The University Of Sheffield.

Department Of Mechanical Engineering

MEng Mechanical Engineering

Finite Element Optimisation of Bolted Joint Locations using Differential Evolution

Blake **HEMINGWAY**

May 2014

Supervisor: Dr Graeme Manson

Thesis submitted to the University of Sheffield in partial fulfilment of the requirements for the degree of Master of Engineering

SUMMARY

Structural design optimisation using Finite Element Analysis (FEA) is a laborious and time-consuming process, but is ubiquitous in the engineering industry. Therefore, this study examined the feasibility of using a class of evolutionary algorithm known as Differential Evolution (DE) to automate this process in the general case, with specific emphasis on the optimisation of bolted joint locations.

Firstly, the performance of the most widely-cited DE algorithms in literature was reviewed. Based on independent testing, the JADE algorithm was selected and modified for use as an FEA optimiser. The modified algorithm was then used to successfully solve a real-world bolted joint optimisation problem.

The study concluded that the use of 'evolutionary FEA' is viable for small linear FEA models of critical components, and is useful for yielding optimum and perhaps non-obvious design solutions. However, real-world optimisation problems requiring FEA are typically large-scale and nonlinear. The large amount of computing time required for such problems may limit the industrial usefulness of this approach at present. Nevertheless, given the current rate of computer hardware development, it is hypothesised that that this technique may become commercially viable within the next 10-20 years.

NOMENCLATURE

ρ	Density (kg/m^3)
ν	Poisson's ratio
a	Plate primary (longest) dimension (m)
b	Plate secondary dimension (m)
cr	DE crossover ratio $\in [0, 1]$
d	Diameter (m)
D	DE problem dimensionality
E	Young's modulus (Pa)
f	Fastener flexibility (m/N)
F	DE mutation scaling factor
g	Acceleration due to gravity on earth = 9.81 (m/s^2)
G	DE generation number
h	Fluid depth (m)
l	Effective length (m)
Np	DE population size
p	Hydrostatic pressure (Pa)
r	Random number
t	Thickness (m)
$\boldsymbol{x}$	DE population vector
y	Out-of-plane displacement (m)

Bold type distinguishes vector quantities throughout.

Numerical calculations have been computed to three significant figures.

CONTENTS

ACKNOWLEDGEMENTS

Firstly, to Jeremy Taylor and Professor Richard Dawkins, for producing the 1987 BBC documentary ‘The Blind Watchmaker’. This programme inspired me to apply Darwinian methods in an engineering context.

Secondly, to Atkins Aerospace and the Bloodhound SSC land-speed record project, for providing me with the perfect benchmark problem on which to test my theory.

Finally, to my supervisor, Dr Graeme Manson, for guiding me through the existing research into evolutionary algorithms and facilitating the entire project.

1 INTRODUCTION

1.1 Motivation

Successful engineering design has always been an iterative process. Very few initial design concepts are without flaws; therefore, there is a requirement to test prototypes, observe their shortcomings, and feed any lessons learnt back into the design.

In recent years, the iterative engineering design process has been greatly accelerated through the use of computational simulation techniques such as Finite Element Analysis (FEA). FEA is capable of predicting the behaviour of an engineering structure under loading, substantially reducing the requirement to manufacture and test physical prototypes.

Despite this, the modern industrial design process remains highly time-consuming. In a world where time-to-market is crucial, it is in the interest of engineering companies to accelerate the process further.

Therefore, in broad terms, this piece of research sought to develop a method to quicken the design process by removing the requirement for manual iteration of Finite Element (FE) models. This would represent a key milestone in the move from Computer Aided Design (CAD) to Computer Automated Design (CAutoD).

1.2 Objective Case

The problem of iterative FEA was thrown into sharp relief during the course of a project undertaken by Atkins Aerospace on behalf of the Bloodhound SSC team in 2013. Bloodhound SSC is a land-speed record car under construction at the time of writing. It features a 430 litre composite tank to house the jet fuel for the vehicle's EJ200 Eurofighter jet engine. This tank incorporates a flat aluminium baffle to prevent excessive sloshing of fuel under maximum acceleration and braking.

The specific problem encountered was how to bolt the baffle to the side of the tank with 16 bolts in such a way that its deflection is minimised under a peak braking force of $3g$. Considering the 16 bolts with respect to a two-dimensional Cartesian coordinate system, this gave rise to a 32-dimensional optimisation problem. The optimum bolting locations were not obvious because of the tank's complex geometry. Moreover, the bolts were highly interdependent, in the sense that moving one could affect the force and displacement distribution around all of the others.

For these reasons, the design process was time-consuming. Around 40 man-hours were spent iterating FE models. If some hypothetical optimisation software were available, capable of being set up and run within one working day, a financial and time saving of 80% would have been realised.

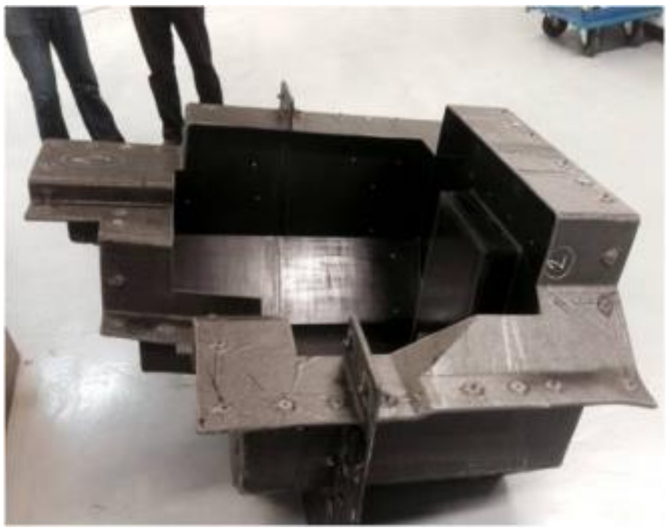


Figure 1.1: Bloodhound SSC jet fuel tank. Note the unusually complex geometry. Reproduced from (1).

1.3 Project Aim

Success in this project would be defined by the realisation of an FEA subroutine capable of optimising the bolting locations around the Bloodhound SSC fuel tank baffle without the requirement for manual iteration. Such a subroutine should be accessible and computationally efficient in order to make it industrially viable for use on other bolted joint optimisation problems.

1.4 Limits of Existing Finite Element Optimisers

Several commercial FEA programs now include tools for optimisation of topology; a type of CAutoD. Two prominent examples are Altair OptiStruct and Abaqus ATOM. Although the exact details of these codes are proprietary, both work by gradually removing material from a pre-determined design space, as illustrated in Figure 1.2.

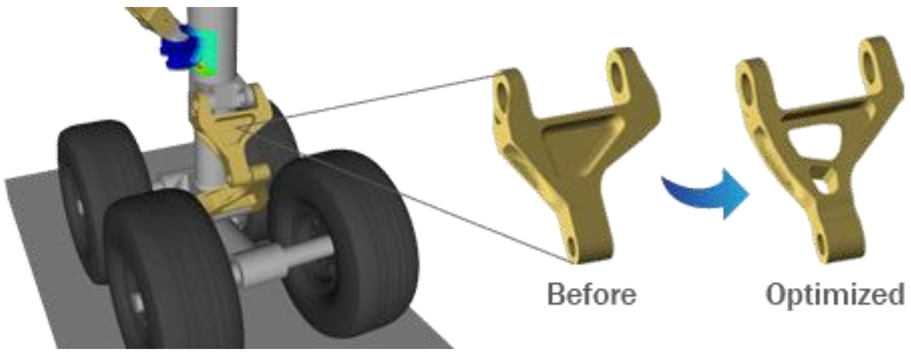


Figure 1.2: CAutoD optimisation of a landing gear bracket using Altair OptiStruct. Reproduced from (2).

It is apparent that a bolted joint location optimiser would require a fundamentally different approach to the material removal strategies of OptiStruct and ATOM. No commercial code is capable of optimising bolted joint locations at the time of writing.

1.5 Engineering Design and Darwinian Evolution

In Biology, Darwinian evolution, also known as Natural Selection, harnesses random variations in natural populations known as mutations. Useful mutations survive and propagate into future generations to derive plants and animals that are optimised for their environment.

The potential for utilising this concept in an engineering context was first explored by Ingo Rechenberg in 1964 (3), before being elucidated fully in 1973 (4). Rechenberg demonstrated that a hinged aerofoil, as depicted in Figure 1.3, could be optimised for minimum drag in 300 ‘generations’, despite his test rig having $51^5 = 345{,}025{,}251$ potential configurations. His Evolutionary Algorithm (EA) was based on random fluctuations of the hinges (calculated by throwing dice) known as ‘mutations’. Solutions that reduced drag in comparison with their ‘parents’ were retained, whilst poorly-performing solutions were discarded.

The most salient and thought-provoking conclusion of Rechenberg’s research was that the algorithm arrived quickly at the optimal solution – a flat plate – despite being fed no *a priori* knowledge of the problem. In addition, unlike many optimisation algorithms, Rechenberg’s EA did not require the objective function landscape to be continuous or differentiable. As FE models are discontinuous by definition, the possibility of using an EA to optimise bolted joint locations was investigated.

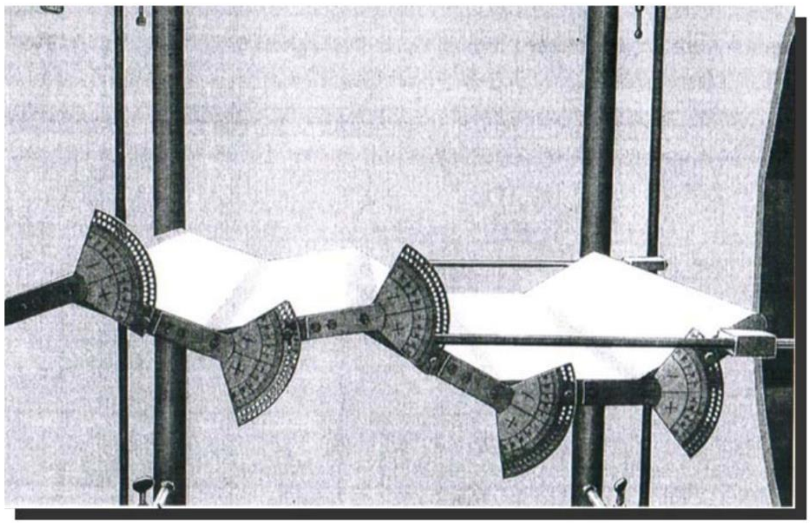


Figure 1.3: Rechenberg's original 'hinged aerofoil' experiment. Reproduced from (4).

1.6 Differential Evolution (DE)

EAs have flourished as a research topic since the work of Rechenberg, although most researchers now use computers instead of dice. Dedicated academic journals exist on the subject, such as IEEE Transactions on Evolutionary Computation, and competitions, such as the International Contest on Evolutionary Optimization (ICEO), are regularly held to determine the best algorithms.

A consistently well-performing algorithm at such competitions is Differential Evolution (DE), published by Storn and Price in 1995 (5). Not only does DE perform well against other EAs, but it has also been shown by its creators (6 pp. 156-182) to perform comparably to or better than other global optimisation algorithms such as Nelder and Mead (7), Simulated Annealing (8) and Particle Swarm Optimization (PSO) (9) with respect to convergence reliability and speed.

Like most EAs, the only two requirements of DE are that the problem at hand can be parameterised into vector form, and that any given vector can be mapped to some cost function, $f(\boldsymbol{x})$. DE will then seek to find the vector $\boldsymbol{x}_{min}$ such that $\forall \boldsymbol{x}, f(\boldsymbol{x}_{min}) \leq f(\boldsymbol{x})$ (10). The objective case for this study satisfies these requirements as n bolted joint locations may be encoded as a flattened vector of Cartesian coordinates: $(x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n)$. $f(\boldsymbol{x})$ would be the maximum total deflection corresponding to $\boldsymbol{x}$ as output by FEA.

For the reasons given above, DE was deemed to be an appropriate class of algorithm upon which to build a bolted joint optimiser. The opportunity to utilise the DE expertise of the Dynamics group at the University of Sheffield was also taken into consideration. The workings of the DE algorithm are discussed in detail in Section 2.

1.7 EAs in FEA – A Literature Review

Although the idea of using EAs to solve engineering optimisation problems is now well-established, their application to FEA is novel, with only a handful of precedents in research papers, all of which were published after the last millennium.

Early efforts focused on small-scale topology optimisation. Li *et al.* (11) used a bespoke EA to optimise the cross-sectional properties of various torsional shafts. Meanwhile, Hull *et al.* (12) used DE to optimise truss frames.

It is probable that the work of these two research groups has now been superseded by the proprietary codes discussed in Section 1.4. However, their papers both pick up on two key themes. Firstly, an important advantage of combining EAs with FEA is the fact that any proficient FEA analyst may take advantage of the technique without learning new mathematics or advanced programming (11). Secondly, and less encouragingly, slow convergence and protracted runtimes are the main barriers to the widespread uptake of the method (12).

Some researchers, such as Chen and Cheng (13), have been sufficiently pessimistic about the "huge" number of function evaluations required by EAs to dismiss them as unsuitable for structural analysis. Instead, Chen and Cheng, along with Su *et al.* (14), focused their research on using surrogate modelling and data mining alongside EAs to reduce computational expense.

However, it is arguable that the above researchers were too quick to make recourse to such approximate methodologies, as rapid improvements in computer hardware and its accessibility have taken place in recent years. Aside from an exponential increase in Central Processing Unit (CPU) speed in accordance with Moore's Law (15), parallel computing has also become much more widely available.

Against this backdrop, Luo *et al.* (16) used DE in conjunction with both FEA and Computational Fluid Dynamics (CFD) to perform a highly successful multi-objective optimisation of an aerospace compressor blade in 2009. The authors reported a 1.58% improvement in isentropic efficiency and an 11.6% reduction in maximum stress in comparison with the standard 'NASA Rotor 37' blade. The total runtime was 16 days over 10 parallel CPUs.

With regards to software, it was encouraging to read that many of the existing papers on the topic of EAs in FEA have used some variant of DE (12), (16), (17). Further justification for the use of DE comes from Vu (17), who conducted a comparative study into how both DE and PSO could be used to optimise toroidal pressure vessels using Abaqus FEA software and FORTRAN subroutines. Vu found that DE outperformed PSO in most practical cases.

1.8 Project Objectives

In light of the literature described above, it was apparent that this project was to have two main objectives in order to achieve the aim stated in Section 1.3. These were as follows:

1. To mitigate the problem of long runtime, a variant of DE was to be developed. This variant would exhibit more rapid convergence than the basic algorithm, whilst keeping the number of control parameters as small as possible to maintain ease-of-use. The development of such an algorithm is covered by Section 2, with the following sub-objectives:
    a. To review the existing DE variants in literature
    b. To independently code and benchmark these variants
    c. To improve further upon the best variant
2. The algorithm was to be modified for use in FEA. This is covered by Section 3, with the following sub-objectives:
    a. To develop the subroutines necessary to pre- and post-process an FEA model
    b. To exploit parallel computing to further mitigate runtime
    c. To test the algorithm on a trivial problem for which the solution is known *a priori*
    d. To use the algorithm to optimise the bolted joint locations around the Bloodhound SSC jet fuel tank baffle

1.9 Software Used

All algorithms were developed in the Python programming language for reasons of open access, code readability and experience. The Abaqus 6.11 FEA package was used as the FE solver as it features a built-in Python Application Programming Interface (API).

2 DEVELOPMENT OF A DE VARIANT

2.1 Baseline Algorithm

DE algorithms operate on a population of Np vectors. The classic DE variant, known as DE/rand/1/bin (6 pp. 37-41), is described below.

At the start of the algorithm, the population is randomly initialised within a predefined range. To 'evolve' the optimum vector, DE generates 'mutant' vectors by perturbing the existing vectors in the population. The perturbation vectors are calculated as the weighted difference between a further two randomly selected vectors, hence the name 'Differential Evolution'. The DE/rand/1/bin mutation operation is described by Equation 2.1:

$$\boldsymbol{x}_{mutant} = \boldsymbol{x}_{r0} + F \cdot (\boldsymbol{x}_{r1} - \boldsymbol{x}_{r2})$$

Equation 2.1

The mutant vectors are then 'crossed over' with their parents to create trial vectors as per Equation 2.2. This means that each trial vector component comes from either its parent or a mutant in accordance with some probability distribution. A random vector index j_{rand} is guaranteed to come from the mutant to prevent a trial vector being identical to its parent:

$$\boldsymbol{x}_{trial,i} = x_{trial,j,i} = \begin{cases} x_{mutant,j} & (r \in (0,1) \leq cr \mid j = j_{rand}) \\ x_{parent,j} & otherwise \end{cases}$$

Equation 2.2

The trial vectors are then tournament-selected against the existing population based on a cost function, as shown in Equation 2.3. The winning vectors go on to form the next generation, and the process is repeated until some convergence criterion is reached.

$$\boldsymbol{x} = \begin{cases} \boldsymbol{x}_{trial} & (cost(\boldsymbol{x}_{trial}) \leq cost(\boldsymbol{x}_{parent})) \\ \boldsymbol{x}_{parent} & otherwise \end{cases}$$

Equation 2.3

A flowchart, C-style pseudo-code and description of the standard DE/X/Y/Z taxonomy are attached as Appendix 1 to further illuminate the baseline algorithm.

2.2 DE Control Parameters

Regrettably, the convergence speed and reliability of DE has been shown to be highly sensitive to the algorithm's control parameters (18), (19), (20). These are:

- The crossover ratio, cr. A small cr is known to be useful on separable function landscapes as it encourages exploration of each vector component independently (20). Only high cr values guarantee the contour matching properties of DE required to solve parameter-dependent functions (6 pp. 97-104).
- The population size, Np. Various conflicting heuristics exist for the choice of Np, including $10D$ (21), $min(10D, 40)$ (22), between $3D$ and $8D$ (23), and 30, 100, and 400 in the case of $D \leq 10$, $= 30$, and $= 100$, respectively (19). Large values of Np are normally associated with slow but reliable convergence. The converse is true for small Np.
- The mutation scaling factor, F. If static, F should lie in the (open) interval $(z, 2)$, where z is the theoretically derived critical value for F published by Zaharie (24), reproduced in Equation 2.4:

$$z = \sqrt{\frac{1 - \frac{cr}{2}}{Np}}$$

Equation 2.4

  A large F has been associated with reliable but slow convergence as it tends to promote diversity in the population (6 pp. 75-79). As with Np, the converse is true for a small F.
- The DE 'strategy', i.e. the method of mutation and crossover. DE/rand/1/bin is known to be stable. Other versions exist which increase the rate of convergence, but raise the probability of converging upon a local rather than global minimum.

The control parameters exhibit a high degree of interdependence and problem dependence (10). Achieving the optimum control parameter settings normally requires a tedious trial-and-error process, which is clearly unacceptable in FEA on grounds of computational expense. Moreover, it can be shown that different control parameters may be required at various phases of the same optimisation problem. Therefore, an algorithm capable of automatically selecting the optimum parameters was sought. In excess of 100 man-hours were dedicated to this task, given the criticality of identifying an efficient algorithm in accordance with the project aim.

2.3 Automatic Parameter Control in DE

Of the algorithms in literature which apply adaptive logic to automate DE's control parameters, three are notably competitive and distinct. These are jDE (10), SaDE (18) and JADE (19).

2.3.1 jDE

jDE was published by Brest *et al.* in 2006 (10). Based upon classic DE, jDE encodes cr and F directly onto the individual vectors in the population. After each generation, the cr and F values are read from the parent individuals, but encounter a 10% probability of being regenerated according to uniform distributions on $[0, 1]$ and $[0.1, 1]$, respectively. Successful parameters survive to be reused in future generations.

2.3.2 SaDE

Qin *et al.*'s SaDE algorithm[1] (18) uses four DE strategies: DE/rand/1/bin; DE/rand-to-best/2/bin; DE/rand/2/bin; and DE/current-to-rand/1. Each trial vector is assembled using one of these strategies, with an initial probability of 0.25 assigned to each. The successes and failures of each strategy are logged. After a predefined number of generations known as the Learning Period (LP), the probabilities are reset in proportion with the number of successes achieved by each strategy. 50

[1] This paper discusses the 2009 'improved' SaDE, as opposed to the original 2005 algorithm (28).

generations is suggested by the authors as an appropriate LP. Preliminary work in this project showed the optimum LP to be moderately problem dependent.

SaDE defines cr as a normally distributed variable, sampled once per crossover operation, initially centred on 0.5, with a standard deviation equal to 0.1, and truncated to $[0, 1]$. The sampled cr values found to be successful are saved. After $(G$ – LP) generations (G > LP), the normal distribution centre moves to the median of the successful cr values. Each DE strategy has its own cr memory, allowing each strategy to adapt its own optimum cr.

F is declared as a normally distributed variable such that $F = N(0.5, 0.3)$. SaDE does not seek to adapt F.

Finally, SaDE implements a local search method at $G = 200$. This was not considered here as local search algorithms are considered independently in Section 2.7.

2.3.3 JADE

Zhang and Sanderson's JADE algorithm (19) uses a single, novel DE strategy known as DE/current-to-*p*best/1/bin. This is a generalisation of the more common DE/current-to-best/1/bin; which purportedly balances the convergence speed of the latter with the robustness of classic DE. Moreover, JADE includes information from recently explored inferior vectors (an 'archive') in the construction of vector differences. This process allows the algorithm to exploit promising progress directions more efficiently. JADE's mutation strategy is given by Equation 2.5, in which '*p*best' denotes one of the top-performing *p*% vectors in the population, and $\widetilde{\boldsymbol{x}}_{r2}$ is randomly selected from the union of the population and the archive.

$$\boldsymbol{x}_{mutant} = \boldsymbol{x}_i + F \cdot \left(\boldsymbol{x}_{pbest} - \boldsymbol{x}_i\right) + F \cdot (\boldsymbol{x}_{r1} - \widetilde{\boldsymbol{x}}_{r2})$$

Equation 2.5

JADE's strategy for cr adaptation is similar to that of SaDE, although JADE uses a shorter LP (one generation) and compensates for any erratic behaviour this may cause by using an under-relaxation factor. Moreover, JADE moves cr towards the arithmetic mean of the successful values, as opposed to the median, after the LP elapses.

F is adaptively controlled by JADE. Initially, F obeys a Cauchy distribution centred on 0.5, with scaling factor = 0.1. After each generation, the distribution centre is moved towards the Lehmer mean of the F values generating successful mutations (given by Equation 2.6, thereby favouring larger, diversifying values of F). An under-relaxation factor is employed. Experiments conducted as part of this study showed that F adaptation based on the arithmetic mean is 'greedy': F tends to move to low values to encourage rapid but often premature convergence. In this light, Zhang and Sanderson's use of the Lehmer mean may be thought of as a 'back pressure' to this greedy tendency.

$$L(a_1, \dots, a_n) = \frac{\sum_{k=1}^{n} a_k^2}{\sum_{k=1}^{n} a_k}$$

Equation 2.6

2.4 Benchmarking

Each paper discussed above (10), (18), (19) employs different benchmarking techniques and statistical measures to indicate that the subject algorithm outperforms all others. Most pertinently, a direct and impartial comparison between JADE and the 2009 'improved' SaDE algorithm does not exist in literature as the two papers were published in the same year. Therefore, an independent benchmarking exercise was undertaken. Firstly, an object-oriented version of classic DE was coded in Python, independently and from scratch. SaDE, JADE and jDE were then implemented as subclasses based on the algorithmic descriptions presented in the relevant papers, such that the differences may be readily appreciated. The full source code, enclosed on disc, may be of interest to readers of this paper.

2.4.1 Benchmark Functions

Five 'classic' optimisation problems were used as the cost functions, with four of the five coming from the widely-used De Jong five-function test suite (25). The functions are briefly summarised below, with full mathematical definitions (taken from (6 pp. 513-533)) and two-dimensional visualisations given as Appendix 2.

- De Jong's first function, (the sphere, henceforth known as f_1), is a smooth, separable, unimodal function. It is known to be simple to optimise, and is included as a measure of speed.
- De Jong's second function, (Rosenbrock's banana, henceforth known as f_2), is unimodal and non-separable. It contains a curved valley with a shallow gradient to the optimum, which is notoriously difficult for optimisation algorithms to find.
- De Jong's third function, (the step function, henceforth known as f_3), is a quantised unimodal function. This is an important benchmark for an FEA optimiser, which will be required to work in quantised (node-based) landscapes.
- De Jong's fourth function, the noisy quartic, is a measure of an optimisation algorithm's performance on noisy data; an attribute that was deemed irrelevant to this study. Instead, it was replaced by Ackley's function, henceforth referred to as f_4. According to Price *et al.* (6 pp. 518-519), this is one of the most commonly-cited multimodal test functions in optimisation literature.
- De Jong's fifth function, (known as Shekel's Foxholes, henceforth known as f_5), is a multimodal function. A successful algorithm must avoid prematurely converging upon local optima.

Each function was optimised in 10 and 30 dimensions, with the exception of f_5, which is restricted by definition to five dimensions. Each function was to be optimised 30 times at each dimension by each of the three adaptive algorithms, and by classic DE with the standard parameter settings cited by Storn (22). Given that none of these algorithms adaptively select Np, this parameter was set to 50 for all studies, in line with the approach used by Qin *et al.* (18).

2.4.2 Justification for the Use of Benchmark Functions

The use of these test functions is standard practice, and is the only sensible way of benchmarking an optimiser's performance. It is not the intention of this thesis to imply that bolted joint optimising function landscapes output from FEA will resemble any of the benchmark functions directly. Nevertheless, the principle of optimising a higher-dimensional objective function, which may be multimodal, parameter dependent and challenging to solve, is common to both the test and objective (Bloodhound) cases. On these grounds, the selection of the 'best' DE variant based upon the five test functions presented in Section 2.4.1 is justified.

2.4.3 Benchmarking Results

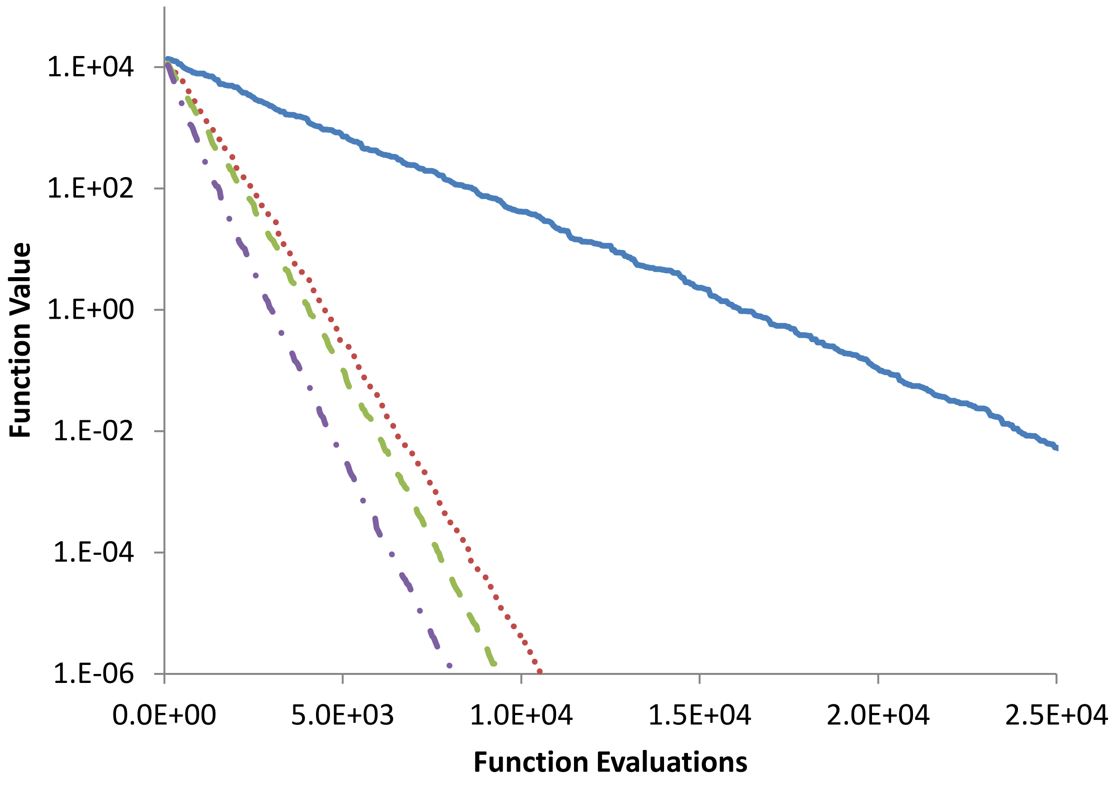
The algorithms were assessed on two key performance metrics: success rate of convergence upon the (known) global minimum to within a function value of 10^{-6} after 50,000 ($D \leq 10$) or 100,000 ($D = 30$) function evaluations (Table 2.1), and median best-so-far function value (Figure 2.1- Figure 2.5).

	f_1		f_2		f_3		f_4		f_5
D	*10*	*30*	*10*	*30*	*10*	*30*	*10*	*30*	*5*
DE	**100**	0	0	0	**100**	**100**	0	0	10
jDE	**100**	**100**	20	0	**100**	**100**	**100**	**100**	13
SaDE	**100**	**100**	10	0	**100**	**100**	**100**	93	**20**
JADE	**100**	**100**	**93**	**73**	**100**	**100**	**100**	**100**	17

Table 2.1: Convergence success rate % after 50,000 (D≤10) or 100,000 (D=30) function evaluations (best results are marked in bold)

In the figures that follow, the following legend is adopted:

———	DE/rand/1/bin
······	jDE
– –	SaDE
– · ·	JADE



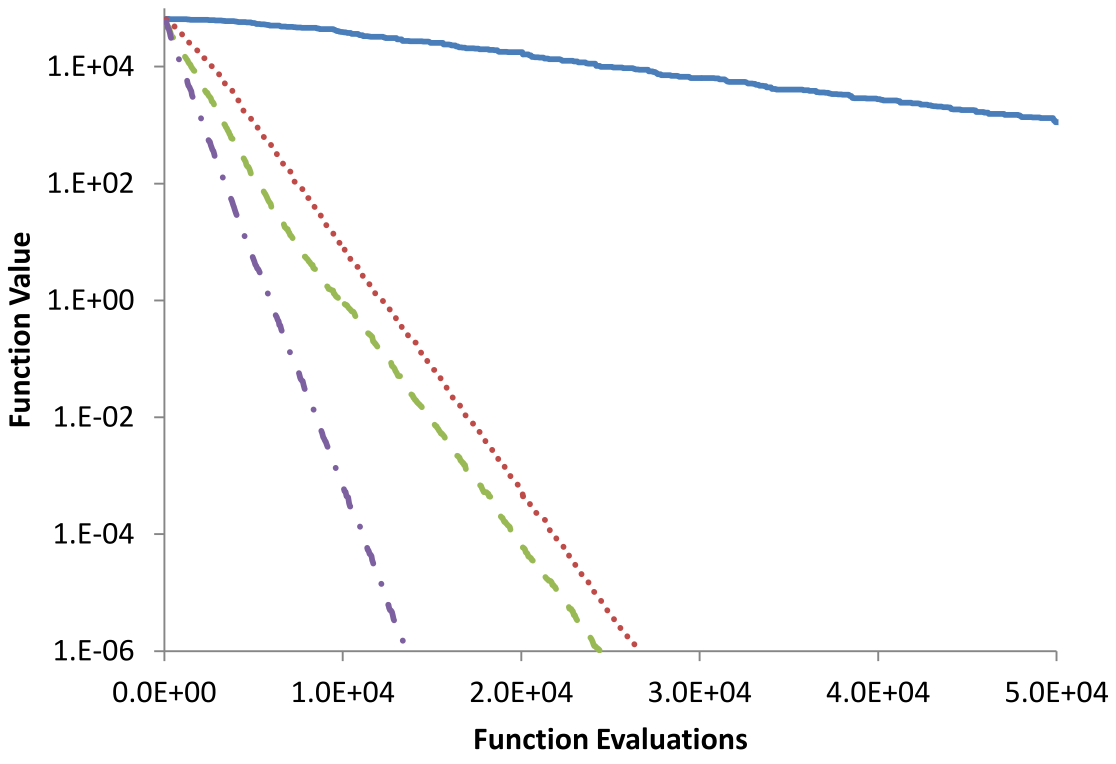
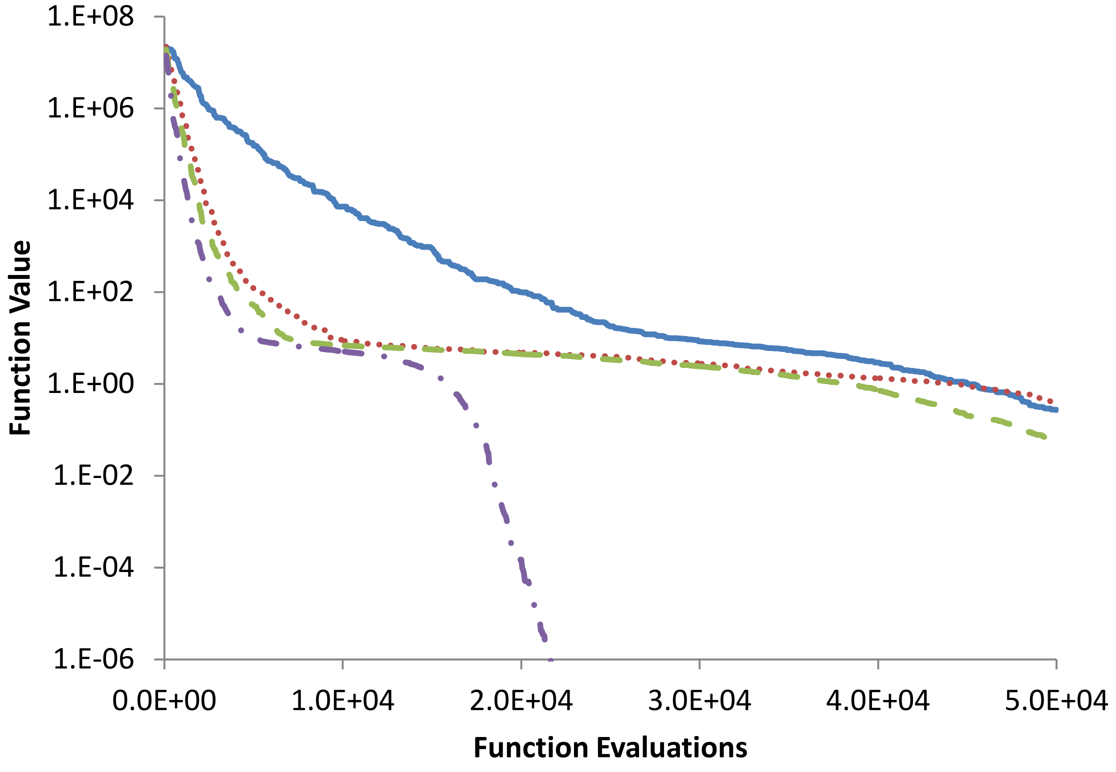


Figure 2.1: Median best-so-far function values for f_1 (Sphere) in 10D (top) and 30D (bottom)



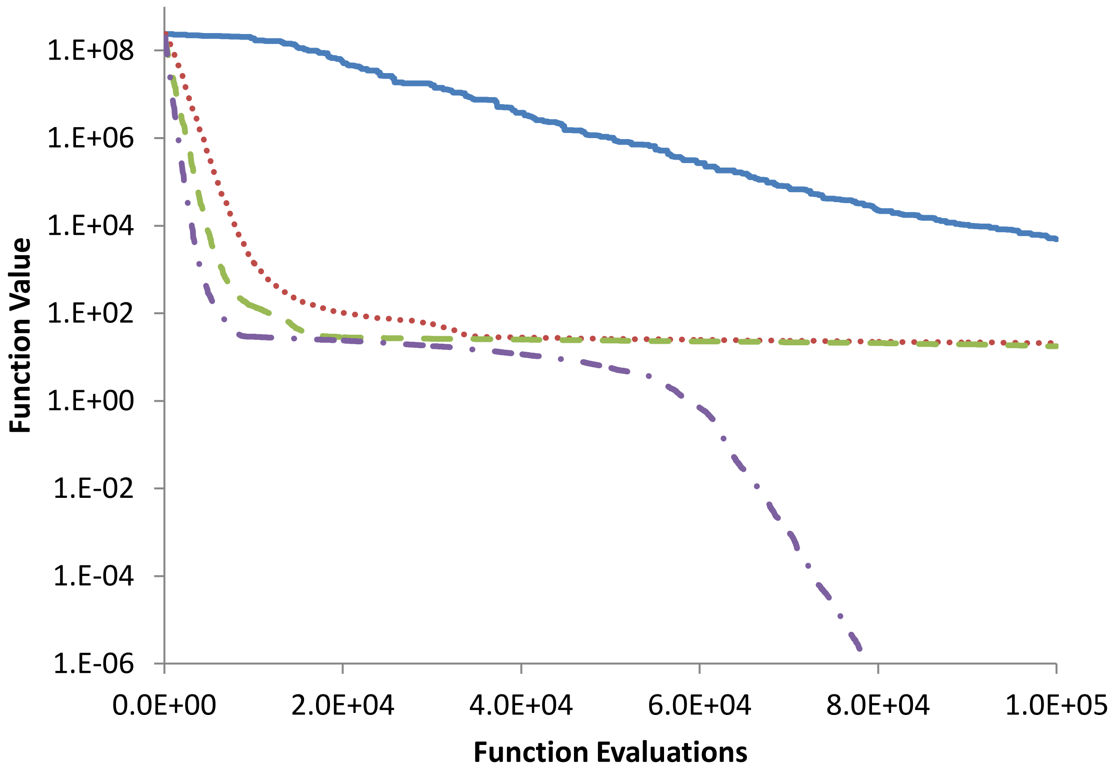
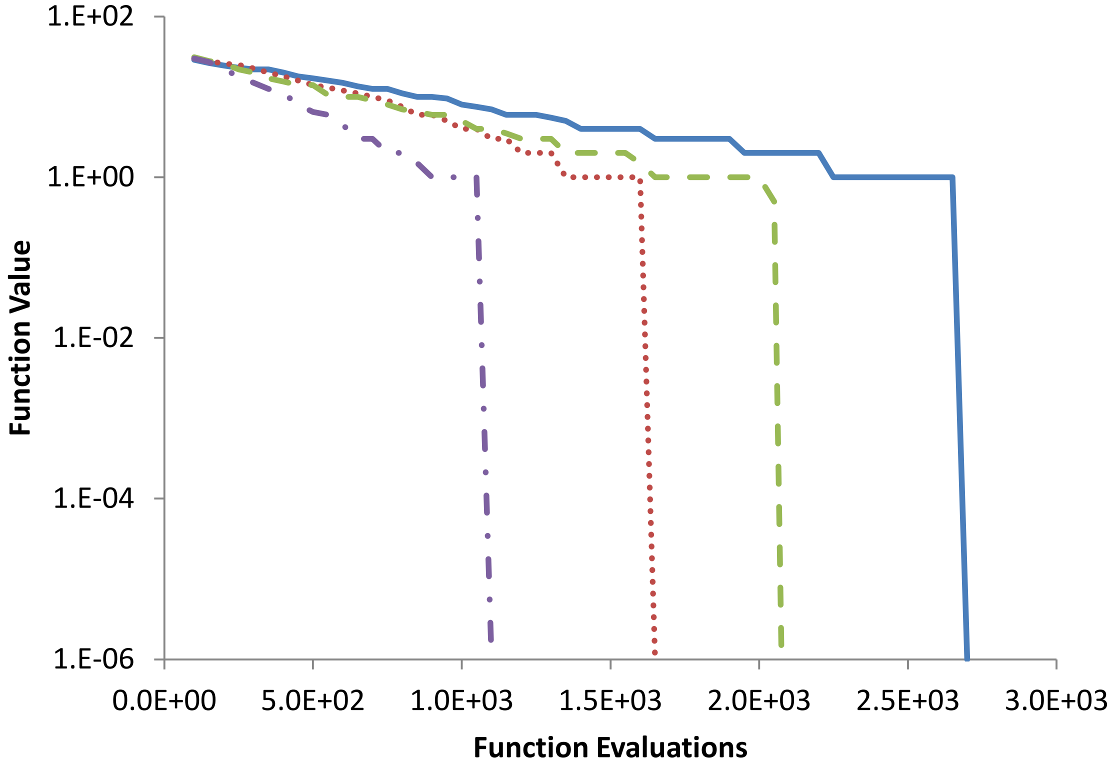


Figure 2.2: Median best-so-far function values for f_2 (Rosenbrock) in 10D (top) and 30D (bottom)



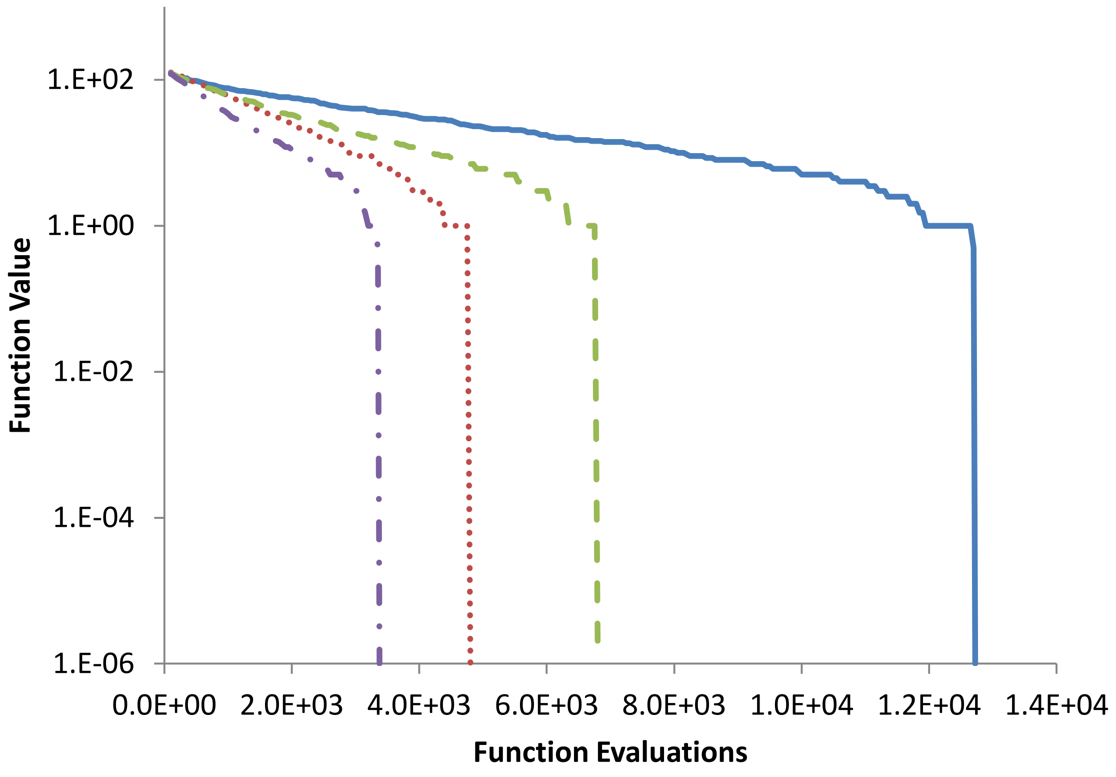
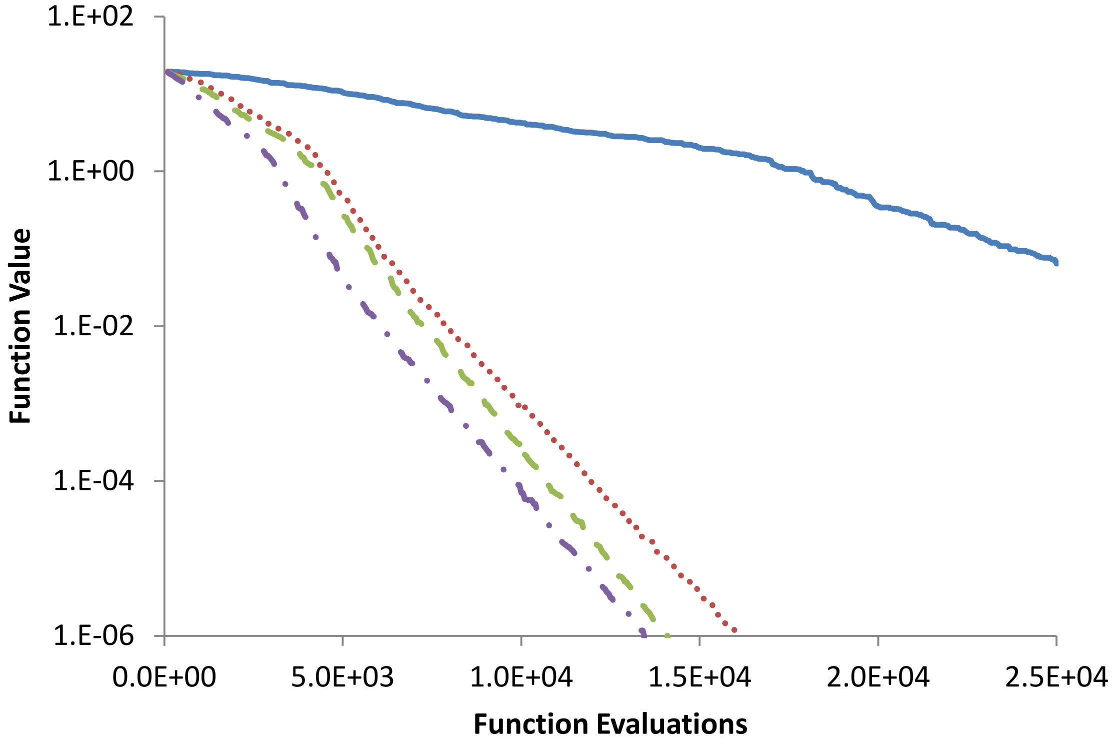


Figure 2.3: Median best-so-far function values for f_3 (Step) in 10D (top) and 30D (bottom)



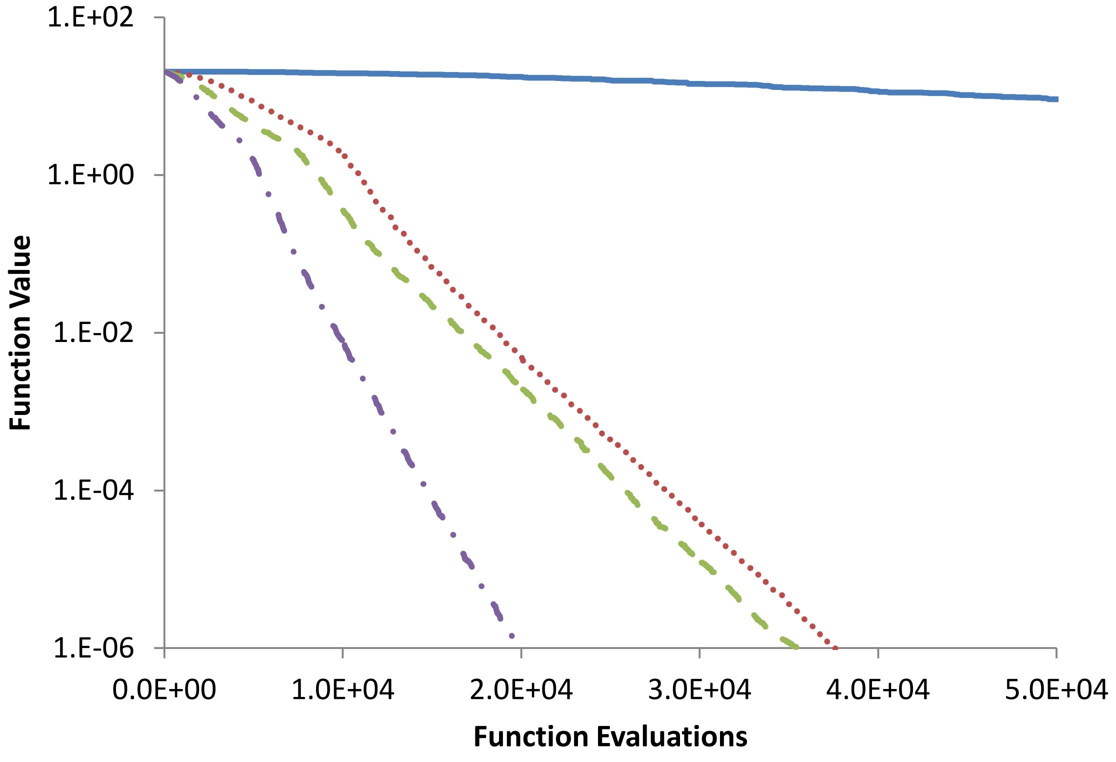


Figure 2.4: Median best-so-far function values for f_4 (Ackley) in 10D (top) and 30D (bottom)

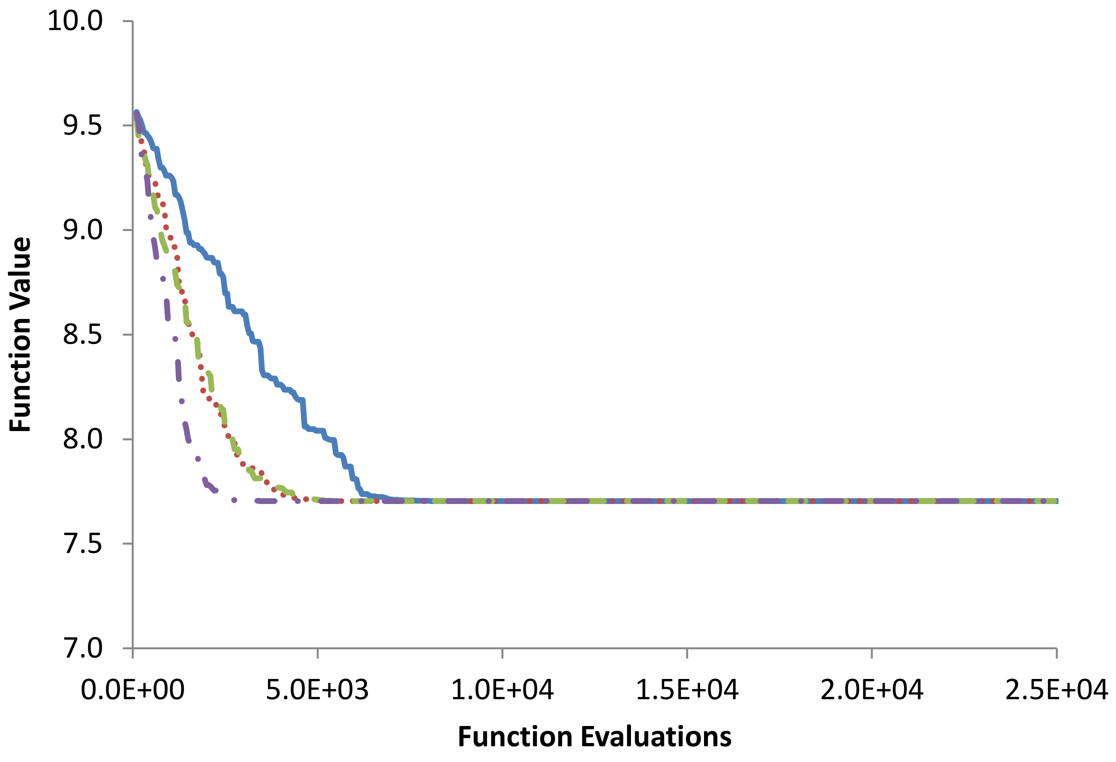


Figure 2.5: Median best-so-far function values for f_5 (Shekel's foxholes) in 5D

2.4.4 Benchmarking Results Discussion

It is evident first of all that each of the adaptive algorithms performed much better than classic DE. DE/rand/1/bin exhibited the slowest convergence rates in every problem graphed above. It was also badly affected by the so-called 'curse of dimensionality', falling further behind the other algorithms at higher D.

SaDE was found to converge fractionally faster that jDE on all functions except f_3 (Figure 2.3), with the two algorithms showing similar convergence consistency in Table 2.1. jDE's respectable performance was noteworthy as its self-adaptive mechanism is much simpler than SaDE's. With reference to Table 2.1, SaDE's performance as the most consistent algorithm for solving f_5 was not considered to be statistically significant. None of the algorithms performed well on f_5 as the function landscape is 'deceptive': there is no clear descent path to the global minimum; therefore, successful convergence is based largely on good luck.

JADE, however, emerged a convincing victor in terms of both consistency and speed. In isolation, the DE/current-to-*p*best/1/bin strategy was found to be highly effective when compared with other common DE strategies in a preliminary study. Further, the use of a single generation LP, in combination with an under-relaxation factor, allowed JADE to ascertain the appropriate control parameters correctly and more rapidly than its rivals. This is in strong contrast with SaDE, which was generally slow to adapt the appropriate F and cr. Zhang and Sanderson (19) attribute the particularly impressive performance of JADE at higher dimensionality (observing both graphs in, for example, Figure 2.4) to the archive mechanism, which was found to make little difference at $10D$ or below in preliminary studies.

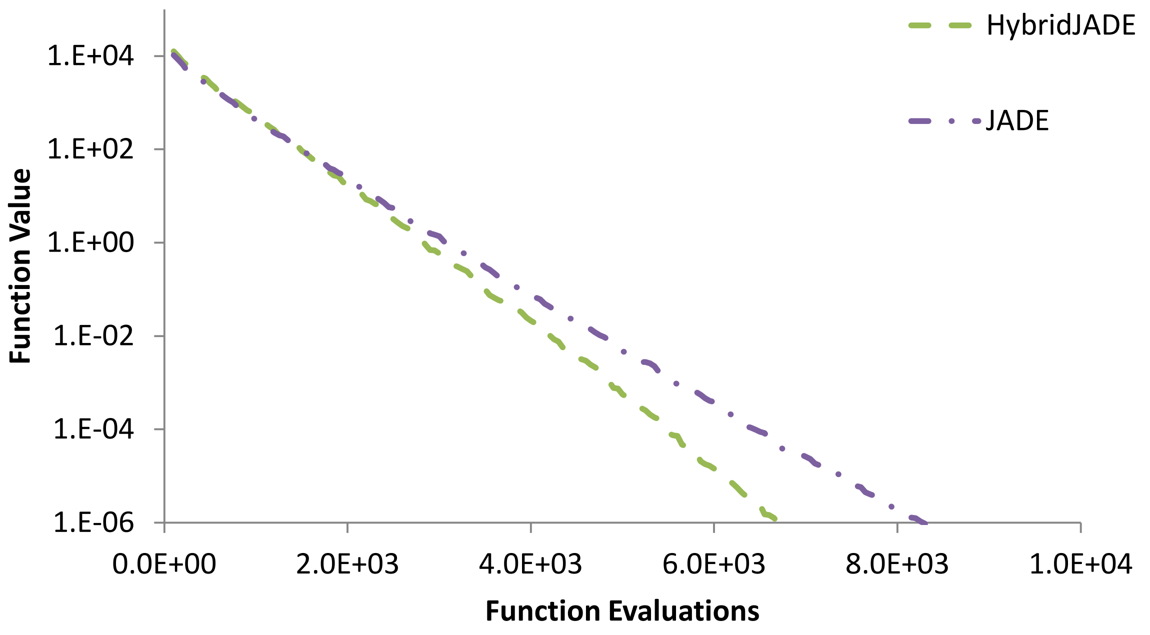
2.5 Hybrid and Original Algorithms

A number of novel algorithms, derived by cannibalising jDE, SaDE and JADE, were developed in an attempt to create a yet higher-performing DE variant. Several experiments showed promise: for example, using JADE's F and cr adaptation strategy improved the performance of SaDE, and a jDE variant incorporating multiple DE strategies outperformed the original jDE. However, neither of these strategies consistently outperformed JADE.

Original adaptive controls tested included a binary cr, selected at random from 0.05 and 0.95. One value would become increasingly probable over the course of the optimisation process in proportion with the number of successes achieved by each. The premise of this idea was that good cr values tend to be either low or high: values around 0.5 promote neither an independent coordinate-based search (as with low cr), nor contour matching (as with high cr). Unfortunately, this technique transpired to be no more effective than the method used by JADE, and was therefore discarded.

The best algorithm generated from the hybrid experiments incorporated jDE-like mutation scaling factor adaptation into JADE, and was dubbed HybridJADE. The resulting algorithm was found to be superior to JADE for simple problems at low dimensionality, but inferior for complex problems at high dimensionality on a 30-run benchmark test (Figure 2.6). It is hypothesised that higher F values, as promoted by the Lehmer mean, are more effective in realising convergence than the lower F values promoted by jDE at higher dimensionality.

The decision was taken that performance in complex, high-dimensional environments was desirable in an FEA optimiser; hence, the as-published JADE algorithm was selected to undergo further work.



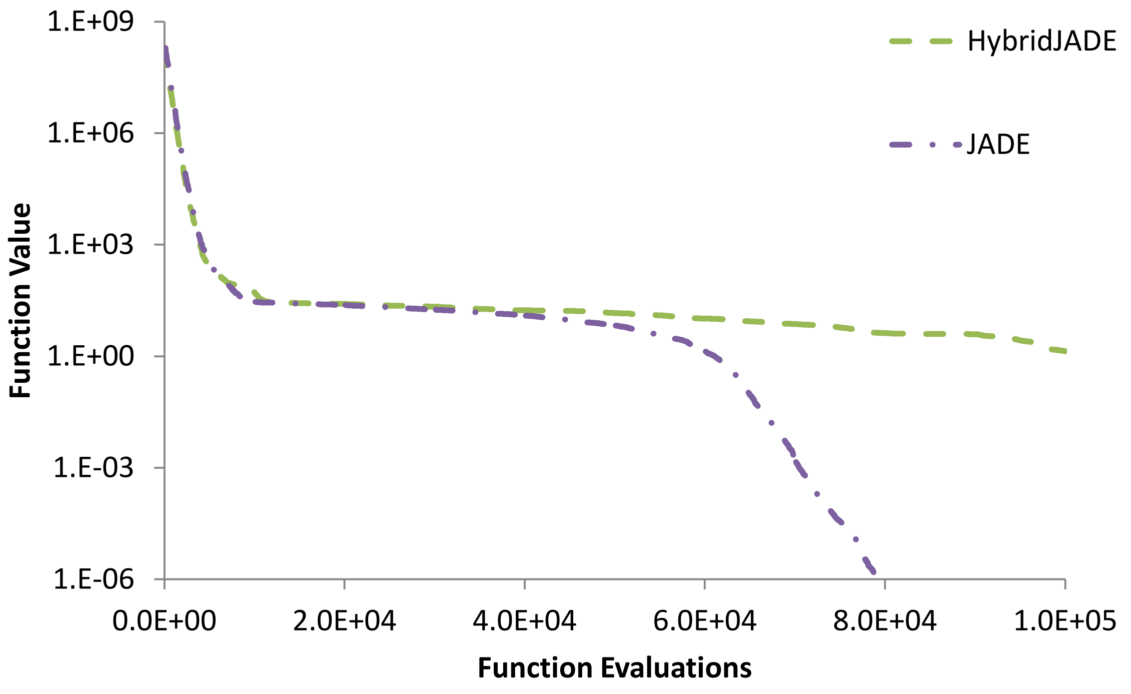


Figure 2.6: Comparison of JADE and HybridJADE Algorithms by median best-so-far function values. HybridJADE excels on simple functions e.g. 10D f_1(sphere, top). JADE performs better on complex functions e.g. 30D f_2(Rosenbrock, bottom).

2.6 Population Size Adaptation

Previous attempts by researchers to automatically adapt DE's final parameter – Np – have been abortive.

Brest *et al.* (26) created a variant of jDE which reduces Np as the optimisation progresses. Impressive results were published; however, this algorithm introduced a population reduction frequency parameter, which had to be well-chosen for each specific problem. Naturally, this was self-defeating for an adaptive algorithm.

Teo (27) devised a complex closed-loop adaptive population strategy. Based on the progress of the optimisation, Np was updated each generation. However, Teo's algorithm only outperformed classic DE on one of the five De Jong test functions.

The potential for reframing the Zaharie limit for critical F in terms of a critical Np was considered in this research; however, this only establishes a minimum permissible Np, not an optimal one. Curve-fitting F, cr and number of function evaluations vs. Np to ascertain an optimum-Np rule was also considered; however, it became apparent that the optimum Np is a strong function of problem complexity, and only a weak function of the other control parameters.

Np adaptation is therefore an open research problem, requiring further work. The decision was taken to retain Np as a user-defined parameter in this study.

2.7 Local Search

A good deal of recent optimisation research concerns the enhancement of global optimisers, such as DE, by hybridising them with local search heuristics. Noman and Iba (28) divide such heuristics into two categories:

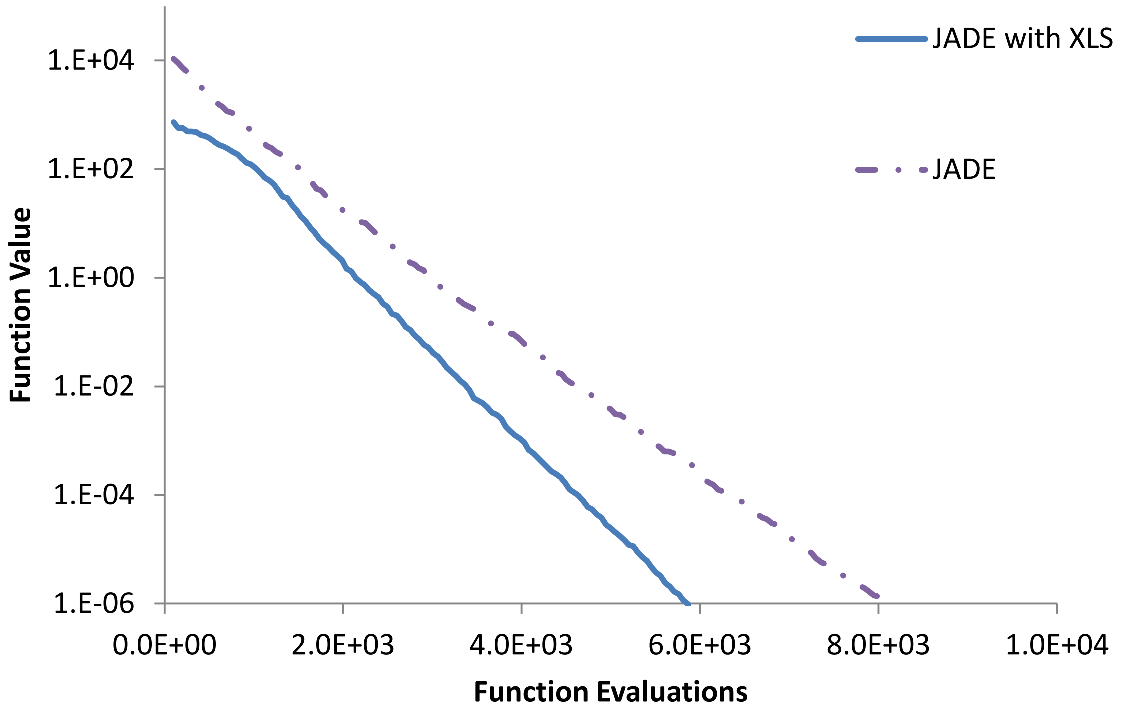
1. *Local Improvement Process* (LIP) algorithms apply gradient-descent based procedures to individual vectors. The original SaDE algorithm (29) applies such a method at $G = 200$.
2. *Crossover-Based Local Search* (XLS) algorithms recombine the members of the population to generate new trial vectors.

Given that the aim of this project was to generate an FEA optimiser, LIPs were not explored further as their effectiveness is known to be weak in discrete function landscapes.

Noman and Iba (28) propose a variety of XLS strategies, many of which are based on a 'centre of gravity' based search, exploring the mean of two or more vectors in the population. Keen to avoid digressing into what is, in itself, a detailed research topic, this study attempted to combine JADE with the following basic XLS algorithm:

- At each generation, evaluate the mean vector of the population as a whole.
- If the function value of the mean vector is lower than that of the median function value in the population, replace the worst vector in the population with the mean vector before the main tournament selection.

In conjunction with JADE, this algorithm showed great promise on lower-dimensional functions in preliminary tests, even when the initialisation region was shifted so that the known function minimum did not lie in the centre. However, its potency was greatly diminished at higher dimensions. Figure 2.7, taken from a full benchmark run with 30 repeats, illustrates this. Hence, XLS was not incorporated into the final FEA optimisation algorithm. The inclusion of more sophisticated XLS extensions is a possible avenue for further work.



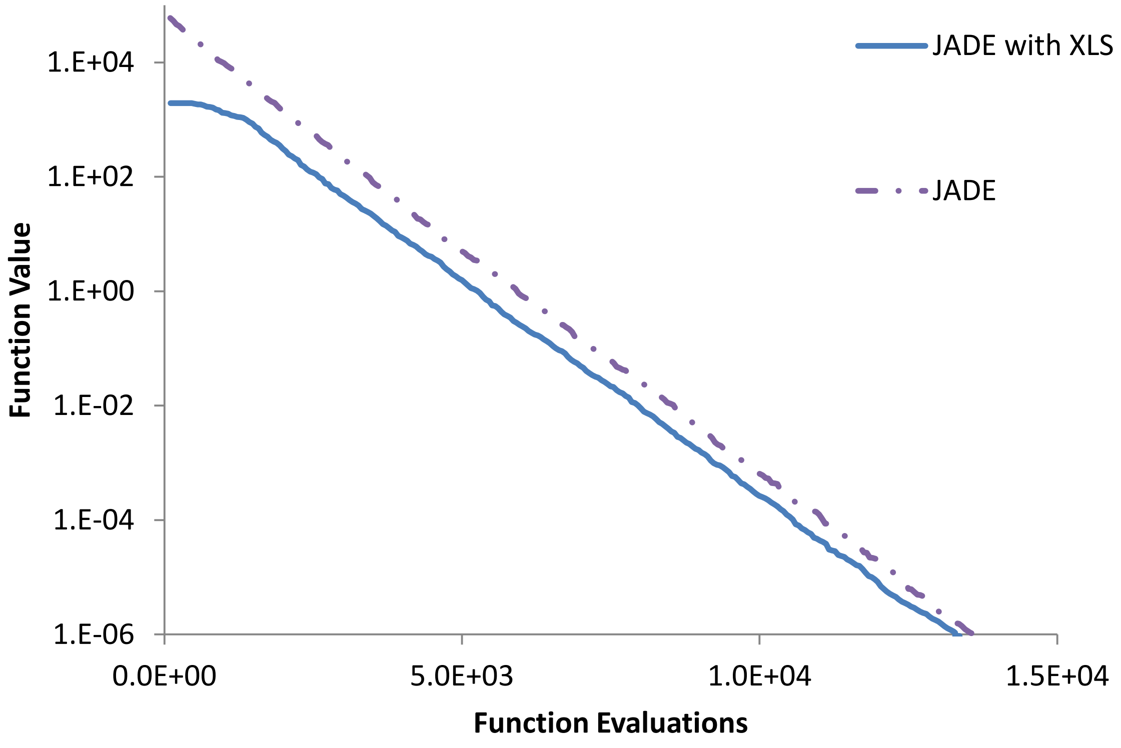


Figure 2.7: Comparison of JADE with and without XLS by median best-so-far function values. XLS showed promise on low-dimensional functions, e.g. 10D f_1(sphere, top); however, only it outperformed JADE at higher dimensions, e.g. 30D f_1(sphere, bottom) because the known minimum was at the centre of the initialisation region.

3 APPLICATION TO FINITE ELEMENT ANALYSIS

3.1 Node-based DE

Normally, DE algorithms operate in the continuous domain. Vectors in the population may take any real value. By contrast, FEA uses a system of nodes, which are discrete points in space. Nodes are the only valid locations for the attachment of conventional connector elements representing bolts, and for the extraction of outputs such as forces and displacements. Applying DE in a nodal environment presented challenges, as well as opportunities for simplification, as described below.

3.1.1 Quantisation

When applying DE to discrete problems, Lampinen and Zelinka (30) note that vectors should continue to be evolved and manipulated in the continuous domain, and only quantised at the point of function evaluation. Hence, the requirement arose to efficiently quantise continuous vectors to nodes.

This requirement was satisfied by implementing a nearest-neighbour search based on a k-d tree. Through repeated queries to this k-d tree, a continuous vector of the form $(x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n)$ may be converted to a list of nearest nodes $(n_1, n_2, \dots, n_n)$.

3.1.2 Commutativity of Nodes

It was envisaged that node ordering would be commutative in all bolted joint problems of interest to this study. In other words, placing bolts at nodes A, B and C would be equivalent to placing bolts at B, C and A, and all other possible permutations. Hence, an additional method was added to sort the population vectors by x, y and z coordinates (in descending order of priority) before nodal quantisation takes place.

3.1.3 Uniqueness of Nodes

It was further assumed that any potential solution should quantise to a unique set of nodes, i.e. two bolted joints may not occupy the same point in space. The algorithm was modified such that if two coordinate groups (x_1, y_1, z_1) and (x_2, y_2, z_2) in the same trial vector quantise to the same node n, the trial vector is not directly evaluated. Instead, such a vector is given a 'penalty' cost of infinity, making it ineligible for selection into the next generation.

3.1.4 Termination Criterion

In Section 2, convergence was judged based on the best-so-far cost reaching a known value. For most real-world problems, this value-to-reach is not known *a priori*; therefore, another termination criterion must be used.

When using DE in a node-based environment, a natural termination criterion for the optimisation process presents itself. The optimisation may be considered complete when all members of the population have converged on the same node or nodes.

3.1.5 Lookup for Previously Visited Nodes

From the perspective of computational efficiency, it is not desirable to evaluate all vectors in the continuous domain that quantise to the same nodes in the discrete domain. Therefore, a lookup system was established, such that each unique set of nodes was analysed only once in FEA.

3.1.6 Automatic Creation of Bounds

In Section 2, the problem initialisation boundaries for the benchmark functions were defined manually. However, it is possible to infer the problem boundaries from the feasible nodes of an Abaqus model instead, eliminating this requirement and simplifying the user interface. The automatically generated problem boundaries will form the smallest possible cube, aligned with the global Cartesian coordinate system, encompassing all feasible nodes, as shown in Figure 3.1.

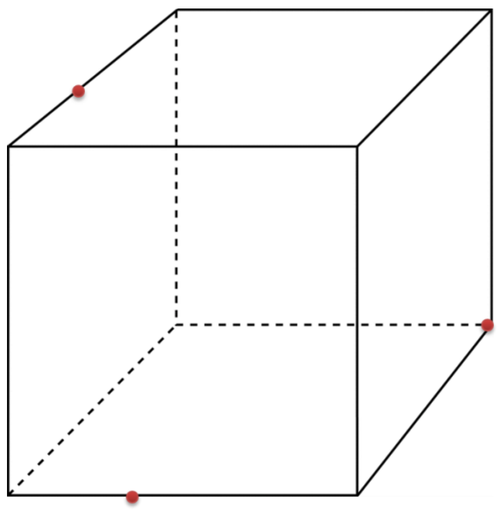


Figure 3.1: Smallest possible cube (black) encompassing three nodes (red)

In FEA, problem bounds will always be ‘absolute’ as exploration outside the bounds is meaningless. Therefore, mutant vectors straying outside the problem bounds were truncated to lie on the boundary in this study.

3.1.7 Phantom Dimensions

The Abaqus Python API works with nodal coordinates in three dimensions, even when the problem under consideration is only one- or two-dimensional. For these lower-dimensional problems, or for specific cases of three-dimensional problems, variables may exist in the optimisation that are constrained to have a constant value. These variables are ‘phantom dimensions’.

This poses problems for the crossover procedure, which normally guarantees that at least one mutant variable is carried forward into the corresponding trial vector, as noted in Section 2.1. If this randomly chosen variable is a phantom dimension, then it is possible that a trial vector may be identical to its parent, which would cause the optimisation process to stagnate.

It was therefore necessary to adjust the DE code to note phantom dimensions at the start of the analysis, which may be inferred from the problem boundaries. The crossover method was modified to guarantee that at least one non-phantom variable would be carried through to the corresponding trial vector.

3.2 Parallelisation

Price *et al.* (6 p. 267) remark that the need for parallel processing "is particularly acute when optimising models based on simulations".

Initially, the scope for parallelising the pre-processing, running and post-processing of FEA models was considered. However, the idiosyncrasies of the parallel computing facilities available at the University of Sheffield, known as *Iceberg*, with regards to process queuing rendered this infeasible. Instead, a 'farming' model based on parallelisation of FEA runs only was developed. As shown in Figure 3.2, the main thread carries out all of the tasks illustrated by full-width blocks, whereas the runs are carried out in parallel across a network of n CPUs. In practice, this was achieved by submitting each Abaqus run to *Iceberg*'s queuing engine, and waiting for the queue to become empty before post-processing.

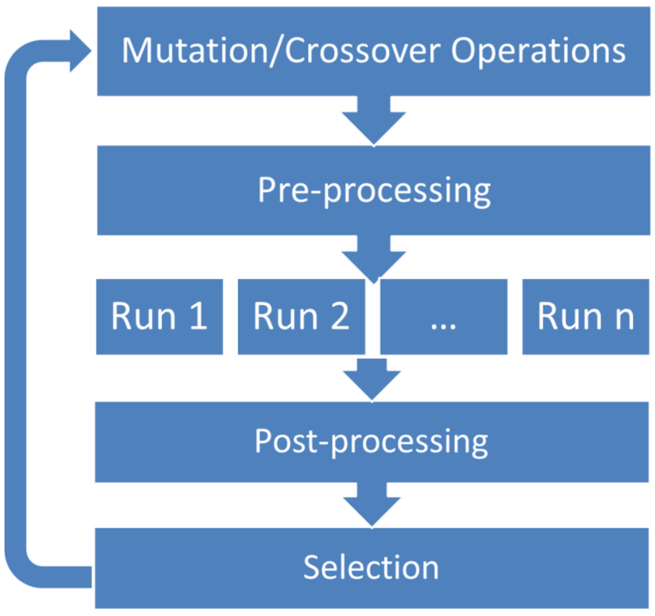


Figure 3.2: Parallelisation of FEA runs

3.3 Completed AbaqusJADE Algorithm

A flowchart of the completed 'AbaqusJADE' algorithm's generate-and-test loop is shown in Appendix 3 for reference. The full source code is attached on disc.

3.4 Trial Case: Supporting a Beam

As a simple, experimental and illustrative case, the newly-developed AbaqusJADE algorithm was applied to a problem for which the solution was known *a priori*. A one metre long one-dimensional beam, discretised into 500 ‘B21’ beam elements, under the action of a distributed load, was considered as the objective case. Three bolted joints, represented simply as encastré boundary conditions, were to be positioned at the optimum locations along the beam to minimise the peak deflection. The optimum locations may be computed from Euler-Bernoulli beam theory as 138mm, 500mm and 862mm from one end of the beam. Np was set to $5D = 5 \times 3 = 15$.

The analysis converged upon the correct nodes after 55 generations, having conducted 590 unique Abaqus runs. The optimum vector was discovered at generation 32.

Figure 3.3 to Figure 3.5 depict the best-so-far solutions at generations 0, 5 and 32 respectively, and show the analysis gradually adapting the best bolting locations, identifiable as the points of zero displacement. Displacement units may be considered relative.

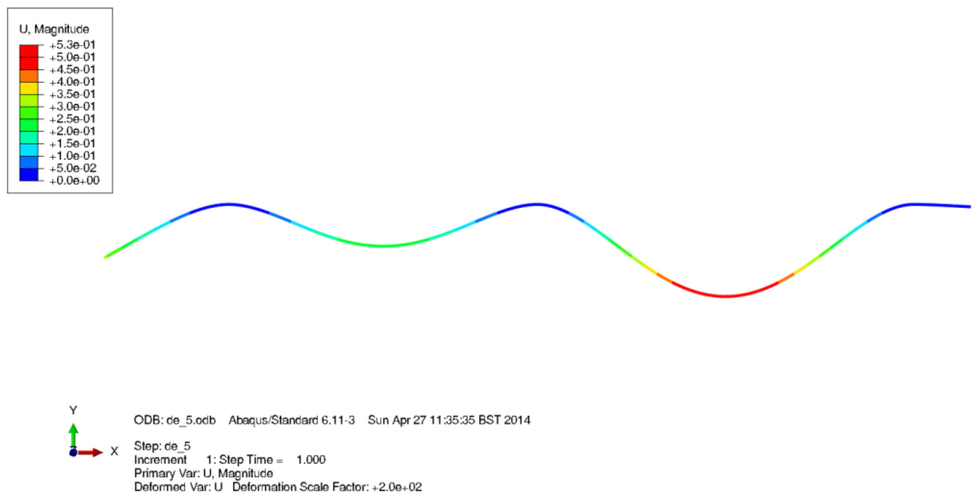

U, Magnitude
+5.3e-01
+5.0e-01
+4.5e-01
+4.0e-01
+3.5e-01
+3.0e-01
+2.5e-01
+2.0e-01
+1.5e-01
+1.0e-01
+5.0e-02
+0.0e+00
Y
X
ODB: de_5.odb Abaqus/Standard 6.11-3 Sun Apr 27 11:35:35 BST 2014
Step: de_5
Increment 1: Step Time = 1.000
Primary Var: U, Magnitude
Deformed Var: U Deformation Scale Factor: +2.0e+02

Figure 3.3: Best-so-far beam optimisation for minimum deflection at Generation 0

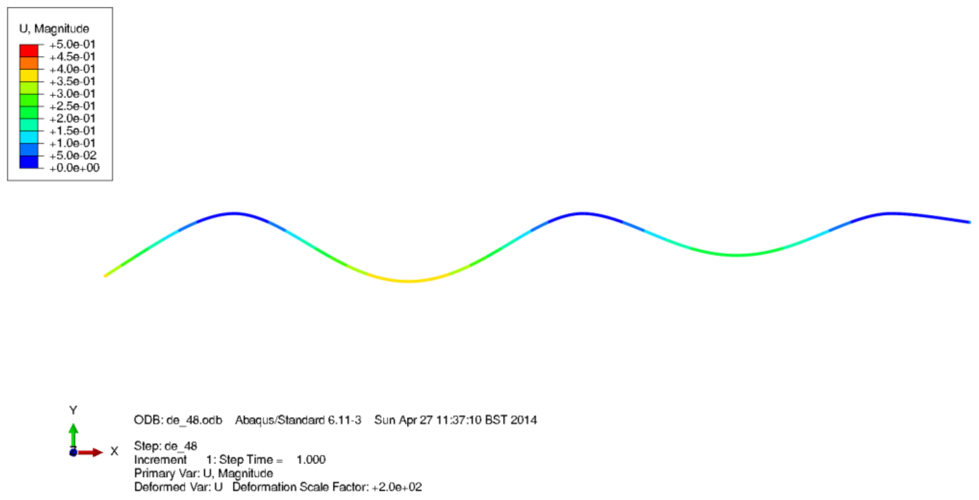

U, Magnitude
+5.0e-01
+4.5e-01
+4.0e-01
+3.5e-01
+3.0e-01
+2.5e-01
+2.0e-01
+1.5e-01
+1.0e-01
+5.0e-02
+0.0e+00
Y
X
ODB: de_48.odb Abaqus/Standard 6.11-3 Sun Apr 27 11:37:10 BST 2014
Step: de_48
Increment 1: Step Time = 1.000
Primary Var: U, Magnitude
Deformed Var: U Deformation Scale Factor: +2.0e+02

Figure 3.4: Best-so-far beam optimisation for minimum deflection at Generation 5

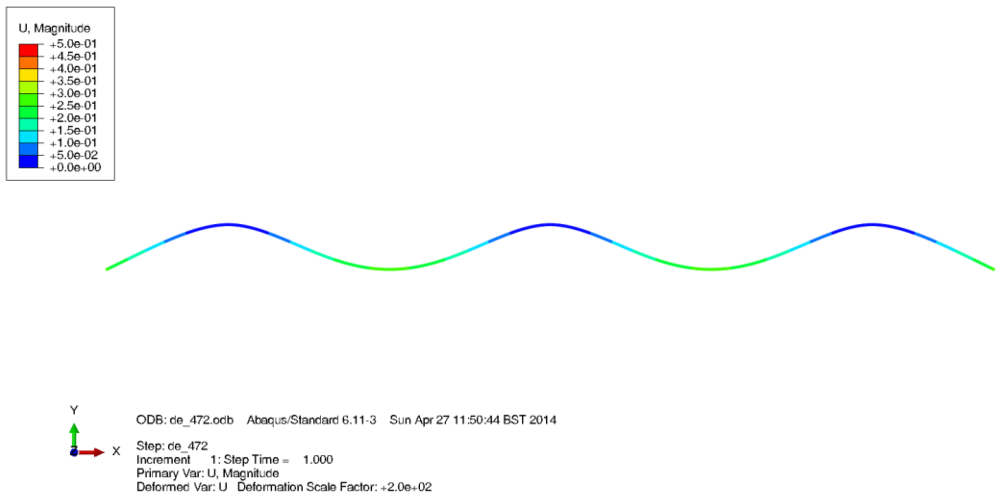

U, Magnitude
+5.0e-01
+4.5e-01
+4.0e-01
+3.5e-01
+3.0e-01
+2.5e-01
+2.0e-01
+1.5e-01
+1.0e-01
+5.0e-02
+0.0e+00
Y
X
ODB: de_472.odb Abaqus/Standard 6.11-3 Sun Apr 27 11:50:44 BST 2014
Step: de_472
Increment 1: Step Time = 1.000
Primary Var: U, Magnitude
Deformed Var: U Deformation Scale Factor: +2.0e+02

Figure 3.5: Best-so-far beam optimisation for minimum deflection at Generation 32

3.5 Objective Case: Bloodhound SSC Jet Fuel Tank

In accordance with the project aim, AbaqusJADE was then deployed to solve the problem of how to bolt the baffle of the Bloodhound SSC jet fuel tank around its perimeter, using 16 bolts, such that the peak deflection of the baffle is minimised.

3.5.1 Model Construction

An Abaqus model of the Bloodhound SSC jet fuel tank baffle was constructed from scratch based on geometric details (Figure 3.6) obtained directly from the Bloodhound SSC team. Only the baffle itself – a 3mm-thick aluminium plate – was explicitly modelled. The precise aluminium grade was not divulged; therefore, the generic properties for aluminium shown in Table 3.1 were taken from Calvert and Farrar (31):

E	70×10^9
ν	0.3
ρ	2700

Table 3.1: Engineering properties (SI) for aluminium plate

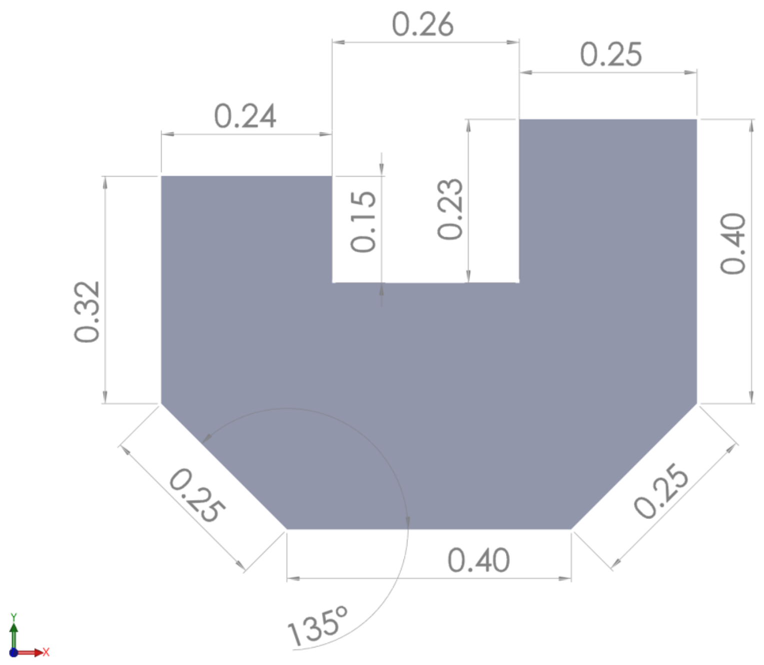


Figure 3.6: Dimensions (m) for Bloodhound Baffle

The baffle was meshed with 806 standard Abaqus shell elements (S3/S4R), which are suitable for thin, membrane-loaded structures. The mesh is depicted in Figure 3.7. A mesh independence study showed that this relatively coarse level of mesh refinement was sufficient to accurately resolve the baffle's displacement field under the loading described later in this section.

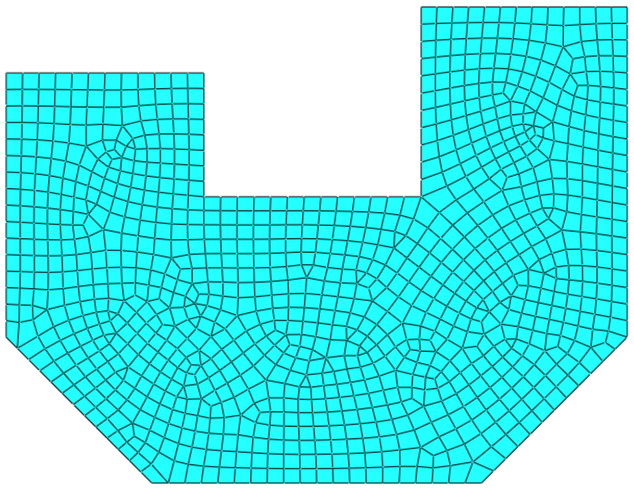


Figure 3.7: Mesh for Bloodhound Baffle

In accordance with the original design concept, the baffle was to be fixed to the composite tank walls using 16x M4 steel fasteners ($E = 210GPa$). In contrast to the trivial case presented in Section 3.4, these fasteners were explicitly modelled using Cartesian/Cardan connector elements. The fasteners were configured to be anchored in all six degrees of freedom 13.2mm away from the baffle, equivalent to points on the 13.2mm-thick composite tank's outer skin (effectively assuming a rigid tank).

The shear stiffness of these fasteners was computed using the Huth method (32), based on curve-fitting of experimental data done by Heimo Huth in 1985. This is a commonly used method in industry, employed by Airbus and Atkins amongst others. According to Huth, the flexibility of a bolted joint in single shear obeys Equation 3.1:

$$f = \left(\frac{t_1 + t_2}{2d_f}\right)^{2/3} \times 3 \times \left(\frac{1}{t_1 E_1} + \frac{1}{t_2 E_2} + \frac{1}{2t_1 E_f} + \frac{1}{2t_2 E_f}\right)$$

Equation 3.1

Note that, in this case, subscript 1 denotes tank properties, subscript 2 denotes baffle properties, and subscript f denotes properties of the fastener.

The composite tank is a sandwich material, consisting of a 10mm foam core between two 1.6mm Carbon Fibre Reinforced Polymer (CFRP) facesheets of modulus 54 GPa (33). Assuming an effective shear modulus of zero for the core, the equivalent modulus for the tank was calculated by Equation 3.2:

$$E = \frac{0.0016 \times 2 \times 54 \times 10^9}{0.0132} = 1.31 \times 10^{10} Pa = 13.1 GPa$$

Equation 3.2

Applying Equation 3.1, the shear flexibility of the joints was taken to be 5.53×10^{-8} m/N. The corresponding shear stiffness is given by the reciprocal, and is equal to 1.81×10^7 N/m.

The tensile stiffness of the bolts was taken as per Equation 3.3:

$$\frac{EA}{l} = \frac{\pi E d^2}{4l} = \frac{2.1 \times 10^{11} \pi \times 0.004^2}{4 \times 0.0132} = 2.00 \times 10^8 N/m$$

Equation 3.3

By convention in industry, the bolts were considered to be infinitely flexible in rotation about their shaft axes, and rigid in the remaining rotational degrees of freedom.

Loading was based on the worst-case braking load anticipated by Bloodhound of $3g$. Conservatively based on a full fuel tank pressure head of 0.5m and a jet fuel density of 800kg/m^3 (values given by the Bloodhound SSC team), this corresponds to the uniform pressure load given by Equation 3.4:

$$p = \rho(3g)h = 800 \times 3 \times 9.81 \times 0.5 = 11800 Pa$$

Equation 3.4

The analyses were run as linear static and did not account for inertial effects. The bolts were attached in the plane of the baffle, as the envisaged mode of attachment to the main tank was via a 90° return in the aluminium, which was not explicitly modelled in this study. Bolt pre-tension was not considered. This whole methodology (the use of the Huth method, linear static analysis, in-plane bolting and no pre-tension) was identical to that used by Atkins in the original project.

3.5.2 Optimisation Results

AbaqusJADE was run three times on the Bloodhound model, using a population size of 40. This Np was based on Storn's heuristic (22). Each run converged upon the same 16 nodes, giving confidence that the solution presented below is the true optimum. In the median case, convergence was achieved after 47 hours and 15 minutes of runtime, and 2512 generations. The other two runs required 2489 and 2601 generations. The median number of function evaluations (simulations) required was 48,716.

Figure 3.8 reveals that the algorithm found relatively good solutions very quickly, placing the bolts in a configuration that resulted in a baffle deflection only 10% greater than the optimum at $G = 158$. However, a true optimum solution was not discovered until $G = 1814$.

Figure 3.9 demonstrates AbaqusJADE's automatic parameter control strategies working on F and cr over the course of the median optimisation run. The algorithm quickly selected a low cr value. This implies that the cost function landscape was locally separable. This was unexpected given that the work done at Atkins showed the function landscape to be globally parameter dependent. Physically, this means that each bolting location may be incrementally optimised in isolation, holding all others constant.

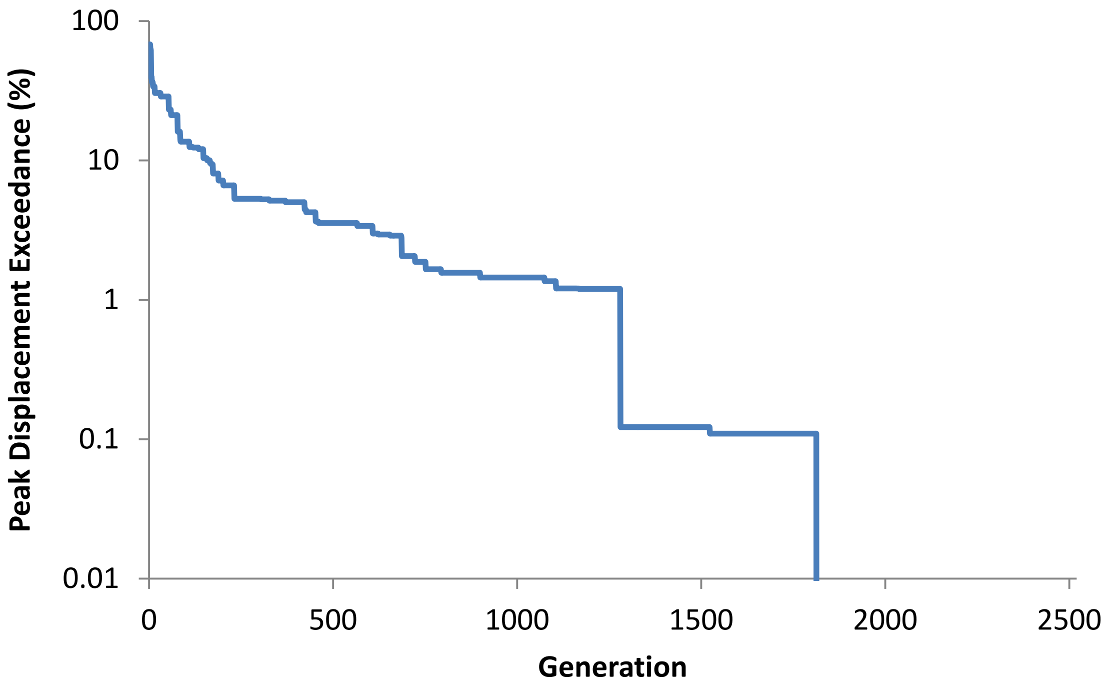


Figure 3.8: Generation vs. best-so-far peak displacement exceedance (median convergence case) for optimisation of the Bloodhound jet fuel tank baffle. Exceedance computed as (y/y_{min}).

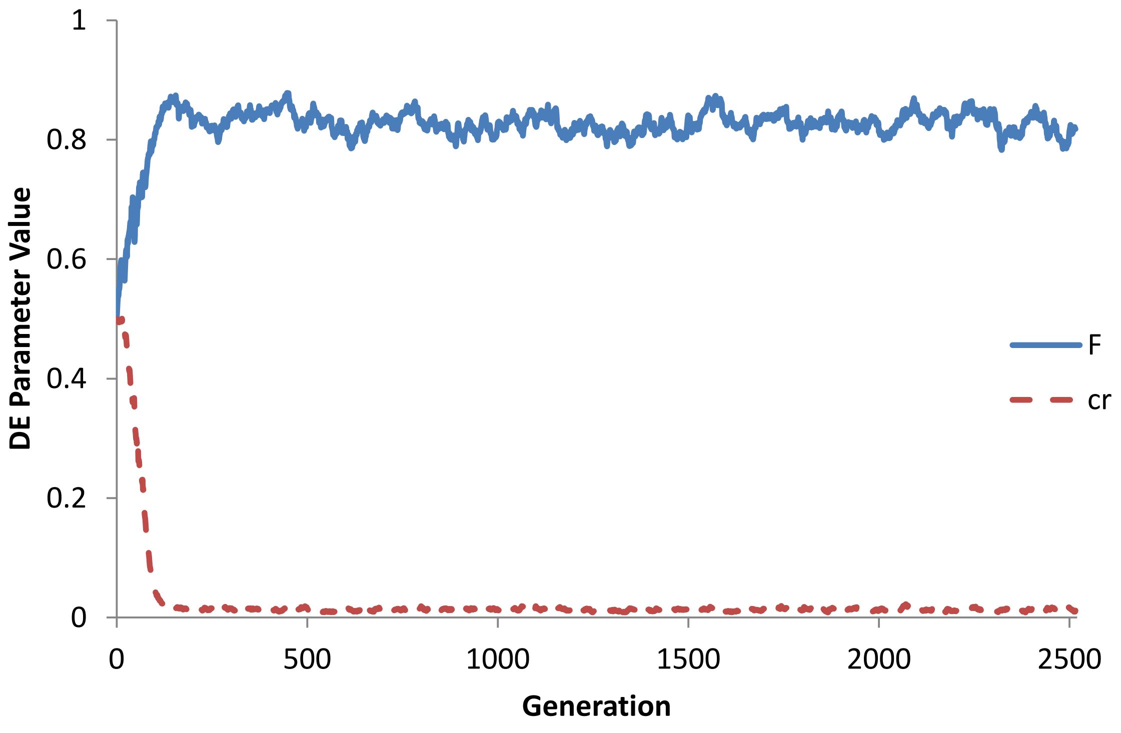


Figure 3.9: Generation vs. DE parameter values (median convergence case) for optimisation of the Bloodhound jet fuel tank baffle.

Figure 3.10 to Figure 3.13 illustrate the progress of the evolutionary process for the median convergence case.

At $G = 0$, the bolting locations (represented by boundary condition symbols) were, as expected, distributed arbitrarily and ineffectively around the baffle. Figure 3.10 demonstrates this. However, by $G = 100$, AbaqusJADE had begun to arrange the bolts in such a way that might resemble an initial guess by a human engineer. As Figure 3.11 reveals, the $G = 100$ solution is approximately symmetrical, with a greater proportion of bolts occupying the perimeter of the largest unsupported span in the centre of the baffle.

By $G = 1000$, a curious phenomenon had begun to arise, with bolts grouping into collocated pairs. This was unexpected, and was initially considered to be the manifestation a programming error. However, upon greater reflection, it was realised that bolts positioned closer together really do impart greater stiffness to the parent components than bolts that are evenly spaced. The relatively large forces invoked to balance the moments between adjacent bolts act to locally preload the baffle, increasing its membrane stiffness.

The optimum solution was obtained at $G = 1814$, and is shown in Figure 3.13. In this solution, most of the bolts are concentrated around the main unsupported span at the centre. 10 bolts have arranged themselves into pairs, thereby imparting maximum stiffness. This solution is non-obvious and may not have been considered by a human engineer.

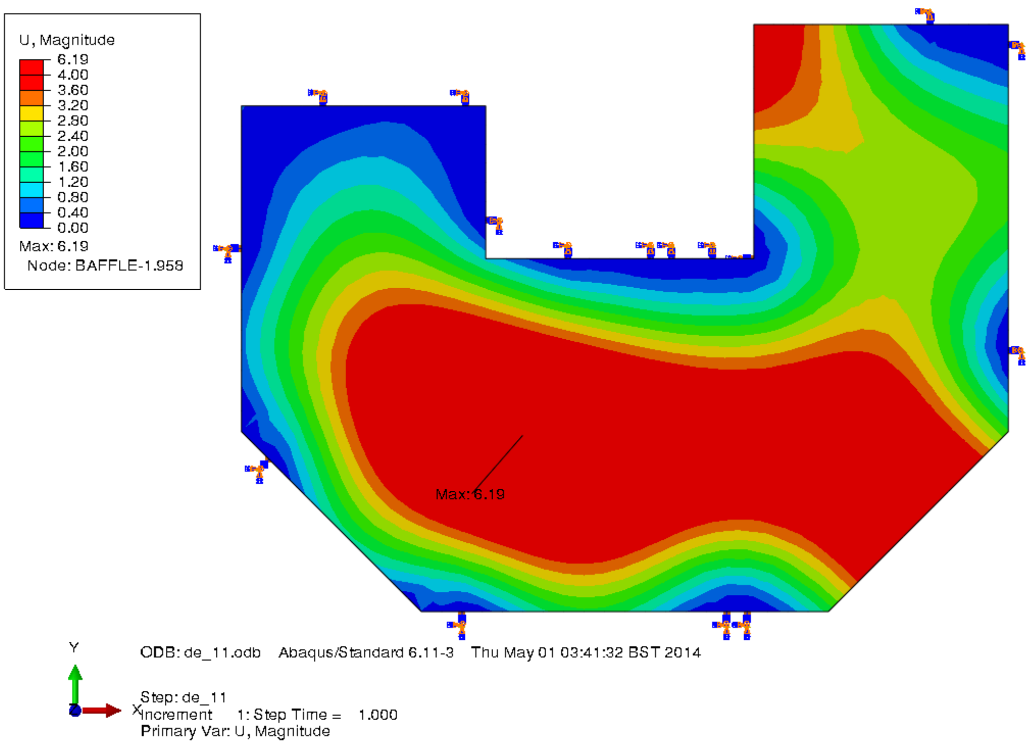


Figure 3.10: Best-so-far solution minimising baffle deflection (mm) at Generation 0

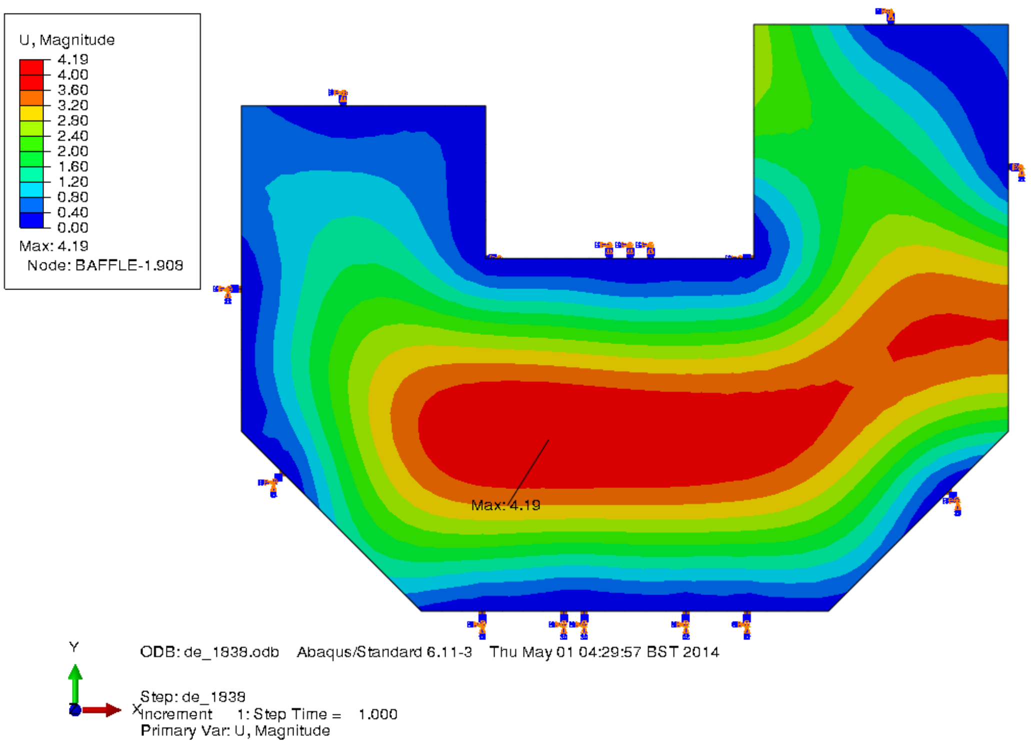


Figure 3.11: Best-so-far solution minimising baffle deflection (mm) at Generation 100

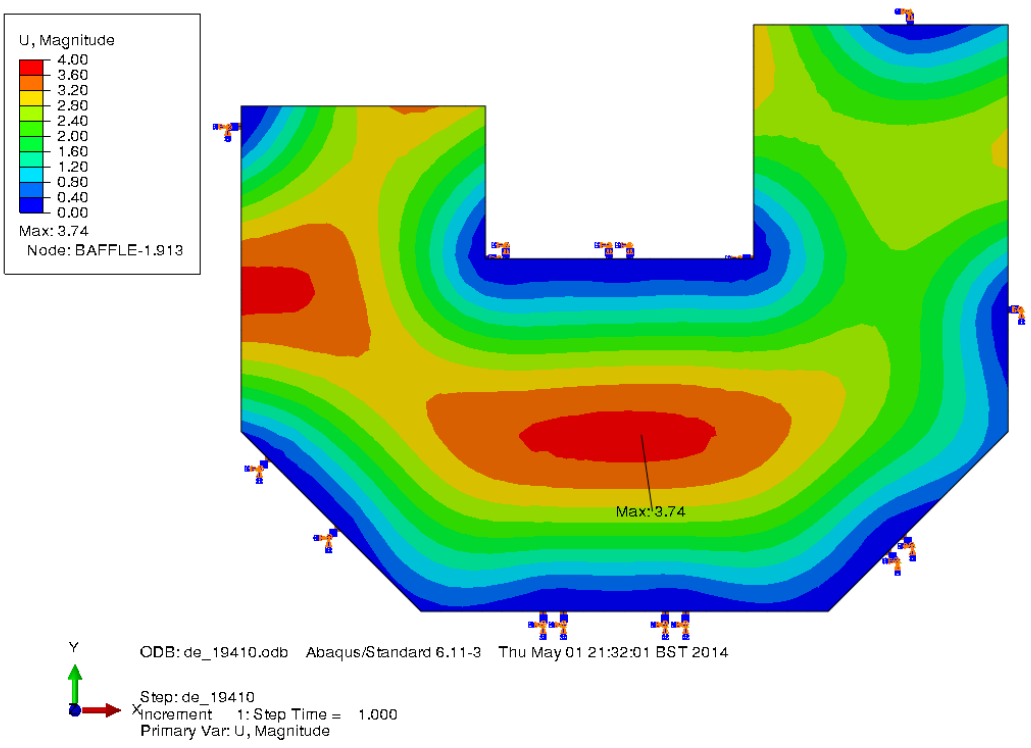


Figure 3.12: Best-so-far solution minimising baffle deflection (mm) at Generation 1000

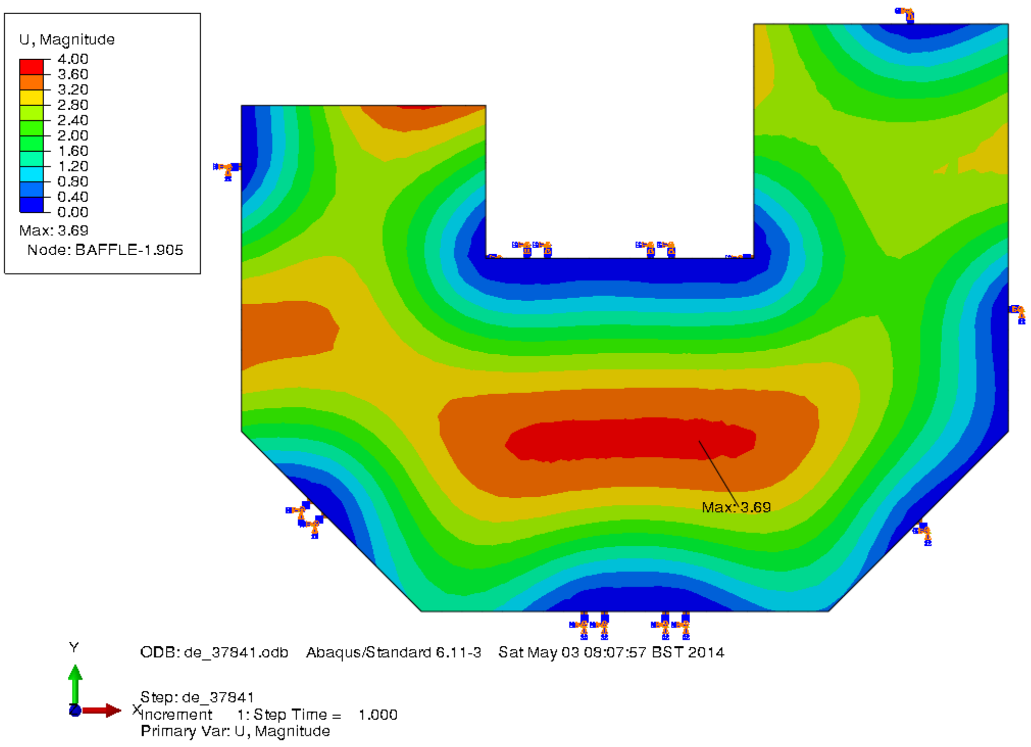


Figure 3.13: Best-so-far solution minimising baffle deflection (mm) at Generation 1814

3.5.3 Model Validation

A hand-calculation based exercise was undertaken based on a simplified rectangular baffle to validate the results presented in Section 3.5.2.

Given that the peak displacement was around the centre of the baffle, the upper flanges and corner cutaways were ignored. These calculations assumed that the perimeter of the baffle was constrained in six degrees of freedom.

The geometry of the simplified baffle was $a = 0.75m$, $b = 0.35m$, $t = 0.003m$. This corresponds to the rectangle given by the red overlay in Figure 3.14.

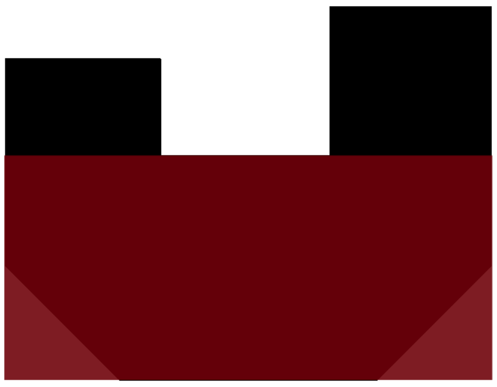


Figure 3.14: Assumed rectangular plate (given by red overlay)

According to Young and Budynas (34) in the industry-standard textbook 'Roark's Formulas for Stress and Strain', the deflection of such a plate is given by Equation 3.5, where α is an empirically derived constant based on the value of a/b:

$$y = \frac{\alpha p b^4}{E t^3}$$

Equation 3.5

Taking the value of a/b to be approximately equal to 2 gives $\alpha = 0.0273$. Therefore, by Equation 3.6:

$$y = \frac{0.0273 \times 11772 \times 0.35^4}{70 \times 10^9 \times 0.003^3} = 2.55 \times 10^{-3} m = 2.55mm$$

Equation 3.6

As would be expected, this value is slightly lower than the deflection observed in Section 3.5.2, given that the FEA model is bolted, not built-in around its perimeter. Nevertheless, the value is of the correct order of magnitude, giving confidence in the FEA results. Further, the strong dependence of y on b justifies AbaqusJADE's solution, which places particular emphasis upon making the baffle as rigid as possible in the b direction by clustering bolts around this unsupported span.

Naturally, these results should be taken with caution until a full experimental validation exercise is undertaken. In particular, the grouping of the bolts into pairs should be closely investigated as this may be an artefact of the modelling approach used. This study did not undertake such an exercise as the primary aim of these simulations was to demonstrate the efficacy of an evolutionary approach in FEA only. It is noteworthy, though, that the accuracy and validity of any FEA output will be limited by the accuracy of the model. It is acknowledged that the physical Bloodhound jet fuel tank baffle will undergo dynamic loading, use bolt preloads, incorporate a 90° return in the aluminium, and exhibit nonlinear membrane effects. All of these factors may subtly affect the optimal bolted joint configuration.

4 DISCUSSION

Reflecting upon the project aim given in Section 1.3, it is apparent that this study has been broadly successful. An FEA optimiser was realised, capable of optimising the bolting locations around the Bloodhound SSC jet fuel tank baffle for minimum peak deflection, without the requirement for manual iteration.

However, the project aim placed emphasis on computational efficiency. The protracted runtime observed in the objective case remains a key barrier to the widespread industrial uptake of what might be called Evolutionary Finite Element Analysis (EFEA) techniques. Secondly, a point neglected thus far is that an optimum bolted joint configuration is usually based on multiple criteria, rather than just one, as was the case in this analysis. Therefore, there is a need for multi-objective optimisation logic to be built into a commercially viable optimiser.

The discussion that follows is framed around the abovementioned barriers to the commercial implementation of EFEA and how they could be overcome. Once these barriers are removed, a number of ideas are presented for how EFEA might be extended and developed as a software product.

4.1 The Problem of Protracted Runtime

The Bloodhound model required around two days to achieve convergence, despite the fact that each single run took less than 10 seconds. Although this is still less than the five working days that it took to manually optimise the baffle at Atkins, it is noteworthy that most real-world problems that require optimisation are larger, and may be multi-phase and/or nonlinear. Single runs of such models may require several minutes, hours or even days with modest computing resources. Therefore, the time ‘ballooning’ for the optimisation run seen in this study would be simply unacceptable.

One way in which this could be mitigated is to run the optimisation on a dedicated computing resource. A distinct disadvantage of using a shared computing resource such as *iceberg* is the necessity to wait in a queue to run simulations. Given a dedicated computing resource with 40 CPUs, each generation for the Bloodhound problem could have been processed in the time taken for a single run. This would have reduced the median convergence runtime (2512 generations) to $10 \times 2512/(60 \times 60) \approx 7$ hours.

There is also scope for improving the problem conditioning before undertaking each run. The Bloodhound problem used global Cartesian coordinates; therefore, the problem presented for optimisation was 32-dimensional. It would be possible to articulate this problem as 16-dimensional by using, for example, the clockwise distance from a fixed point on the baffle as the sole coordinate reference. Although no experiments were undertaken using this technique, evidence from the benchmark functions presented in Section 2.4.3 suggests that reducing problem dimensionality by a factor of R reduces the number of required function evaluations by the same factor. Further, the use of a single coordinate reference would establish a more direct causal relationship between the independent variables (the pre-quantised vector components) and the dependent variable (the maximum deflection based on the quantised vector components), thereby aiding convergence.

The trade-off for using this technique would be a significant increase in time investment and programming effort on the part of the analyst setting up the problem. Although the standard of programming ability is beginning to improve in professional engineering, the personal experience of the author is that it remains low in general. Therefore, any FEA optimisation technique requiring significant programming effort would be unlikely to gain traction in industry.

A third way of mitigating the problem of protracted runtime would be to continue research into the adaptive and local search techniques elucidated in Section 2. Although a variety of ideas were tested in this section which, disappointingly, transpired to be ineffective despite the significant time investment, the strategies tried were not exhaustive. Other, as-yet untested, parameter adaptation methods may bear fruit in the future. For example, the problem of automatic Np control remains one of the key open questions in DE research. In addition, it is likely that an FEA analyst may be put off from using this technique if there is a requirement to select control parameters manually.

It may even be possible to hone the JADE algorithm further by fine-tuning its internal workings. Candidates for optimisation include the under-relaxation factors and distribution deviations used in the algorithm. One way to tune these values would be to use a trial-and-error approach using a sufficiently broad testbed of benchmark functions. Alternatively, one could use an evolutionary approach, perhaps even DE itself, to ascertain the most optimal values for the general case.

Lastly with respect to the algorithm, refining the mutant vector truncation scheme described in Section 3.1.6 may also accelerate convergence. Price *et al.* (6 p. 206) found that "setting parameter values equal to the bounds they violate should be avoided because it lowers the diversity of the difference vector population". Regrettably, this particular piece of literature was discovered after the objective case optimisation runs were undertaken. Time constraints prohibited re-running the optimisation with an improved technique.

Moving on, improvements in computer hardware itself may help to make EFEA viable in coming years. The development of computers with greater numbers of CPUs will be particularly pertinent, not only for parallelising a large population of Np individuals across Np processors, but also for parallelising individual runs. FEA simulations internally parallelise almost linearly; therefore, an EFEA run with population size Np on a machine with $a \times Np$ CPUs may be accelerated by a factor of a relative to a machine with Np CPUs. Further, Moore's law (15), which states that the number of transistors on integrated circuits doubles approximately every two years, shows no sign of abating at present. Therefore, a processing speed increase of at least 10 times per CPU would appear to be a conservative expectation within the next 10-20 years.

An exciting, if distant, vision for EFEA may be realised by the advent of quantum computation. A quantum computer exploits quantum mechanical phenomena to perform multiple calculations simultaneously. If FEA solvers can harness this capability, even the largest, highly nonlinear FEA models may be able to run in a matter of moments. The large number of runs required by EFEA techniques would be rendered trivial, and such techniques may enter common usage to automate any engineering design process that can be parameterised and simulated.

None of the above implies, though, that there is no place for EFEA in present-day mechanical engineering. Across high-tech industry, there exist small, highly-optimised, weight-critical or stress-critical components, for which incremental improvements carry large financial rewards. Examples incorporating bolted joints or other fasteners include supercar or race car chassis structures, and aircraft wing mounts. In these cases, computation times of days or even weeks may be easily justified by the promise of new optimum and perhaps non-obvious design concepts, capable of being generated by algorithms such as DE. Moreover, there may be other cases in which an approximately optimal design solution may be 'good enough'. The results presented in Section 3.5.2 demonstrate that, although AbaqusJADE takes time to find the true optimum, perfectly acceptable engineering solutions may be generated in just a few tens of generations at minimal computational expense.

4.2 Multi-Objective Optimisation

The objective case considered the maximum deflection of the Bloodhound jet fuel tank baffle only: a somewhat artificial optimisation exercise as most industrial bolted joint optimisation problems are multi-objective.

In particular, care must be taken when designing bolted interfaces to ensure that all relevant failure modes are protected against with sufficient Reserve Factors (RFs), concurrently with optimising for other criteria, such as the maximum deflection or stress in the bolted components.

Bolted joints in homogeneous materials may fail in shear/tension of the bolt, thread shear, pull-through, shear-out and bearing failure. In composite materials, a variety of other failure modes arise, increasing the complexity of the problem. The RF for each failure mode is a function of the bolt forces, bolt and parent material properties, and the local geometry around the bolt hole. In many areas of industry, bolting RFs are computed using ad-hoc spreadsheets and computer scripts for want of dedicated tools. It would relatively simple and extremely helpful to build a bolt RF calculator into a future version of the optimiser presented in this thesis.

Supposing such a calculator was built in, the next challenge would be to use it effectively in the optimisation process. In a simple case, an engineer may wish to define a minimum acceptable RF for each bolt in a structure. This could be handled in DE by allocating a penalty cost to trial solutions which violate the minimum RF value, with higher exceedances attracting higher costs.

In a more subtle case, though, an engineer may wish to maximise the minimum RF across all bolts in the structure, concurrently with one or more other objectives. It is unlikely that every objective function will be optimised by the same bolting locations. The number of bolts used may itself be an objective function to minimise. In this example, Price *et al.* (6 pp. 243-255) give two available approaches.

The first of these is to define a weighted objective function before the analysis begins. In Equation 4.1, w_k represents the weight allocated to the kth objective. The values of w_k must be well-chosen by an expert to guarantee an optimal solution, which detracts from the simplicity and ease-of-use of the algorithm.

$$f(\boldsymbol{x}) = \sum_{k=1}^{n} w_k f_k(\boldsymbol{x}), \;\; n \geq 2$$

Equation 4.1

The second (and generally preferred) approach is to exploit the notion of Pareto-optimality. A Pareto-optimum solution is defined as "a solution around which there is no way of improving any objective without worsening at least one other objective" (35). Price *et al.* (6 pp. 250-255) demonstrate how DE can be modified to yield a group of Pareto-optimal solutions, from which the most appropriate may be selected after the optimisation ends. Allowing *a posteriori* solution selection by an engineer would be particular attractive for EFEA as it would facilitate selection based on manufacturability: a parameter that would otherwise be very difficult to build into the software logic.

4.3 EFEA: A Software Product

Having overcome the hurdles described above, the next logical step for EFEA is to develop a coherent and commercially viable software package that is attractive to industry.

A great deal of emphasis has been placed over the course of this paper on the necessity for simplicity. It would therefore be helpful to remove the programming requirement of EFEA completely by developing a Graphical User Interface (GUI).

Accessibility of EFEA would also be greatly improved if the software was made solver-agnostic, working with Ansys, HyperWorks, LS-Dyna, Nastran etc., as well as Abaqus. This would remove the requirement for engineers to purchase and/or learn an entirely new FEA package for optimisation work.

Throughout this paper, it has been assumed that FEA cost function landscapes are necessarily discrete, as FEA uses a system of nodes. However, it would also be possible to use the adaptive remeshing utilities provided by modern FEA packages to mesh each trial solution generated by DE separately. This would allow for optimisation to take place in the continuous domain, potentially enhancing the optimality of the generated solutions by facilitating detailed exploration of local minima.

Another way of expressing bolted joint location optimisation problems in the continuous domain would be to exploit 'mesh independent fasteners'. Mesh independent fasteners are Extended Finite Element Method (XFEM) entities pioneered by Abaqus. Using such entities, connectors may be attached at any point on an FE surface, not just at nodes. With regards to improving fastener modelling more generally, it may be desirable to devise a mechanism capable of considering bolt pre-tension, which would have had a significant impact on the membrane stiffness of the Bloodhound baffle.

By avoiding the need for remeshing, mesh independent fasteners would be more computationally efficient, but are specific to bolted joint problems. As per the motivation presented in Section 1.1, EFEA software should not be restricted to optimising bolted joint locations, but should instead be expanded to solve structural design optimisation problems more generally.

One particularly strong candidate problem for optimisation using an evolutionary approach is composite layup design. In high-end applications, engineers typically seek to optimise the orientations and stacking order of individual anisotropic plies in composite laminate materials. Through this process, it is possible to create composites that are lightweight, but also very strong in the specific loading directions envisaged. Due to the complexity of the analytical tools available to compute global material properties for a laminate material, the generally accepted industrial method for optimising complex composite structures is through iterative FEA.

This is, of course, a slow and complex process which has led to cases of high-profile bad publicity for composites in general. One such case is that of the Airbus A350XWB aircraft, which has suffered project delays and is overweight due to the difficulty in devising optimum composite designs (36).

Hypothetically, it would be possible to parameterise composite design into vector form, making it amenable to analysis using DE. The vector components could represent the total number of plies, the directionality of each ply, and any other variables of interest to the specific problem at hand. Cost functions could be readily generated from modern FEA software, with most packages now containing sophisticated tools for composite modelling. This approach could greatly accelerate the time-to-market for predominantly composite high-end products, such as luxury vehicles and aircraft.

Coming full circle, it may also be possible to use DE to optimise performance-critical aerodynamic components like aerofoils, as Rechenberg did using the very first documented EA (3). This could be done by translating the evolutionary FEA concept into the related field of CFD. CFD analyses tend to require more CPU time than FEA analyses due to the nonlinearity of the underlying Navier-Stokes equations; therefore, very significant improvements would be required in processing speed and/or algorithmic efficiency before this technique becomes viable for complex parts.

5 CONCLUSION

In Biology, evolution by Natural Selection is known to be an extraordinarily effective optimiser. Its only shortcoming is that it takes an awfully long time to generate optimum designs.

Regrettably, one is forced to draw largely the same conclusions about EAs in engineering. Although this study has proved the viability of applying an EA to solve a non-trivial bolted joint design problem in FEA, long runtimes are the main barrier to the industrial uptake of this technique on all but the smallest linear static models. This remains true despite the significant effort placed upon ascertaining the best possible evolutionary algorithm in Section 2. It is hoped that this work may be of value to the optimisation community at the University of Sheffield and beyond, as limited data benchmarking the most recently-published DE variants exists.

Nevertheless, there is a growing trend in the engineering world to use computers to automate, rather than merely aid, the design process. CAutoD is gaining traction through products such as ATOM and OptiStruct. Even the most passionate Luddite would not dispute that engineering components of the future will be designed largely by computers. The only real question is this: which techniques and algorithms will the computers use to optimise their designs?

It may be the case that cleverer and more efficient algorithms exist to tackle certain specific problems. It may even be the case that more efficient algorithms for bolted joint location optimisation will be realised in the future. However, it is worth repeating that the main strength of EAs such as DE is their simplicity. They require almost no *a priori* knowledge of the specific problem to be optimised. Moreover, it has been reasoned that advances in computer hardware in the next 10-20 years will give rise to the processing speed that will make EFEA practical. Therefore, it is the postulate of this thesis that, when the CAutoD epoch arrives, Evolutionary Finite Element Analysis will be at its very heart.

REFERENCES

1. **Green, A.** *Andy Green's Diary*. [Online] 28 December 2013. [Cited: 15 April 2014.] http://www.bloodhoundssc.com/blog/andy-green%E2%80%99s-diary-%E2%80%93-december-2013.

2. **Altair Hyperworks**. *Altair OptiStruct*. [Online] [Cited: 10 April 2014.] http://www.hyperworks.fr/(S(2kuacj45xtn15aubfjrqzqrz))/Product,19,OptiStruct.aspx.

3. **Rechenberg, I.** *Cybernetic Solution Path of an Experimental Problem*. Piscataway: IEEE Press, 1965, D.B. Fogel (Ed.), Evolutionary Computation, pp. 301-309.

4. **Rechenberg, I.** *Evolutionsstrategie*. Stuttgart: Frommann-Holzboog, 1973.

5. **Storn, R and Price, K V.** *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Technical Report TR-95-012, ICSI, 1995.

6. **Price, K, Storn, R and Lampinen, J.** *Differential Evolution – a Practical Approach to Global Optimization*. Heidelberg: Springer, 2005.

7. **Nelder, J and Mead, R.** *A Simplex Method for Function Minimization*. 1965, Computer Journal, Vol. 7, pp. 308-313.

8. **Kirkpatrick, S, Gelatt, C and Vecchi, M.** *Optimization by Simulated Annealing*. 1983, Science, Vol. 220, pp. 671-680.

9. **Kennedy, J and Eberhart, R.** *Particle Swarm Optimization*. 1995, Proceedings of the 1995 IEEE International Conference on Neural Networks. Vol. 4, pp. 1942-1948.

10. **Brest, J, et al.** *Self-adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems*. December 2006, IEEE Transactions on Evolutionary Computation, Vol. 10, pp. 646-657.

11. **Li, Q, Steven, G, Querin, O and Xie, Y.** *Stress Based Optimization of Torsional Shafts using an Evolutionary Procedure*. June 2001, International Journal of Solids and Structures, Vol. 38, pp. 5661-5677.

12. **Hull, P, Tinker, M and Dozier, G.** *Evolutionary Optimization of a Geometrically Refined Truss*. April 2006, Structural and Multidisciplinary Optimization, Vol. 31, pp. 311-319.

13. **Chen, T and Cheng, Y.** *Data-mining Assisted Structural Optimization using the Evolutionary Algorithm and Neural Network.* March 2010, Engineering Optimization, Vol. 42, pp. 205-222.

14. **Su, R, Gui, L and Fan, Z.** *Multi-objective Optimization for Bus Body with Strength and Rollover Safety Constraints based on Surrogate Models*. September 2011, Structural and Multidisciplinary Optimization, Vol. 44, pp. 431-441.

15. **Moore, G.** *Cramming More Components onto Integrated Circuits*. January 1998, Proceedings of the IEEE, Vol. 86, pp. 82-85.

16. **Luo, C, Song, L, Li, J and Feng, Z.** *Multiobjective Optimization Approach to Multidisciplinary Design of a Three-dimensional Transonic Compressor Blade*. June 2009. Proceedings of the ASME Turbo Expo. Vol. 7, pp. 599-608.

17. **Vu, V.** *Minimum Weight Design for Toroidal Pressure Vessels using Differential Evolution and Particle Swarm Optimization*. September 2010, Structural and Multidisciplinary Optimization, Vol. 42, pp. 351-359.

18. **Qin, A, Huang, V and Suganthan, P.** *Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization*. 2009, IEEE Transactions on Evolutionary Computation, Vol. 13, pp. 398-417.

19. **Zhang, J and Sanderson, A.** *JADE: Adaptive Differential Evolution with Optional External Archive*. 2009, IEEE Transactions on Evolutionary Computation, Vol. 13, pp. 945-958.

20. **Brest, J, et al.** *An Analysis of the Control Parameters' Adaptation in DE*. [book auth.] U Chakraborty. Advances in Differential Evolution. Berlin: Springer, 2008, pp. 89-110.

21. **Storn, R.** *Differential Evolution Research – Trends and Open Questions*. [book auth.] U Chakraborty. Advances in Differential Evolution. Berlin: Springer, 2008, pp. 1-31.

22. **Storn, R.** *Differential Evolution (DE) for Continuous Function Optimization (an Algorithm by Kenneth Price and Rainer Storn)*. [Online] [Cited: 11 04 2014.] http://www1.icsi.berkeley.edu/~storn/code.html.

23. **Gämperle, R, Müller, S and Koumoutsakos, P.** *A Parameter Study for Differential Evolution.* [book auth.] N Grmela. Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation. WSEAS Press, 2002, pp. 293-298.

24. **Zaharie, D.** *Critical Values for the Control Parameters of Differential Evolution Algorithms.* 2002, R Matousek and P Osmera (Ed.), Brno: Proceedings of MENDEL 2002, 8th International Conference on Soft Computing, pp. 62-67.

25. **De Jong, K.** *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* (PhD Thesis). University of Michigan, 1975.

26. **Brest, J and Maucec, M.** *Self-adaptive Differential Evolution Algorithm using Population Size Reduction and Three Strategies*. November 2011, Soft Computing, Vol. 15, pp. 2157-2174.

27. **Teo, J.** *Exploring Dynamic Self-adaptive Populations in Differential Evolution.* June 2006, Soft Computing, Vol. 10, pp. 673-686.

28. **Noman, N and Iba, H.** *Accelerating Differential Evolution using an Adaptive Local Search.* February 2008, IEEE Transactions on Evolutionary Computation, Vol. 12, pp. 107-125.

29. **Qin, A and Suganthan, P.** *Self-adaptive Differential Evolution Algorithm for Numerical Optimization.* September 2005, IEEE Congress on Evolutionary Computation. Vol. 2, pp. 1785-1791.

30. **Lampinen, J and Zelinka, I.** *Mechanical Engineering Design Optimization by Differential Evolution.* [book auth.] D Corne, M Dorigo and F Glover. New Ideas in Optimization. London: McGraw-Hill, 1999.

31. **Calvert, J and Farrar, R.** *An Engineering Data Book*. Southampton: Calvert Technical Press, 1998.

32. **Huth, H.** *Influence of Fastener Flexibility on the Prediction of Load Transfer and Fatigue Life for Multiple-row Joints.* Charleston: ASTM Special Technical Publication, 1986. Fatigue in Mechanically Fastened Composite and Metallic Joints, pp. 221-250.

33. **Advanced Composites Group.** *ACG Motor Racing Database Mechanical Data Collation*. September 2011.

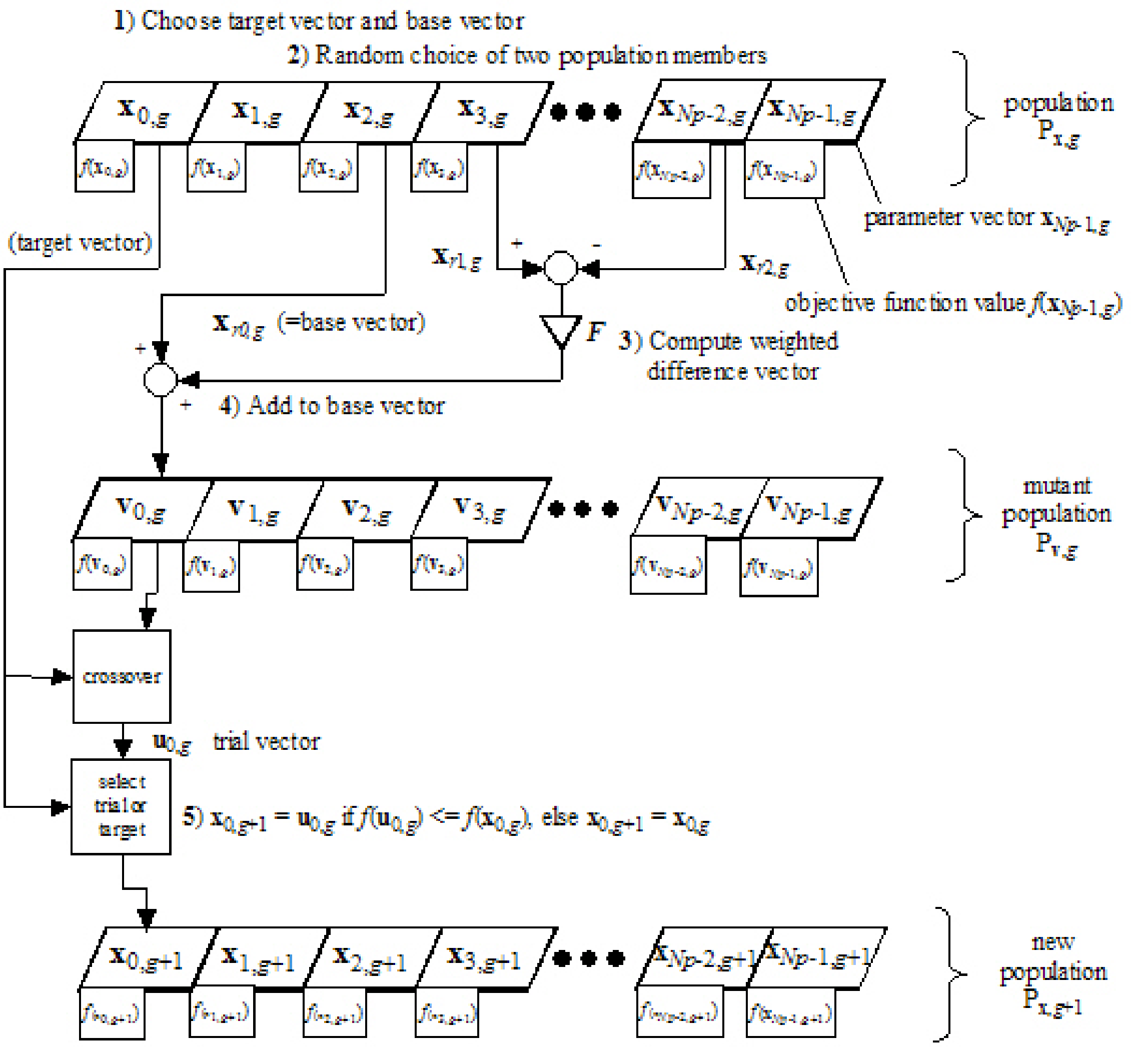
34. **Young, W and Budynas, R.** *Roark's Formulas for Stress and Strain (7th Edition).* New York: McGraw-Hill, 2002.

35. **Palli, N, Azarm, S, McCluskey, P and Sundarajan, R.** *An Interactive E-Inequality Constraint Method for Multiple Objectives Decision Making.* December 1998, Journal of Mechanical Design, Vol. 120, pp. 678-686.

36. **Wall, R.** *Weight Issues Remain For A350-900*. Aviation Week. [Online] 21 June 2011. [Cited: 1 May 2014.] http://aviationweek.com/awin/production-weight-issues-remain-a350-900

APPENDIX 1: THE CLASSIC DE ALGORITHM

Flowchart (Reproduced from (6 p. 43))



1) Choose target vector and base vector
2) Random choice of two population members
$x_{0,g}$ $x_{1,g}$ $x_{2,g}$ $x_{3,g}$ $x_{Np-2,g}$ $x_{Np-1,g}$
population $P_{x,g}$
parameter vector $x_{Np-1,g}$
objective function value $f(x_{Np-1,g})$
(target vector)
$x_{r1,g}$
$x_{r2,g}$
$x_{r0,g}$ (=base vector)
F
3) Compute weighted difference vector
4) Add to base vector
$v_{0,g}$ $v_{1,g}$ $v_{2,g}$ $v_{3,g}$ $v_{Np-2,g}$ $v_{Np-1,g}$
mutant population $P_{v,g}$
crossover
$u_{0,g}$ trial vector
select trial or target
5) $x_{0,g+1} = u_{0,g}$ if $f(u_{0,g}) <= f(x_{0,g})$, else $x_{0,g+1} = x_{0,g}$
$x_{0,g+1}$ $x_{1,g+1}$ $x_{2,g+1}$ $x_{3,g+1}$ $x_{Np-2,g+1}$ $x_{Np-1,g+1}$
new population $P_{x,g+1}$

C-style Pseudocode

```
//  Np = population size, D = problem dimensionality
//  f = mutation scaling factor, cr = crossover ratio
//  x = population array, u = trial population array
//  initialise random x and evaluate 'cost(v)' for each v in x.

//  then...

do
{
    //  generate a trial population
    for (i=0; i<Np; i++)
    {
        //  ro!=r1!=r2!=i
        do ro=floor(rand(0,1)*Np); while (r0==i);
        do r1=floor(rand(0,1)*Np); while (r1==r0 or r1==i);
        do r2=floor(rand(0,1)*Np); while (r2==r1 or r2==r0 or r2==i);
        jrand=floor(D*rand(0,1));

        //  generate a trial vector
        for (j=0; j<D; j++)
        {
            if (rand(0,1)<=cr or j==jrand)
            {
                u[i][j] = x[r0][j] + f*(x[r1][j]-x[r2][j]);
            }
            else
            {
                u[i][j] = x[i][j];
            }
        }
    }

    //  select the next generation
    for (i=0; i<Np; i++)
    {
        if (cost(u[i]) <= cost(x[i])) x[i]=u[i];
    }
} while (termination criterion not met);
```

Description of DE/X/Y/Z Strategy Taxonomy

DE strategies are conventionally named in accordance with a standard DE/X/Y/Z taxonomy. Common examples are DE/rand/1/bin (otherwise known as 'classic' DE), DE/best/1/bin, DE/best/2/bin, DE/current-to-best/1/bin and DE/rand/1/either-or. This list is far from exhaustive. Under this notation:

- The first term after DE refers to the method of selecting the base vector for mutation operations. 'rand' refers to *random*. 'best' always uses the best-so-far vector by cost value as the base vector. 'current-to-best' uses a point interpolated between the parent vector and the best-so-far vector. The latter two methods are generally understood to be 'greedy', giving rise to more rapid convergence at the expense of an increased misconvergence risk.
- The second term denotes the number of vector differences used to generate each mutant. '1' and '2' are the most common values.
- The final term refers to the recombination method, which is typically based on a *bin*omial distribution. Most DE strategies in literature use binomial crossover, although others may use the either-or method (performing either a mutation or recombination method at random), and some do not use it at all. In the latter case, strategies are normally denoted DE/X/Y.

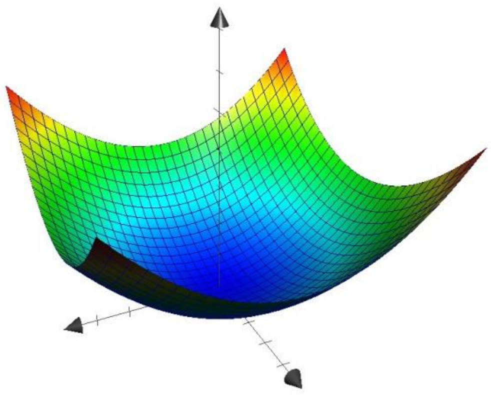
APPENDIX 2: BENCHMARK FUNCTIONS

Mathematical definitions and visualisations for $D = 2$.

f_1: Sphere

$$f(\boldsymbol{x}) = \sum_{j=0}^{D-1} x_j^2$$

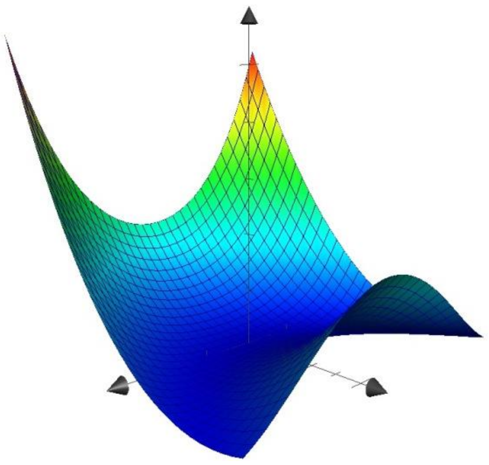
$$-100 \leq x_j \leq 100, \qquad j = 0,1,\dots,D-1$$



f_2: Rosenbrock

$$f(\boldsymbol{x}) = \sum_{j=0}^{D-2} \left(100 \cdot \left(x_{j+1}^2 - x_j^2\right)^2 + \left(x_j - 1\right)^2\right)$$

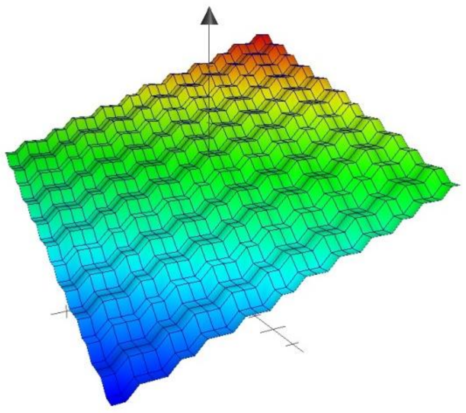
$$-30 \leq x_j \leq 30, \qquad j = 0,1,\dots,D-1$$



f_3: Step

$$f(\boldsymbol{x}) = 6D + \sum_{j=0}^{D-1} floor(x_j)$$

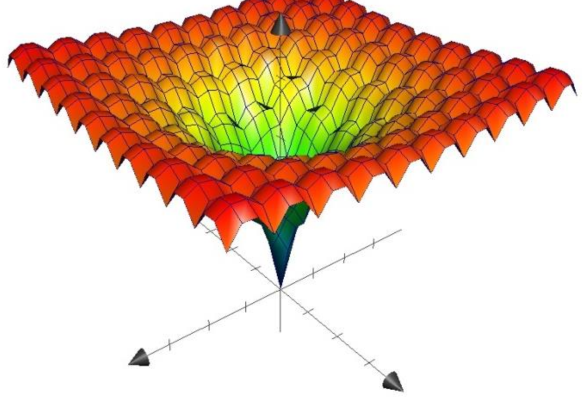
$$-5.12 \leq x_j \leq 5.12, \qquad j = 0,1,\dots,D-1$$



f_4: Ackley

$$f(\boldsymbol{x}) = -20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{D}\sum_{j=0}^{D-1} x_j^2}\right) - \exp(\frac{1}{D}\sum_{j=0}^{D-1} \cos(2\pi \cdot x_j)) + \ 20 + \ e$$

$$-30 \leq x_j \leq 30, \qquad j = 0,1,\dots,D-1$$

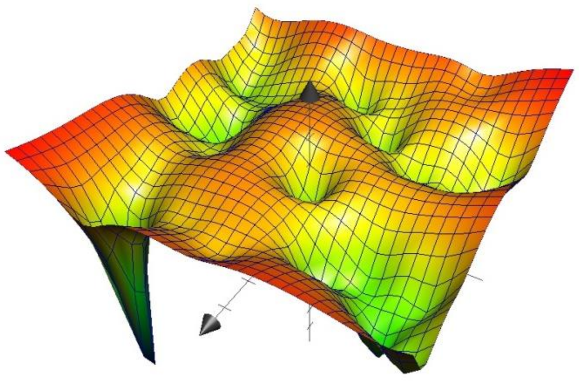


f_5: Shekel's Foxholes

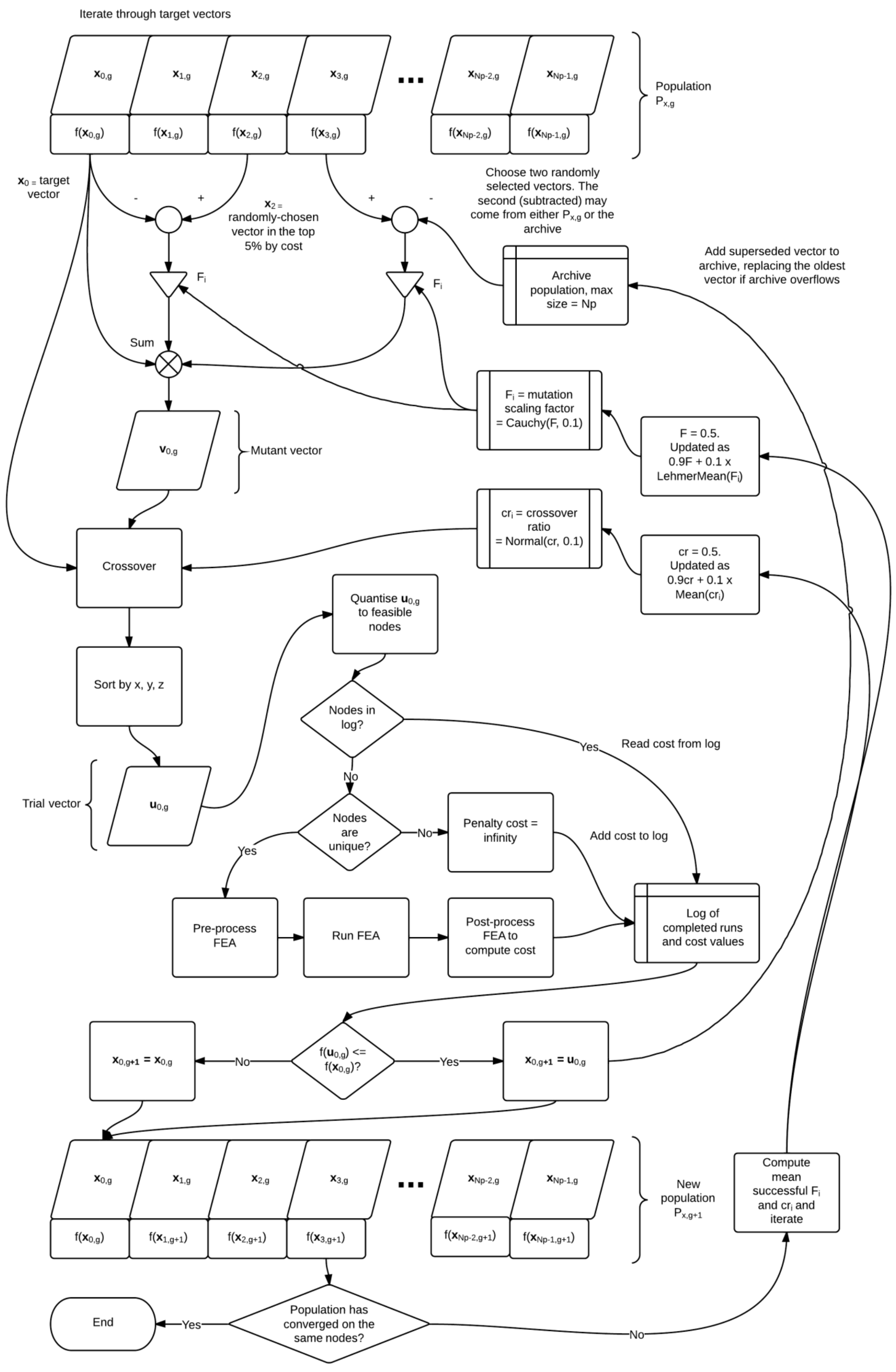
$$f(\boldsymbol{x}) = -\sum_{i=0}^{29} \frac{1}{\sum_{j=0}^{D-1}\left(x_j - a_{i,j}\right)^2 + c_i}$$

$$-0 \leq x_j \leq 10, \qquad j = 0,1,\dots,D-1$$

a and c are predefined matrices (see Price *et al.* (6 pp. 527-528))



Appendix 3: Flowchart for 'AbaqusJADE' DE Algorithm



Iterate through target vectors
$\mathbf{x}_{0,g}$ $\mathbf{x}_{1,g}$ $\mathbf{x}_{2,g}$ $\mathbf{x}_{3,g}$ ... $\mathbf{x}_{Np-2,g}$ $\mathbf{x}_{Np-1,g}$
$f(\mathbf{x}_{0,g})$ $f(\mathbf{x}_{1,g})$ $f(\mathbf{x}_{2,g})$ $f(\mathbf{x}_{3,g})$ $f(\mathbf{x}_{Np-2,g})$ $f(\mathbf{x}_{Np-1,g})$
Population $P_{x,g}$
$\mathbf{x}_0$ = target vector
$\mathbf{x}_2$ = randomly-chosen vector in the top 5% by cost
Choose two randomly selected vectors. The second (subtracted) may come from either $P_{x,g}$ or the archive
Add superseded vector to archive, replacing the oldest vector if archive overflows
Archive population, max size = Np
F_i
Sum
F_i = mutation scaling factor = Cauchy(F, 0.1)
F = 0.5. Updated as 0.9F + 0.1 x LehmerMean(F_i)
$\mathbf{v}_{0,g}$
Mutant vector
cr_i = crossover ratio = Normal(cr, 0.1)
cr = 0.5. Updated as 0.9cr + 0.1 x Mean(cr_i)
Crossover
Quantise $\mathbf{u}_{0,g}$ to feasible nodes
Sort by x, y, z
Nodes in log?
Yes
Read cost from log
No
Trial vector
$\mathbf{u}_{0,g}$
Nodes are unique?
No
Penalty cost = infinity
Add cost to log
Yes
Pre-process FEA
Run FEA
Post-process FEA to compute cost
Log of completed runs and cost values
$\mathbf{x}_{0,g+1} = \mathbf{x}_{0,g}$
No
$f(\mathbf{u}_{0,g}) <= f(\mathbf{x}_{0,g})$?
Yes
$\mathbf{x}_{0,g+1} = \mathbf{u}_{0,g}$
$\mathbf{x}_{0,g}$ $\mathbf{x}_{1,g}$ $\mathbf{x}_{2,g}$ $\mathbf{x}_{3,g}$ ... $\mathbf{x}_{Np-2,g}$ $\mathbf{x}_{Np-1,g}$
$f(\mathbf{x}_{0,g})$ $f(\mathbf{x}_{1,g+1})$ $f(\mathbf{x}_{2,g+1})$ $f(\mathbf{x}_{3,g+1})$ $f(\mathbf{x}_{Np-2,g+1})$ $f(\mathbf{x}_{Np-1,g+1})$
New population $P_{x,g+1}$
Compute mean successful F_i and cr_i and iterate
End
Yes
Population has converged on the same nodes?
No