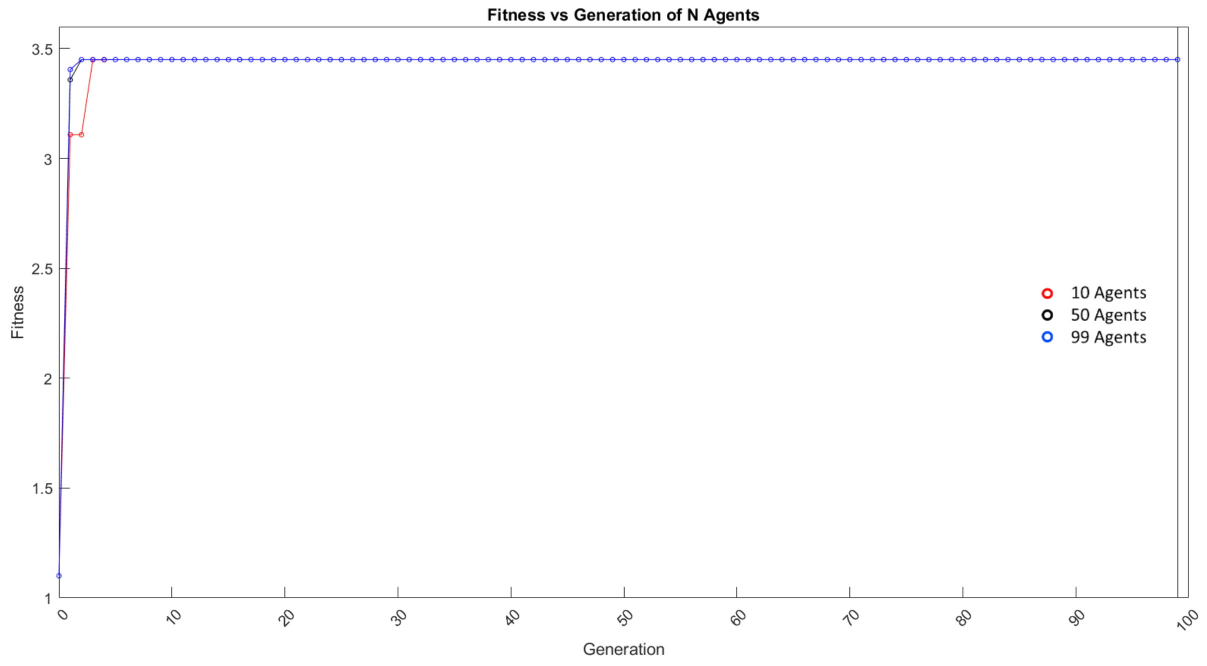


Figure 33 The best agent from three varying agent number (see legend), Differential Evolution optimisations, all at Mach 0.8

Each data point on the graph in Figure 33 represents a CFD simulation. The red points are equivalent to 10 CFD simulations, black represents 50, and blue, 99. After generation 5, the optimisation is considered converged, leaving 94 generations of unused CFD simulations. Sufficient result collection will likely indicate the number of generations needed, in this case, 10 generations was plenty.

The top agent from the 99 agent optimisation (blue) showed the fastest increase in fitness and achieved a final fitness of 3.45 in generation 3, the top agent out of 50 agents (black) also reached the final fitness at generation 3, and it took another generation for the top agent out of 10 (red). Therefore, the assumption can be made that the more agents, the faster the final fitness value will be found, this however, comes at the expense of computational power, and would only be effective where computational resources are not exceeded.

For maintaining lower computational cost, DE finds the final fitness relatively quickly with regards to a high number of agents, such as 99 or 50. A lower number of agents, such as 10 would still provide a sufficient optimisation.

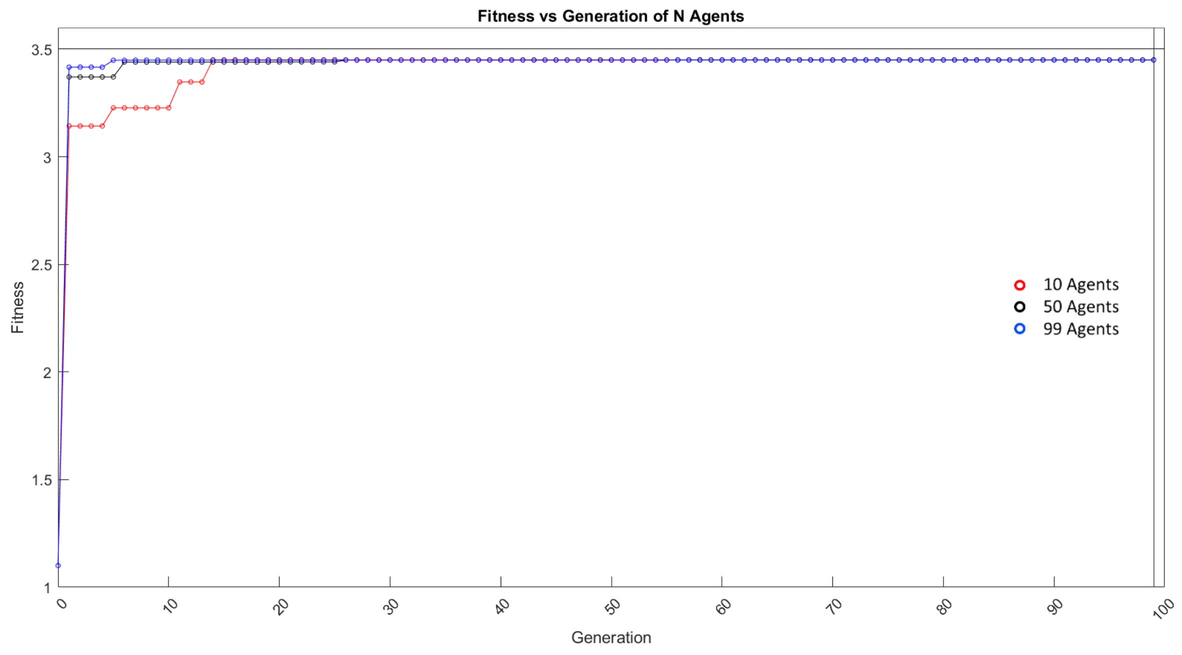


Figure 34 The best agent from three varying agent number (see legend), Modified Cuckoo Search optimisations, all at Mach 0.8

In Figure 34, MCS observes a slower convergence than DE. MCS was designed with the application of multiple degrees of freedom in mind [21][22], where there is only two degrees of freedom, it is expected that DE would show more capability within these case studies. All optimisations reached a final fitness of 3.45 by generation 14, requiring 9 more generations to match the final fitness of DE.

MCS showed multiple instances where the agent did not improve its fitness value from one generation to the next. This is likely due to the top agent not finding a higher fitness within its random search. Instead, it relies on another agent finding a higher fitness, in which it will breed with (see section 1.4.1), resulting in an increase of fitness. Stationary fitness from one generation to the next is seen more commonly with a lower agent number, as there are less agents searching for a higher fitness, thus less probability one will be found.

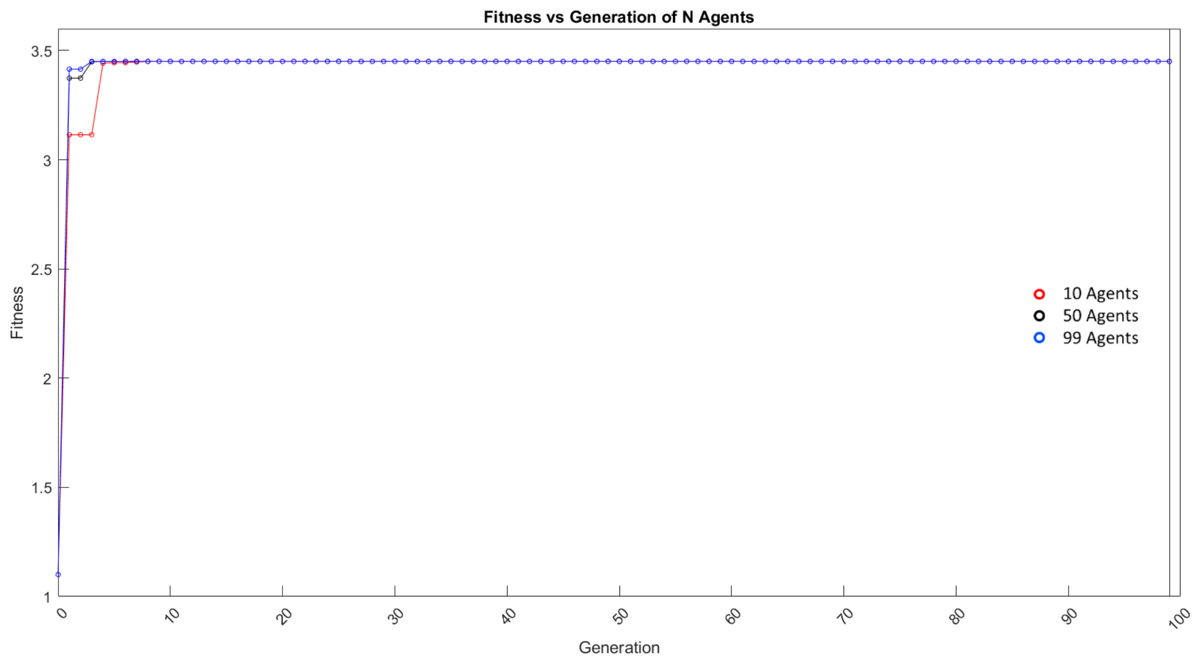


Figure 35 The best agent from three varying agent number (see legend), Particle Swarm Optimisation optimisations, all at Mach 0.8

PSO (Figure 35) shows a similar fitness progression to DE in Figure 33, reaching the top fitness by generation 5 for all agent variations. PSO shows a delay of approximately one generation relative to the top agents in the DE optimisation, with the 99 agent optimisation requiring an extra generation to reach the final fitness, as well as the 10 agent optimisation. This may be due to random selection, or possibly the nature in which the agents interact. As described in section 1.4.3, PSO combines agent's current velocity, split into its magnitude and direction. The degree in which magnitude and direction are combined is dependent on user-preference and will define how quickly the agents converge. Within AerOpt, the PSO algorithm is set within the software and would require scope beyond the project to implement changes into the algorithm's default settings.

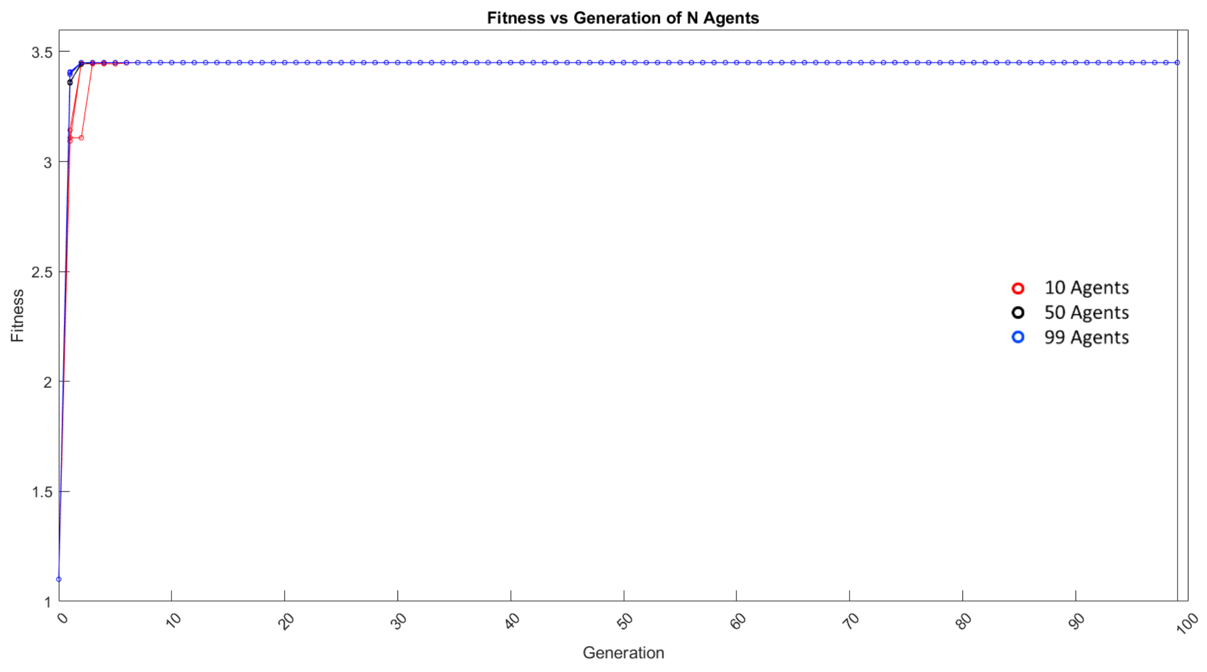


Figure 36 The best agent from all three runs of identical input parameters. Differential Evolution, at Mach 0.8.

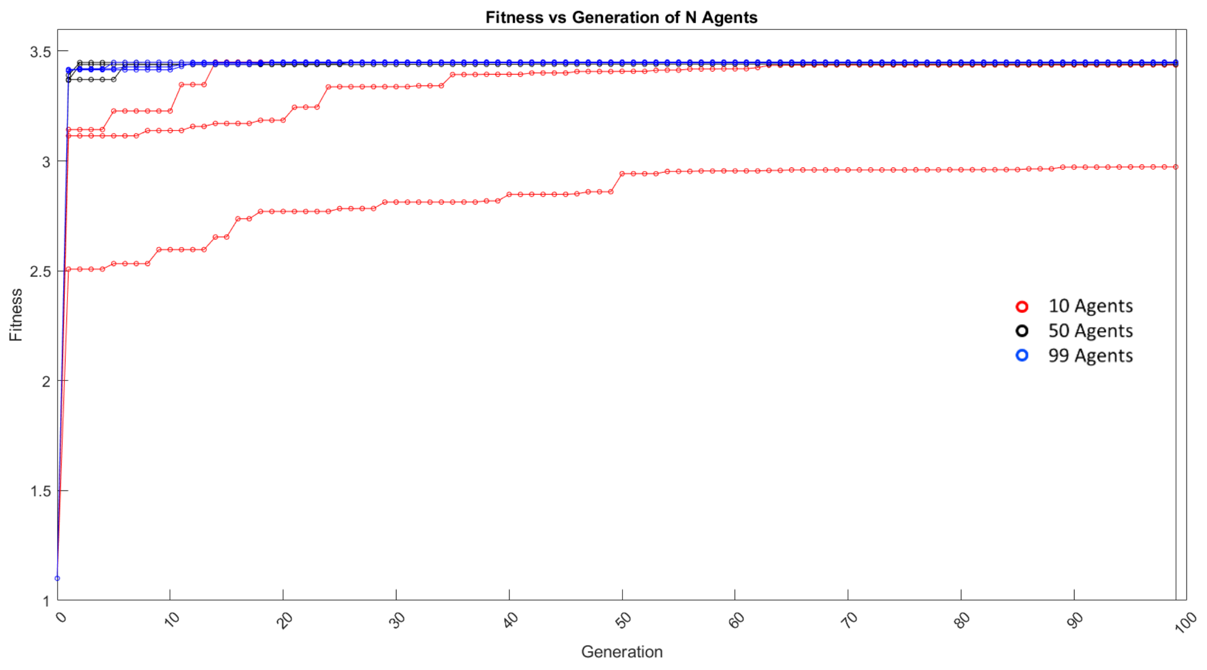


Figure 37 The best agent from all three runs of identical input parameters. Modified Cuckoo Search, at Mach 0.8.

Figure 36 and Figure 37 include three runs of identical input parameters for all three algorithms at the corresponding Mach numbers (colour indicated as above). Both figures demonstrate how including more agents increases the chance of finding a high fitness through random search. The blue markers show the highest fitness at generation 1 in both figures, followed by the black markers, followed by red. DE shows a faster rate of fitness improvement than MCS, with a 10 agent MCS run never reaching the final fitness value of 3.45. Both DE and MCS show similar fitness progression within the 50 and 99 agent runs, however less agents tend to present a more significant effect on MCS ability to find a fitness of high value quickly. This is likely due to the ‘percentage of agents discarded’ feature of MCS has when less agents are inputted into the initial optimisation, leaving a 10 agent optimisation reduced to 3 interactive agents, amounting to 25% of the other algorithms agents.

3.2.3. Control Node Movement

Figure 38, Figure 39, and Figure 40 show where each algorithm placed the best agents throughout the optimisation. Each optimisation consisted of 99 generations; 99 data points cannot be seen plotted in each data set as it is likely the best agent did not move positions for a number of generations during periods of the optimisation.

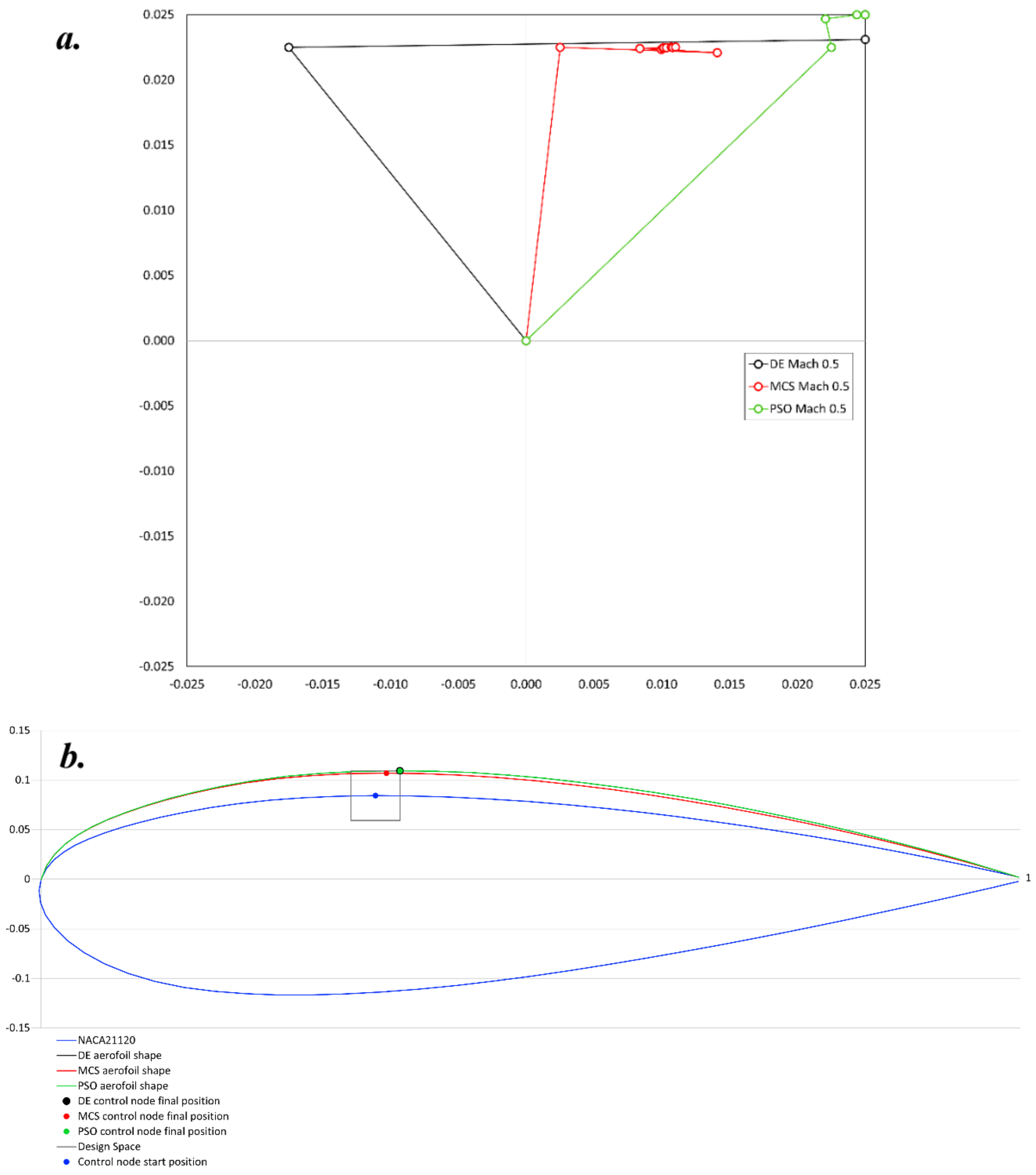


Figure 38 ***(a)*** *Position of the best agent within the control node design space through 99 generations, starting position (0,0). Mach 0.5. Design space constrained to the displayed axis values, 0.025 and -0.025.*
(b) *Results of the control node movement on the overall aerofoil shape of the Mach 0.5 cases. Original aerofoil shape is shown in blue.*

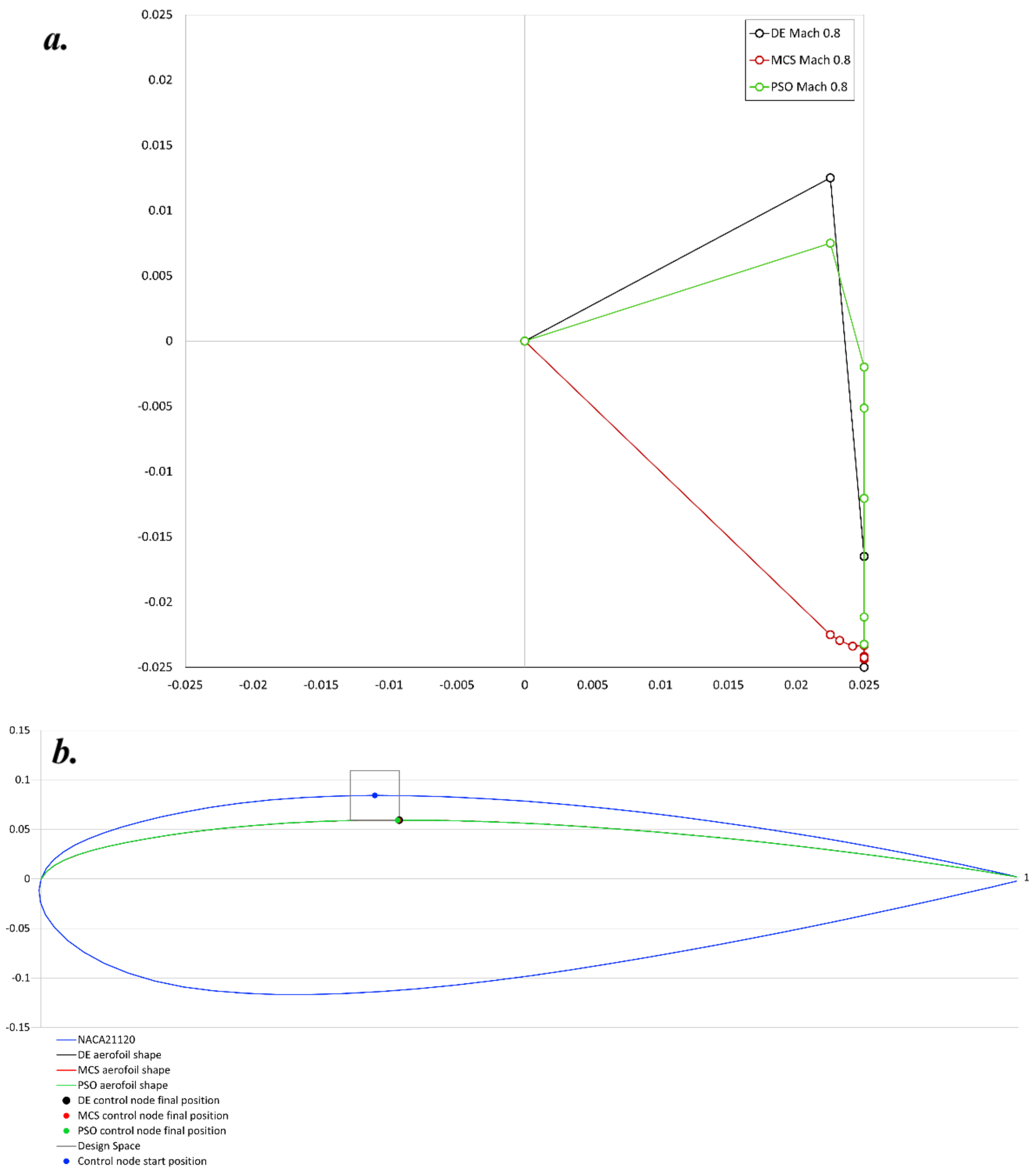


Figure 39 ***(a)*** *Position of the best agent within the control node design space through 99 generations, starting position (0,0). Mach 0.8. Design space constrained to the displayed axis values, 0.025 and -0.025.*
(b) *Results of the control node movement on the overall aerofoil shape of the Mach 0.8 cases. Original aerofoil shape is shown in blue.*

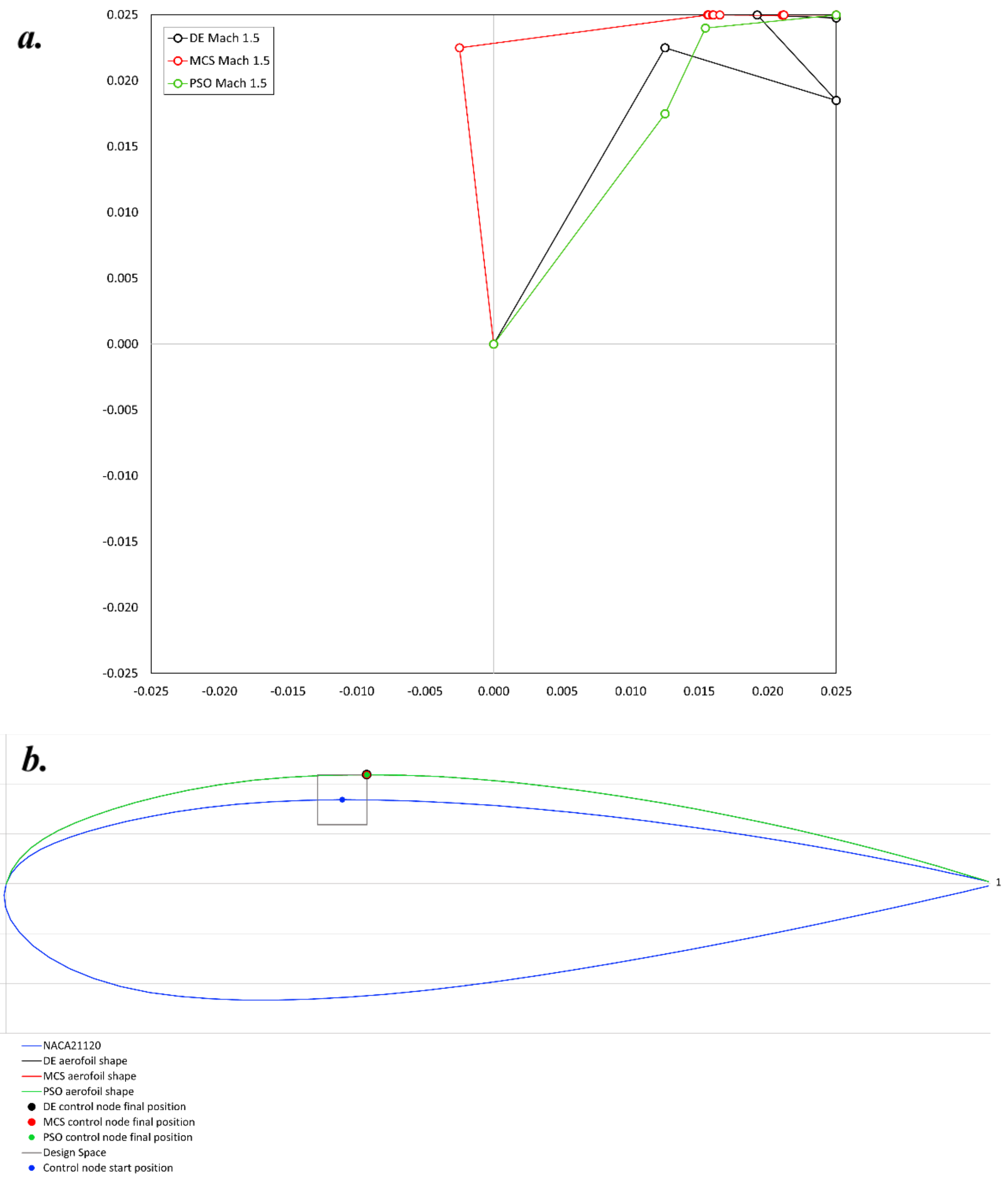


Figure 40 ***(a)*** *Position of the best agent within the control node design space through 99 generations, starting position (0,0). Mach 1.5. Design space constrained to the displayed axis values, 0.025 and -0.025.*
(b) *Results of the control node movement on the overall aerofoil shape of the Mach 1.5 cases. Original aerofoil shape is shown in blue.*

In all three of the plots, the agent makes the largest movement from the initial position to the first position. The first position is a randomly generated position, this position is due to a particular agent being placed in the design space closest to where the algorithm deems optimum, hence why it is categorised as the ‘best’ agent. From generation 1 onwards, the jumps in positions tend to get smaller, in line with the jumps in fitness seen in previous plots. The smaller jumps tend to be seen since the algorithm has found close to the optimum position through the random search and therefore only small further adjustments in the position are necessary.

DE and PSO appear to show the least number of points in which the agent was positioned before reaching the optimum. MCS showed smaller increments throughout the optimisation so took more generations before it reached its final position, its final position matched the final positions of DE and PSO in the latter two cases, but not in the Mach 0.5 optimisation. This may have been down to random sampling, or simply MCS was not as suited to the input parameters as DE and PSO, so could not find an optimum of as high fitness.

The pressure plots show the final geometry produced by PSO at Mach 0.5, 0.8, and 1.5. It was decided to use PSO's results to display Mach number geometry variations as PSO tended to move the control node most significantly within the design space, representing an overall geometry change more clearly. Mach 0.5 and 1.5, both show the control node being positioned in the top-right corner of the design space, resulting in an increase in the aerofoil thickness. The control node was placed in the bottom-right corner of the design space for Mach 0.8, showing a decrease in the aerofoil thickness. The final control node placement results were mostly consistent across all three algorithms, with MCS showing a slightly more conservative control node movement, most notable in Figure 38.

Figure 41, Figure 42, and Figure 43 represent the path of the best agent placement within the control node design space across 99 generations. Each graph represents an EA with varying number of agents inputted, stated in the legend. As mentioned previously, a movement (denoted by a data point) is not seen in every generation since it is likely either the control node remains unmoved for a number of generations, and/or the final position is found at an earlier generation, so any subsequent generations show no control node movement.

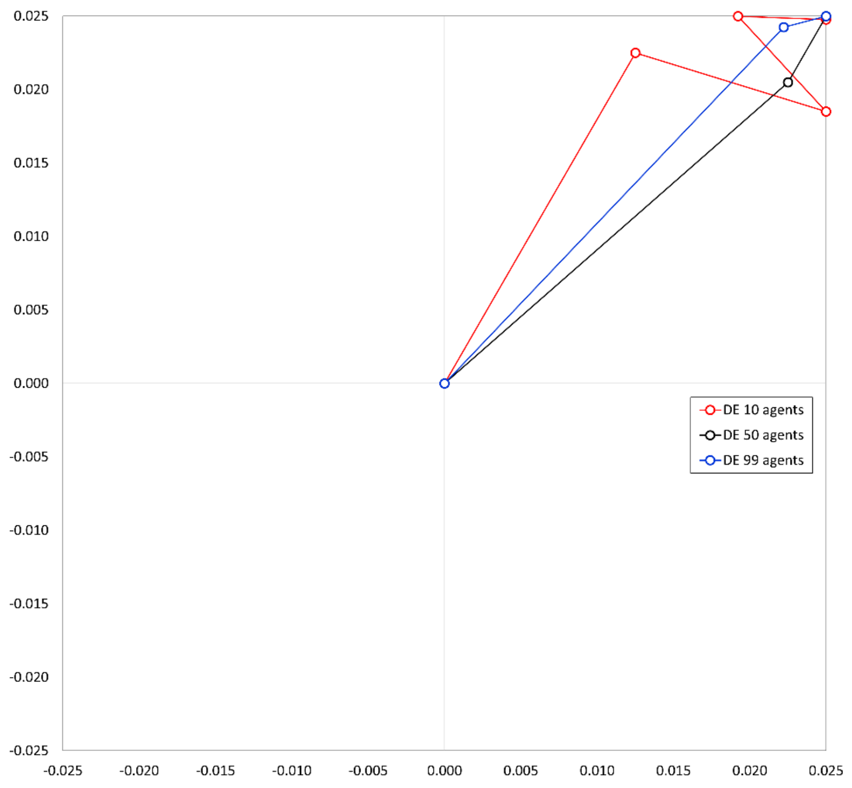


Figure 41 Position of best agent within the control node design space. Mach 1.5, EA- Differential Evolution, where the legend colours indicate the number of agents- red is 10 agents, black is 50 agents, and blue is 99 agents.

Figure 41 shows how the algorithm searches and narrows down the control node's position in the design space; seen in the 10 agent case (red), the control node's position 'jumps' around the top-right quadrant, and as the generations increase, the jumps become smaller.

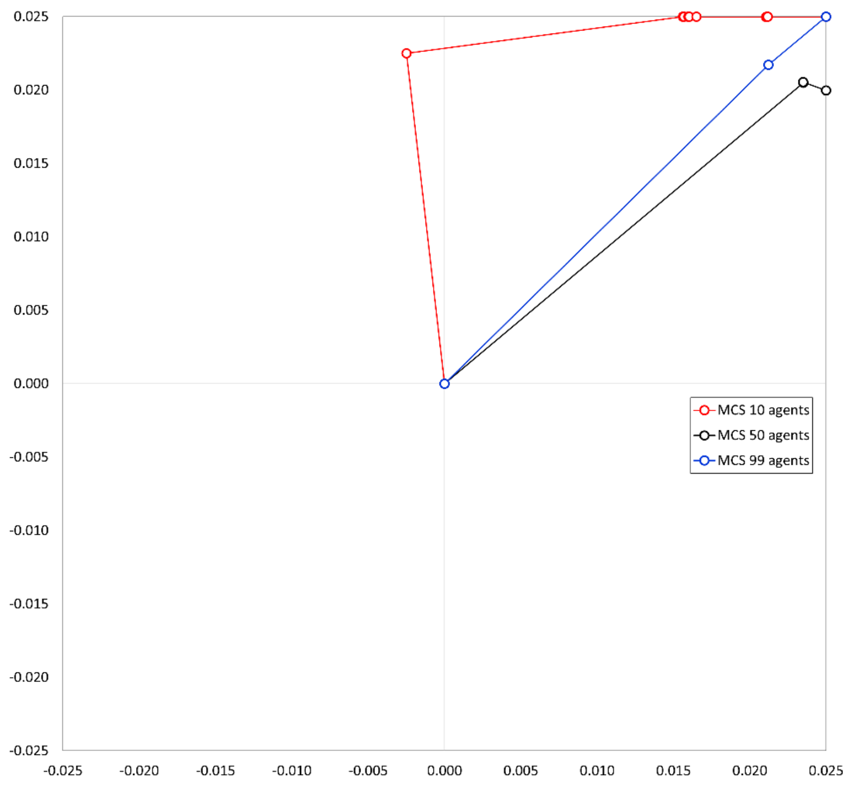


Figure 42 Position of best agent within the control node design space. Mach 1.5, EA – Modified Cuckoo Search, where the legend colours indicate the number of agents- red is 10 agents, black is 50 agents, and blue is 99 agents.

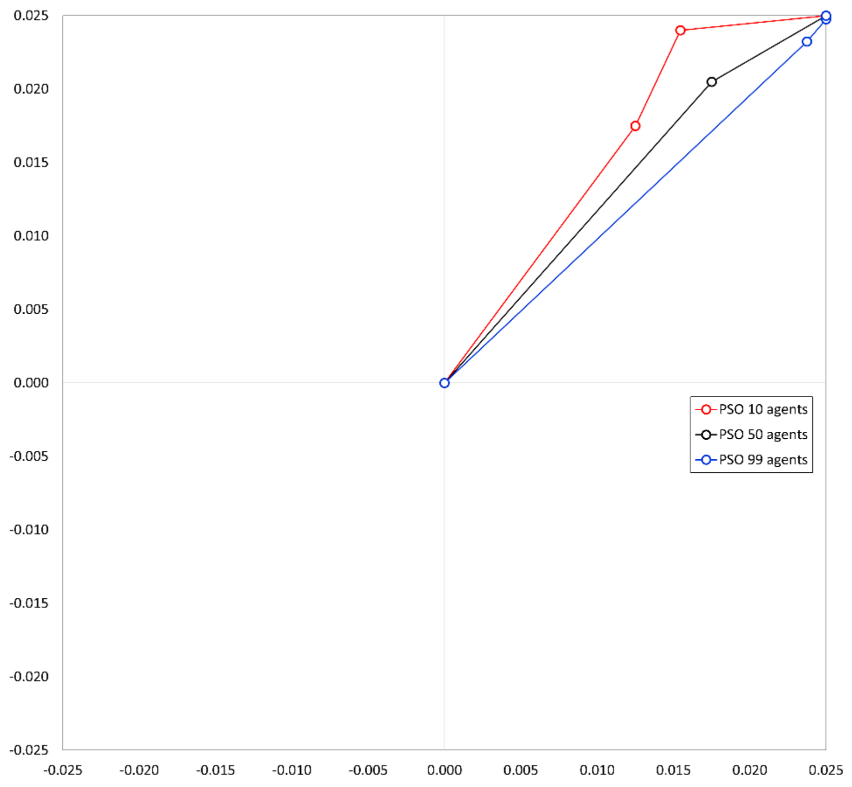


Figure 43 Position of best agent within the control node design space. Mach 1.5 EA – Particle Swarm Optimisation, where the legend colours indicate the number of agents- red is 10 agents, black is 50 agents, and blue is 99 agents.

The three figures demonstrate how adding more agents tends to decrease the number of ‘jumps’ the control node requires before settling at its final position. Figure 43 is an example of a 10 agent optimisation requiring three major movements, when increased to 50 agents, the optimisation makes two major movements, and 99 agents is one. Increasing the number of agents, increases the probability that an agent will be positioned closer to the ‘optimum’ position by random chance, leaving less adjustment to reach the final position, this is somewhat seen in the graphs above.

The data seen in all control node plots shows the algorithms positioning the control node at a far corner of the design space, suggesting that had the design space been increased, the algorithms may had placed the control node further out, as their ‘optimum’ position with a potentially higher fitness. Yet, increasing the design space, comes with increasing computational cost, and to assess the suitable design space size, and where to place it for a given problem, would be a task in itself.

Table VIII Summary of test cases A-C including their final fitness results and fitness percentage increase

Case	*Optimising Algorithm*			*Mach*	*Agents*	*Fitness of best agent*							
						Initial	*Final*			*% Increase*			*Av % Increase*
							Run 1	*Run 2*	*Run 3*				
A	Differential Evolution	A.1	A.1.1	0.5	10	-11.00	-1.88	-1.88	-1.88	82.95	82.95	82.95	82.95
			A.1.2	0.8		1.10	3.45	3.45	3.45	213.46	213.46	213.46	213.46
			A.1.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
		A.2	A.2.2	0.8	50	1.10	3.45	3.45	3.45	213.46	213.46	213.46	213.46
			A.2.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
		A.3	A.3.2	0.8	99	1.10	3.45	3.45	3.45	213.46	213.47	213.47	213.46
			A.3.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
B	Modified Cuckoo Search	B.1	B.1.1	0.5	10	-11.00	-2.44	-2.44	-2.44	77.79	77.79	77.79	77.79
			B.1.2	0.8		1.10	3.45	2.97	3.44	213.46	170.09	212.23	198.59
			B.1.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
		B.2	B.2.2	0.8	50	1.10	3.44	3.45	3.45	212.54	213.31	213.35	213.07
			B.2.3	1.5		0.21	0.27	0.27	0.27	32.25	32.23	30.69	31.72
		B.3	B.3.2	0.8	99	1.10	3.45	3.45	3.45	213.46	213.10	213.46	213.34
			B.3.3	1.5		0.21	0.27	0.28	0.28	31.71	33.44	33.44	32.87
C	Particle Swarm Optimisation	C.1	C.1.1	0.5	10	-11.00	-1.88	-1.88	-1.88	82.90	82.90	82.90	82.90
			C.1.2	0.8		1.10	3.45	3.45	3.45	213.48	213.48	213.48	213.48
			C.1.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
		C.2	C.2.2	0.8	50	1.10	3.45	3.45	3.45	213.48	213.48	213.48	213.48
			C.2.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44
		C.3	C.3.2	0.8	99	1.10	3.45	3.45	3.45	213.48	213.48	213.48	213.48
			C.3.3	1.5		0.21	0.28	0.28	0.28	33.44	33.44	33.44	33.44

In comparison of the A – C test cases, where various Mach numbers were tested, the transonic flow had the largest initial fitness, meaning generally the NACA21120’s geometry was best suited for this flow regime. The algorithms also showed the most ‘effective’ fitness improvement for the aerofoil at transonic speeds; it was not the largest improvement however, which was seen in the subsonic cases. The reason the transonic was more ‘effective’ for the aerodynamic context, is the final fitness of the subsonic cases was still negative regardless of the negative initial fitness, meaning the end geometry was still producing a higher drag value than lift, this was not the case for the transonic cases. Generally, the initial aerofoil was less suited to subsonic and supersonic flow, which had little to do with the algorithm’s optimising capabilities, and more to do with the suitability of the aerofoil’s geometry for the supersonic flow regime. It is suggested the NACA21120 was designed with the transonic flow regime in mind, hence the results seen in Table VIII.

Table IX Summary of cases and their wall-clock time

Case	*Optimising Algorithm*	*Agents*			*Time*							
					Run 1		*Run 2*		*Run 3*			*Average*
					Minutes	*Hours*	*Minutes*	*Hours*	*Minutes*	*Hours*	*Minutes*	*Hours*
A	Differential Evolution	A.1	10	A.1.1	1270	21.2	1270	21.2	1270	21.2	1270	21.2
				A.1.2	104	1.7	104	1.7	108	1.8	105	1.8
				A.1.3	61	1.0	56	0.9	57	1.0	58	1.0
		A.2	50	A.2.2	130	2.2	123	2.1	124	2.1	126	2.1
				A.2.3	124	2.1	123	2.1	137	2.3	128	2.1
		A.3	99	A.3.2	149	2.5	150	2.5	148	2.5	149	2.5
				A.3.3	149	2.5	151	2.5	183	3.1	161	2.7
B	Modified Cuckoo Search	B.1	10	B.1.1	1169	19.5	1170	19.5	1169	19.5	1169	19.5
				B.1.2	105	1.8	104	1.7	107	1.8	105	1.8
				B.1.3	55	0.9	56	0.9	55	0.9	55	0.9
		B.2	50	B.2.2	124	2.1	124	2.1	125	2.1	124	2.1
				B.2.3	81	1.4	78	1.3	76	1.3	78	1.3
		B.3	99	B.3.2	148	2.5	121	2.0	213	3.6	161	2.7
				B.3.3	156	2.6	146	2.4	149	2.5	150	2.5
C	Particle Swarm Optimisation	C.1	10	C.1.1	1261	21.0	1260	21.0	1264	21.1	1262	21.0
				C.1.2	108	1.8	106	1.8	106	1.8	107	1.8
				C.1.3	62	1.0	60	1.0	59	1.0	60	1.0
		C.2	50	C.2.2	119	2.0	122	2.0	128	2.1	123	2.1
				C.2.3	81	1.4	83	1.4	83	1.4	82	1.4
		C.3	99	C.3.2	150	2.5	151	2.5	154	2.6	152	2.5
				C.3.3	178	3.0	170	2.8	169	2.8	172	2.9

It is commonly seen in Table IX the larger the Mach number, the shorter the wall-clock time, this is due to the CFD solver used by AerOpt, designed for higher speed flows, predominantly in the transonic and supersonic regime.

Due to the nature in which EAs lend themselves well to high performance computing, increasing the number of agents had little to no effect on the wall-clock time with cluster runs. Generally, MCS had the shortest wall-clock time throughout the cases seen, followed by PSO, followed by DE. There is no definitive reason for this observation and may have been due to a number of reasons. MCS includes a feature of ‘percentage of agents discarded’, meaning a certain percentage of agents (75% in these particular test cases) do not interact with each other. It may be assumed the crossover process that PSO and DE perform with all their agents, leads to the increase in total time taken for the optimisation. However, it is important to note that, due to the ‘percentage of agents discarded’ feature, 75% of the agents are being randomly generated positions with no other informative input at every generation, meaning it is likely a lot of the discarded agents are generating poor geometries regarding fitness values. With a poor geometry, comes a more disjointed procedure for the mesh parameterisation and CFD system to solve, consequently, taking more time. These two contrasting reasons may be the reason for MCS showing a relatively insignificant decrease in wall-clock time compared to PSO and DE.

Table X A comparison of average wall-clock time for test cases with identical input parameters run locally and on cluster.

Case	Optimising Algorithm		Agents	Generations		Mach	Wall-clock time			
							Local		Cluster	
							Minutes	Hours	Minutes	Hours
A	Differential Evolution	A.1	10	99	A.1.1	0.5	10464	174.4	1270	21.2
					A.1.2	0.8	2950	49.2	105	1.8
					A.1.3	1.5	644	10.7	58	1.0
B	Modified Cuckoo Search	B.1	10	99	B.1.1	0.5	10860	181.0	1169	19.5
					B.1.2	0.8	990	16.5	105	1.8
					B.1.3	1.5	604	10.1	55	0.9
C	Particle Swarm Optimisation	C.1	10	99	C.1.1	0.5	10999	183.3	1262	21.0
					C.1.2	0.8	1218	20.3	107	1.8
					C.1.3	1.5	604	10.1	60	1.0

Table X displays a comparison of the wall-clock time taken for optimisations inputted with 10 agents and 99 generations. Local optimisations took on average approximately ten times longer than optimisations run on cluster. With executing optimisations locally, the core processing power is lower, therefore some number of agents have to be run in series as well as generations, lengthening the total optimisation time. Running on cluster eradicates the need to run any agents in series, leaving only the number of generations to define the length of the optimisation.

It is important to note that some wall-clock times appear as anomalies compared to others, such as the locally run test case A.1.2, this likely is due to the particular optimisation encountering more cumbersome CFD geometries to solve. All the EAs undergo random sampling, if some of these samples provide a poorer than usual geometry, it may take longer for the mesh and CFD solver to implement solutions, increasing the overall wall-clock time.

3.2.4.Exploring Multiple Control Node Movement

As mentioned in section 3.2.2, during development, MCS was predicted to show better performance when a particularly large number of degrees of freedom is applied [58]. Previous testing in section 3.2.2 involved 2 degrees of freedom, a single moveable control node with movement in the x and y axis, assessing only the ‘parameter tuning’ aspect of the algorithm’s behaviour. With the increase in degrees of freedom, the algorithms performance is analysed with respect to number of degrees of freedom.

Table XI Test cases for exploring control node placement

Case	Algorithm	Mach	Reynolds no.	Agents	Generations
D	Differential Evolution	1.5	3.62E+07	10	99
E	Modified Cuckoo Search	1.5	3.62E+07	10	99
F	Particle Swarm Optimisation	1.5	3.62E+07	10	99

Figure 44 shows a significant difference between the algorithm’s performance when observing the best agent out of 10 agents. Seen in green, PSO found the highest final fitness, followed by DE in black, then MCS in red.

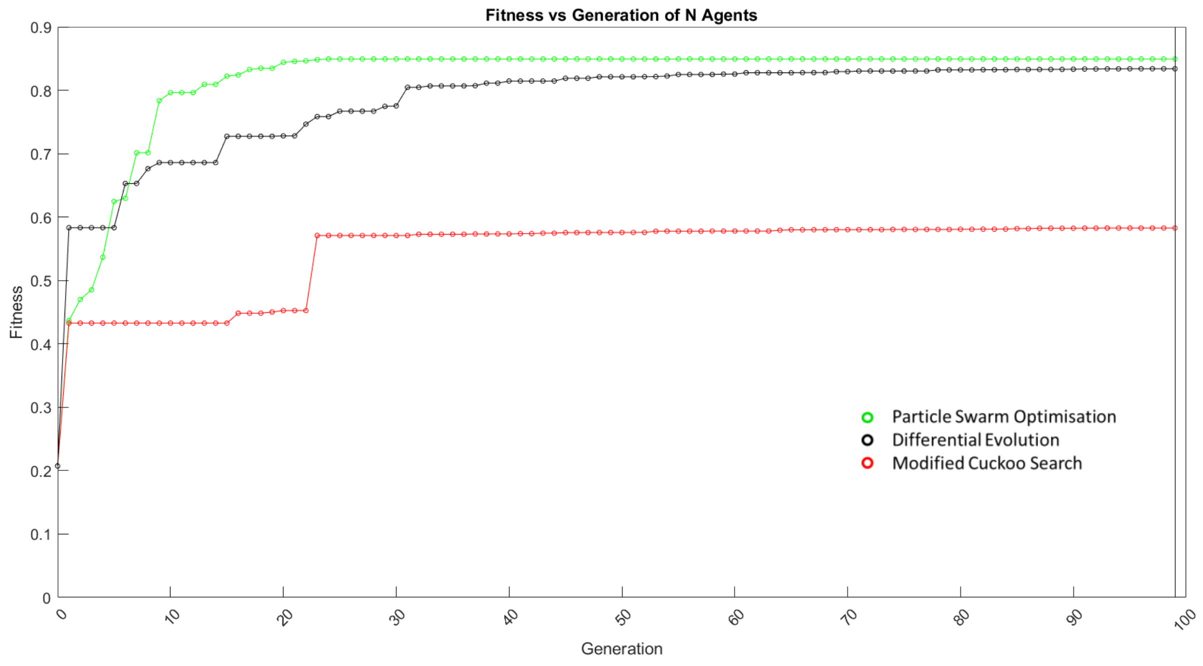


Figure 44 The best agent's fitness progression from each algorithm's optimisation.

PSO appears to reach its final fitness at a faster rate than MCS, with DE showing the slowest rate of fitness increase. Due to the increase in control nodes, the size of the optimisations total design space is increased, allowing a much larger area for the algorithms to randomly search. Leading to the assumption that the randomising element of each algorithm has a greater influence on the overall fitness progression and may define a better outcome for some optimisations over others, purely by chance.

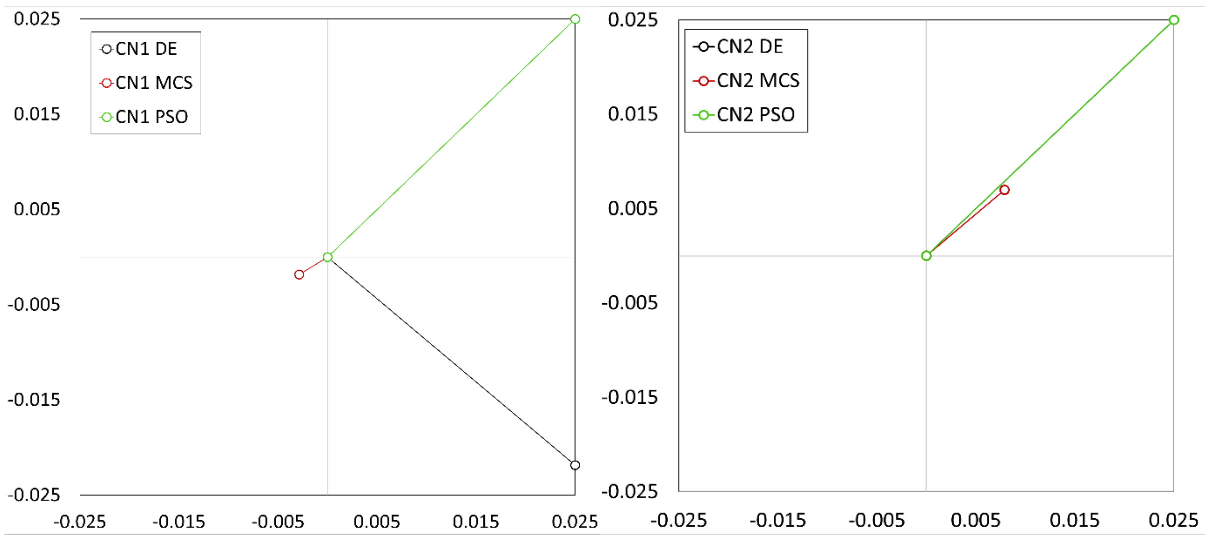


Figure 45 Best agent within control nodes 1 & 2 where the origin is their initial position

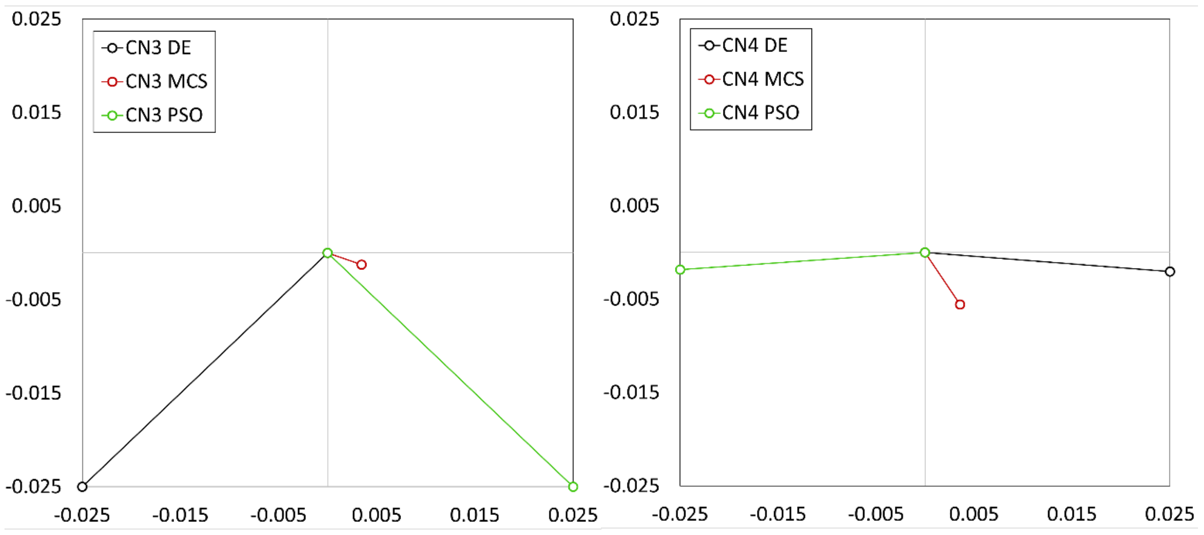


Figure 46 Best agent within control nodes 3 & 4 where the origin is their initial position

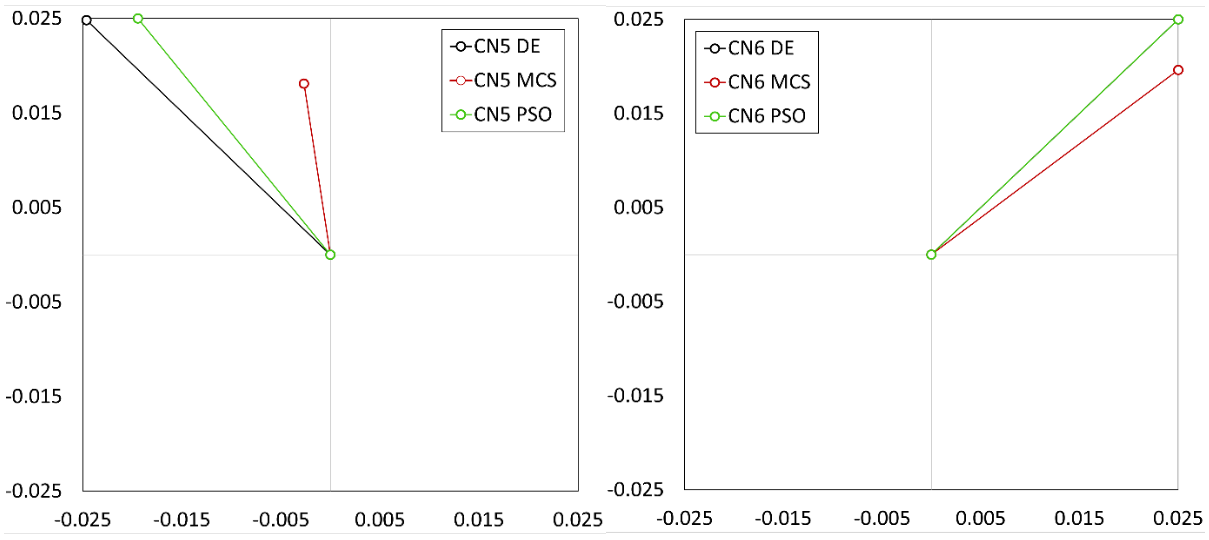


Figure 47 Best agent within control nodes 5 & 6 where the origin is their initial position

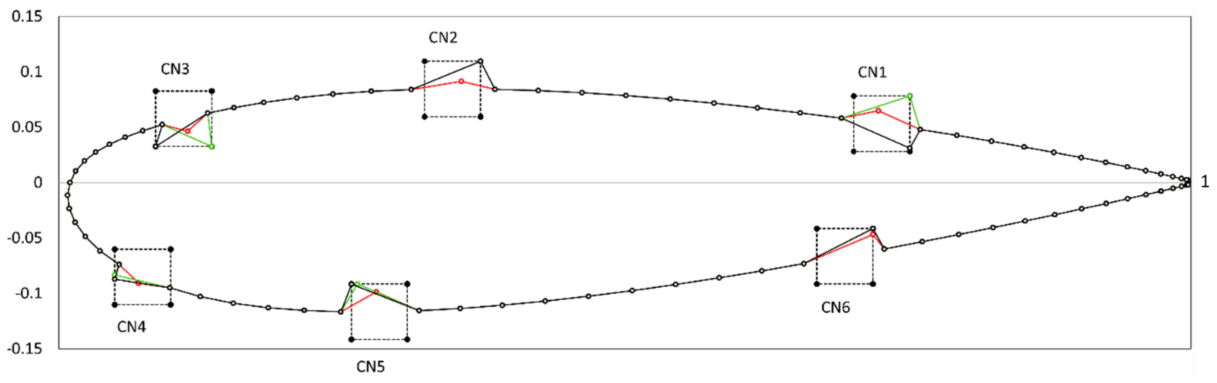


Figure 48 All six control nodes final position within the design space, shown in Figure 45, Figure 46, and Figure 47 on the aerofoil boundary, all other boundary points are remain in their initial positions

Figure 45, Figure 46, and Figure 47 show each control node's design space in detail, highlighting where the algorithms placed the control node as the optimum position. DE and PSO tended to position the control node furthest from the starting position, usually at the edge of the design space. Whereas MCS took a more conservative approach for the majority of the control node movement. Leading the assumption that if the design space was extended, the control nodes may have been positioned further out from their final position seen in the figures.

It is clear from control node plots of cases D-F, the final agent position varies between the algorithms much more than what is seen in the single control node cases in the previous results section. Increasing the degrees of freedom, increases the amount of design space area in which the agents must search. As a result, the ratio of agents to total design space is reduced compared to the single control node cases, meaning there is less chance of an agent being placed close to the 'optimum' position in the design space, this also decreases the chance of the algorithms placing the agents in the same position, hence the variations seen in the geometries. Increasing the number of agents to design space ratio of all three algorithms may decrease the difference seen between the geometry variations. This is also discussed in literature for recommending a number of agents to input into an MCS optimisation depending on the number of degrees of freedom [47][48].

It is important to note that Figure 48 shows movement of the control nodes only, all other nodes on the aerofoil boundary are shown here as unmoved, this is for the purpose of clearly showing the magnitude and direction of the control node's movement for each algorithm only. Changes in the entire aerofoil geometry can be seen in Figure 49, Figure 50 and Figure 51.

Figure 49, Figure 50, and Figure 51 show pressure plots of the final geometry produced by each algorithm at Mach 1.5 (Case D, E, and F), where the red line in the left plots, indicate the initial geometry. Pressure values can be seen scaled on the right side of the pressure plots, indicating the range in which the pressure varies across the flow field.

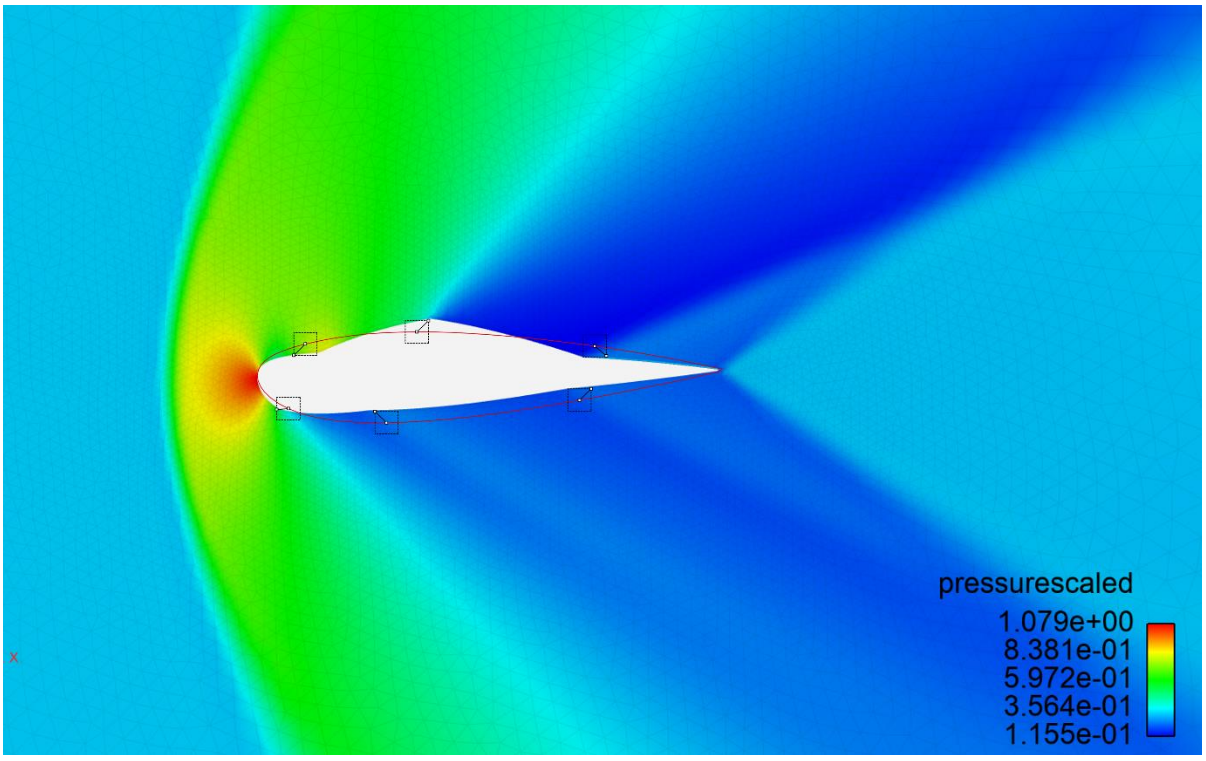


Figure 49 AerOpt pressure distribution plot at the final generation of Case D (Differential Evolution, Mach 1.5). Initial aerofoil geometry is outlined in red with the control nodes in black.

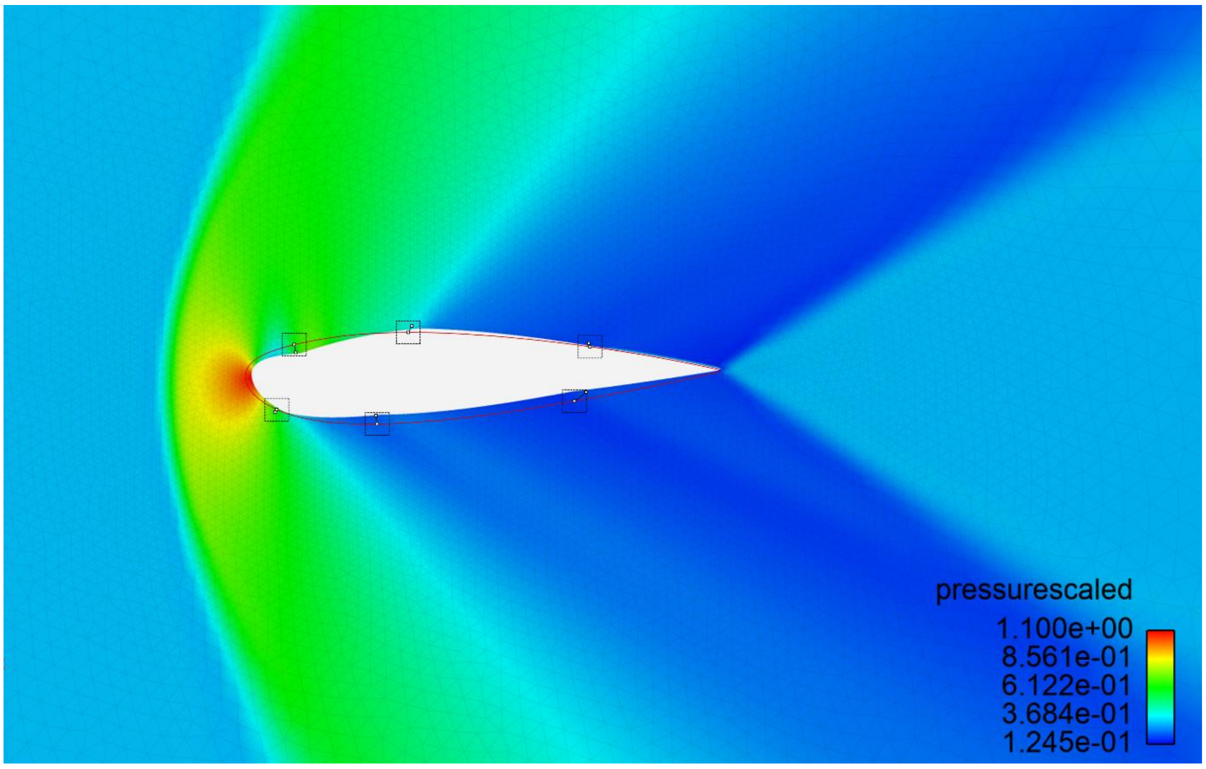


Figure 50 AerOpt pressure distribution plot at the final generation of Case E (Modified Cuckoo Search, Mach 1.5). Initial aerofoil geometry is outlined in red with the control nodes in black.

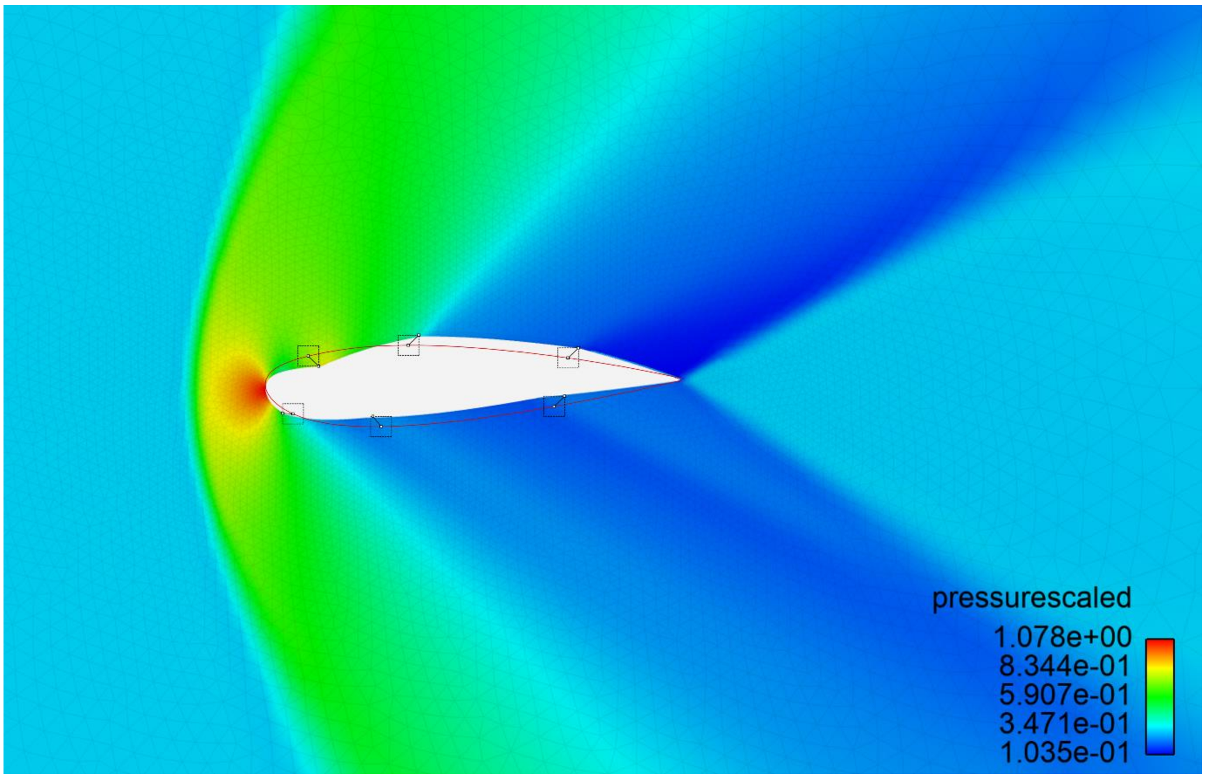


Figure 51 AerOpt pressure distribution plot at the final generation of Case F (Particle Swarm Optimisation, Mach 1.5). Initial aerofoil geometry is outlined in red with the control nodes in black.

The final results show a similar geometry in the front half of the aerofoil; however, DE reduced the thickness of the trailing edge whereas MCS and PSO increased it. DE and PSO display most similar final fitness values of 302% and 308%, suggesting even though they produced differing final geometries, they still achieved a high fitness in their own designs. MCS found a lower fitness increase of approximately 181%, which is notably lower and may have been due to the smaller control node movements, further reasoning for this is discussed.

Table XII Summary of test cases D-F including their final fitness results and fitness percentage increase

Case	*Optimising Algorithm*	*Agents*	*Control Nodes*	*Mach*	*Re*	*Fitness of best agent*		
						Initial	*Final*	*% Increase*
D	*Differential Evolution*	10	6	1.5	3.62E+07	0.207	0.834	302.048
E	*Modified Cuckoo Search*	10	6	1.5	3.62E+07	0.207	0.583	180.926
F	*Particle Swarm Optimisation*	10	6	1.5	3.62E+07	0.207	0.849	309.470

Table XIII A comparison of fitness results in test cases D-F with 6 control nodes, to test cases A.1.3, B.1.3, C.1.3 with 1 control node. Identical input parameters between the cases, but with a different number of control nodes.

Case	*Optimising Algorithm*	*Agents*	*Mach*	*Re*	*Fitness of best agent (6 Control Nodes)*			*Fitness of best agent (1 Control Node)*		
					Initial	*Final*	*% Increase*	*Initial*	*Final*	*% Increase*
D	*Differential Evolution*	10	1.5	3.62E+07	0.207	0.834	302.048	0.207	0.277	33.439
E	*Modified Cuckoo Search*	10	1.5	3.62E+07	0.207	0.583	180.926	0.207	0.277	33.445
F	*Particle Swarm Optimisati*	10	1.5	3.62E+07	0.207	0.849	309.470	0.207	0.277	33.439

Results from Table XIII show by increasing the number of control nodes, the overall aerofoil shape can be optimised to a higher fitness.

In cases D-F, MCS appears to show a significantly lower final fitness than DE and PSO, whereas seen in cases A.1.3-C.1.3, it achieves a higher final fitness. It is mentioned in literature that MCS generally tends to perform favourably with a proportion of N = 10d, where N is number of agents and d is degrees of freedom [47][48]. Here, MCS' results give some indication of this being the case.

All the optimisations seen were run on cluster, with the 6 control node cases (D-F) taking on average approximately 83 minutes to complete one optimisation. The 1 control node cases (A.1.3-C.1.3) took on average approximately 57 minutes to complete one optimisation. Fitness values in cases D-F increased by an average of 264% whereas cases A.1.3-C.1.3 increased by an average of 33%. To achieve closer results to an optimum in terms of fitness, if resources such as time constraints were applicable, these results suggest using a higher number of control nodes may increase the likelihood of achieving a higher fitness in the overall design.

4. Conclusions & Future Work

Each of the three algorithms demonstrated full capability of optimising the cases carried out in this research. In some cases, variation in fitness results between the algorithms were seen, particularly in the higher degree of freedom cases.

It is suggested the NACA21120 was designed with the transonic flow regime in mind, and to test the algorithm's optimising capabilities only, future work may consist of narrowing the test scope, and experimenting on one flow regime only for a particular aerofoil.

Seen in the single control node cases, and particularly with a larger number of agents, the algorithms converged within a much smaller number of generations than the user-inputted, with most converging within 10% of the total number of inputted generations. This left identical CFD simulations, which are considered computationally costly, being solved over and over for the remaining length of the optimisation, using up unnecessary computational resources. Observing the number of CFD simulations essentially 'discarded' once the final fitness was obtained, suggests a stopping criteria may be introduced to prevent this. A stopping criteria may consist of a condition such as,

- 'if the top agent remains at a fitness for > 10 generations, end optimisation'
- 'if there are > 5 agents holding the same top fitness, end optimisation'

Plots (Figure 38, Figure 39, Figure 40 etc.) show the algorithms positioned the control node at the edge of the design space in most cases, suggesting that if the control node's design space was increased, the algorithms may have found an optimum position outside of the original constraints. Further testing would be necessary to validate this theory. This may consist of creating a design space study where the x and y-axis design space constraints are incrementally increased until the algorithms settle for an optimum position comfortably within. This would be effective for the single control node cases as well as the multiple control node cases.

It is already recommended in literature [47][48] that the number of agents (N) to degrees of freedom (d) generally lends well to a ratio of N=10d. It is already notable in the results that increasing the number of agents increases the overall convergence rate of the optimisations, however, not by a significant enough amount to implement the maximum number of agents into any optimisation as a standard parameter, hence the recommended ratio, aiding in improving the efficiency of the optimisation whilst maintaining a realistic availability on computational resources.

A common question upon EA users is the reliability of EAs due to the randomising element involved as well as the lack of control the user has over the direction of the optimisation [30]. Every single control node case was run three times to produce an average of the final fitness, in which some variation was seen. Suggesting that unless EAs input parameters were tuned to the specific case, ensuring the optimisation was as effective in its own parameters as possible, to enhance the robustness and reliability of the final results, it may be suitable to execute the same optimisation multiple times.

It was occasionally seen that MCS found an optimum of significantly lower fitness than DE and PSO. This may have been MCS lacking in its optimisation capabilities for a number of reasons, such as, its number of agents to degrees of freedom ratio was 75% lower than DE and PSO due to the nature in which it optimises, or the Levy Flight step size, which deciphers the degree of randomness introduced into the search, was not optimal for the optimisation problem itself. PSO tended to show the fastest convergence of all the algorithms with DE shortly behind, this could raise questions as to whether PSO was settling on a local optimum rather than the true global optimum [51], this raises a question of robustness in finding a global optimum to assess in future work. Naumann suggests, despite MCS generally showing the slowest convergence, it may have higher potential of ensuring a global optimum is found rather than local, due to its enhanced randomness implementation [51]. Adjusting randomness-embedded parameters such as the Levy Flight step size, holds the potential to provide future work into the influence the randomisation has on the direction of the optimisation.

5. References

[1] Muller J. Radical Reduction of Aircraft Fuel Consumption by Optimising Aerofoil by New Evolutionary Algorithms. 2021 June.

[2] Evans B, Naumann D, Walton S, Edmunds M. Aeropt - Aerodynamic Optimisation Software. Zenodo. 2015 Sept.

[3] Evans B, Hassan O, Jones J, Morgan K, Remaki L. Simulating steady state and transient aerodynamic flows using unstructured meshes and parallel computers. Computational Fluid Dynamics Review 2010. World Scientific. 2010; 1-27.

[4] Skinner SN, Zare-Behtash H. State-of-the-art in aerodynamic shape optimisation methods. Applied Soft Computing. 2018; 62: 933-962.

[5] Fisher RA. Design of experiments. British Medical Journal. 1936 Mar; 1 (3923): 554.

[6] Ypma TJ. Historical Development of the Newton-Raphson Method. Society for Industrial and Applied Mathematics. 1995 Dec; 37 (4): 531-551.

[7] Bazaraa MS, Sherali HD, Shetty CM. Nonlinear programming: theory and algorithms. John Wiley & sons; 2013 Jun.

[8] More JJ. The Levenberg-Marquardt algorithm: implementation and theory. Numerical analysis. 1978; 105-116.

[9] Hestenes MR, Stiefel E. Methods of conjugate gradients for solving. Journal of research of the National Bureau of Standards. 1952; 49 (6): 409-438.

[10] Hare W, Nutini J, Tesfamariam. A survey of non-gradient optimisation methods in structural engineering. Advances in Engineering Software. 2013 May; 59: 19-28.

[11] Heijnen J. Can Neuro-Evolution Enhance Evolutionary Aerodynamic Design? Swansea University, Department of Computer Science. 2021 Oct.

[12] Yu Y, Lyu Z, Martins J R.R.A. On the influence of optimisation algorithm and initial design on wing aerodynamic shape optimisation. Aerospace Science and technology. 2018 Apr; 75: 183-199.

[13] Zingg DW, Nemec M, Pulliam TH. A comparative evaluation of genetic and gradient-based algorithms applied to aerodynamic optimisation. European Journal of Computational Mechanics. 2008 Jan; 17 (1-2): 103-127.

[14] Whitley D, Rana S, Dzubera J, Mathias KE. Evaluating evolutionary algorithms. Artificial Intelligence. 1995; 85: 245-276.

[15] Fortin FA, Rainville FMD, Gardener MA, Parizeau M, Gagne C. Evolutionary Algorithms made easy. Machine Learning Research. 2012; 13: 2171-2175.

[16] Poli R, Langdon WB. A Review of Theoretical and Experimental Results on Schemata in Genetic Programming. In: Banzhaf W, Poli R, Schoenauer M, Fogarty TC. Genetic Programming. Springer – Verlag Berlin Heidelberg; 1998. p. 1-16.

[17] Beyer HG, Schwefel HP. Evolution strategies. Natural Computing. 2002; 1: 3-52.

[18] Kennedy J, Eberhart R. Particle Swarm Optimisation. Proceedings of ICNN'95 - International Conference on Neural Networks. 1995.

[19] Sahab MG, Toropov VV, Gandomi AH. A Review on Traditional and Modern Structural Optimization: Problems and Techniques. Metaheuristic Applications in Structures and Infrastructures. 2013: 25-47.

[20] Storn R, Price K. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimisation. 1997; 11: 341-359.

[21] Walton S, Hassan O, Morgan K, Brown MR. Modified Cuckoo Search: A new gradient free optimisation algorithm. Chaos, Solitons & Fractals. 2011; 44: 710-718.

[22] Naumann DS, Evans B, Walton S, Hassan O. A novel implementation of computational aerodynamic shape optimisation using Modified Cuckoo Search. Applied Mathematical Modelling. 2015; 40: 4543-4559.

[23] Yang XS, Deb S. Cuckoo Search via Levy Flights. Proceedings of World Congress on Nature & Biologically Inspired Computing. 2009; 1: 210-214.

[24] Yang XS, Deb S. Engineering Optimisation by Cuckoo Search. Int. J. Mathematical Modelling and Numerical Optimisation. 2010; 1: 330-343.

[25] Solanki C, Thapliyal P, Tomar K. Role of Bisection method. International Journal of Computer Applications Technology and Research. 2014; 3 (8): 533-535.

[26] Mier PR. A tutorial on Differential Evolution with Python. [Internet]. 2017 Sept [cited 2022 Jan 28]. Available from: https://pablormier.github.io/2017/09/05/a-tutorial-on-differentialevolution-with-python/

[27] Evans B, Morton T, Sheridan L, Hassan O, Morgan K, Jones J, Chapman M, Ayers R and Niven I, "Design optimisation using computational fluid dynamics applied to a land–based supersonic vehicle, the BLOODHOUND SSC," Structural and Multidisciplinary Optimization. 2013 Feb; 47(2): 301-316.

[28] Jameson A and Vassberg J. Computational Fluid Dynamics for Aerodynamic Design: Its Current and Future Impact. AIAA. 2001 Jan; 538.

[29] Ditchburn H. Evolutionary Optimisation of Aerofoils in Subsonic, Transonic, and Supersonic Flows. 2021 April.

[30] Vincalek J, Walton S, Evans B. A user-centered approach to evolutionary algorithms and their use in industry. Cogent Engineering. 2021 Nov; 8:1.

[31] Pinel F, Danoy G, Bouvry P. Evolutionary Algorithm Parameter Tuning with Sensitivity Analysis. International Joint Conferences on Security and Intelligent Information Systems. 2011 Jun; 204-216.

[32] Mirafzal SH, Khorasani AM, Ghasemi AH. Optimizing time delay feedback for active vibration control of a cantilever beam using a genetic algorithm. Journal of Vibration and Control. 2015; 22 (19): 4047-4061.

[33] Evans B, Walton S. Aerodynamic optimisation of a hypersonic reentry vehicle based on solution of the Boltzmann–BGK equation and evolutionary optimisation. Applied Mathematical Modelling. 2017; 52: 215-240.

[34] Johnson JM, Rahmat-Samii V. Genetic algorithms in engineering electromagnetics. IEEE Antennas and Propagation Magazine. 1997; 39 (4): 7-21.

[35] Hempsell M. Progress on the Skylon and Sabre. Proceedings of the International Astronautical Congress. 2013; 11: 8427-8440.

[36] Liu X, He W. Airfoil optimization design based on the pivot element weighting iterative method. Algorithms. 2018 Oct; 11 (10): 163-184.

[37] Packham S, Parmee IC. Data analysis and visualisation of cluster-oriented genetic algorithm output. 2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics. 2000 Jul; 173-178.

[38] Interrante V, Laidlaw D, Taylor R, Ware C. Design and Evaluation in Visualisation Research. IEEE Visualisation Research. 2005 Oct.

[39] Pohlheim H. Visualization of evolutionary algorithms – set of standard techniques and multidimensional visualization. Proceedings of the Genetic and Evolutionary Computation Conference. 1999 Jul; 1: 533-540.

[40] Collins TD. Applying software visualization technology to support the use of evolutionary algorithms. Journal of Visual Languages & Computing. 2003 Apr;14 (2): 123-150.

[41] Collins TD. HENSON: a visualization framework for genetic algorithm users. Proceedings of the 1999 Congress on Evolutionary Computation - CEC99. 1999 Jul; 1: 562-569.

[42] Lawrence AW, Badre AM, Stasko JT. Empirically evaluating the use of animations to teach algorithms. Proceedings of 1994 IEEE Symposium on Visual Languages. 1994 Oct: 48-54.

[43] Sayma A. Computational Fluid Dynamics. Bookboon; 2009 Jan.

[44] Dettmer W G. Fundamentals of aircraft performance, stability, and control. Flight Mechanics.

[45] RSEs at SA2C. Swansea Academy of Advanced Computing. [Internet]. [cited 2022 Sept]. Available from: https://www.swansea.ac.uk/compsci/swansea-academy-of-advanced-computing-sa2c/

[46] Supercomputing Wales. About sunbird. [Internet]. [cited 2022 Sept]. Available from: https://portal.supercomputing.wales/index.php/about-sunbird/

[47] Walton S, Hassan O, Morgan K. Selected Engineering Applications of Gradient Free Optimisation Using Cuckoo Search and Proper Orthogonal Decomposition. Archives of Computational Methods in Engineering. 2013; 20: 123-154.

[48] Walton S, Hassan O, Morgan K. Reduced order mesh optimisation using proper orthogonal decomposition and a modified cuckoo search. International Journal for Numerical Methods in Engineering. 2012 Feb; 93 (5): 527-550.

[49] Naumann DS, Evans B, Walton S, Hassan O. Discrete boundary smoothing using control node parameterisation for aerodynamic shape optimisation. Applied Mathematical Modelling. 2017; 48: 113-133.

[50] Sørensen KA. A multigrid accelerated procedure for the solution of compressible fluid flows on unstructured hybrid meshes. [Ph.D. thesis]. 2001; Swansea University.

[51] Naumann DS. An automated, mesh-based Framework for Computational Aerodynamic Shape Optimisation. [Ph.D. thesis]. 2017 Jul; Swansea University.