Imperial College London

Imperial College London
Department of Aeronautics

MEng Individual Project

Investigating Efficient Algorithms for the Probabilistic Optimisation of Composite Panels

Author
Marij Tariq Qureshi

Supervisor
Prof. M.H. Ferri Aliabadi

Second Marker
Dr Zahra Sharif Khodaei

Submitted in partial fulfilment of the requirements for the degree of
Master of Engineering

June 2020

Abstract

This report investigates the potential benefits of probabilistic optimisation methods for the design of composite stiffened panels, along with a detailed comparison of algorithms for this purpose. Performance variation in aircraft structural design due to uncertainties has been a widely reported problem, which is addressed in this study through tailored design methods for increased robustness and reliability. Structural simulations were performed on a mono-omega-stringer stiffened panel and the novel use of dual-layer surrogate modelling achieved a factor of 700 speed-up in optimisation for only an additional 1% approximation error compared to standard methods. Unconstrained, constrained and multi-objective Pareto front design studies were carried out for deterministic, robust, reliability-based, combined robust and reliability formulations. These concluded the benefit of hybrid methods for producing more robust and similarly reliable designs compared to deterministic solutions. Gradient, direct, evolutionary, genetic algorithm and combined algorithms were rigorously compared for effective implementation of these design methods. For simpler problems such as deterministic design, direct methods were found to dominate in both speed and accuracy. However, as problem complexity increased, exploratory methods were increasingly found to be superior for locating global optima. The recommendation for future research is therefore, to utilise hybrid uncertainty-based optimisation with a multi-island genetic algorithm to efficiently find lighter, stronger and more consistently performing designs for aircraft structures.

Acknowledgements

Firstly, I would like to express gratitude to my supervisor Prof. M.H. Ferri Aliabadi for his invaluable guidance throughout all stages of this project. His vast experience in the field and positivity fostered great ambition and pushed me to achieve far more than I had initially imagined.

Secondly, I would like to thank Kwangkyu Alex Yoo, who acted as a brilliant mentor throughout the project and continued to offer useful advice and insights into further academic research.

Finally, I would like to thank Dr Zahra Sharif Khodaei for her valuable advice at key points in the project and the rest of the Imperial College staff who have made this achievement possible over four years.

1 Algorithm Comparison

1.1 Selection

1.1.1 Categorisation

There are a vast number of optimisation algorithms that have been developed over the years. These can be broadly broken down in a number of ways, such as derivative vs. derivative free or direct vs. evolutionary. For ease of understanding this report breaks them down further to more clearly highlight differences in approach.

Gradient-based methods - These use information about derivatives of objective function to determine optimised search directions. Many methods exist for this: simple steepest descent, conjugate gradient, sequential quadratic programming (SQP) etc. At each step, the optimal search direction calculated using first order Taylor approximation (gradient method) or 2nd order (newton method) - whilst more accurate, a drawback of the Newton method is that an expensive hessian needs to be calculated. SQP instead breaks down a non-linear problem into set of quadratic sub-problems (these are convex so very easy to solve). Overall, gradient methods tend to require a very small number of iterations and are extremely fast. However the use for gradient requires the objective function to be twice continuously differentiable to be accurate. Gradient algorithms terminate with Karush–Kuhn–Tucker(KKT) conditions, a combination of necessary conditions for optimality. This means that they can easily get stuck in local minima that satisfy this condition.

Direct Methods - These are derivative-free methods that search a few points within a local region, take steps in the best-found direction and gradually reduce size of local searched region for convergence. There are two common algorithms. The Hooke-Jeeves method perturbs each design variable one at a time until an improvement is found. If no improvement found, the size of perturbation decreases. Orthogonality of independent design variables ensures actions cannot "un-do" previous actions, however there is missed potential for faster improvement in non-orthogonal directions. The simplex (or Nealder-Mead) method offers possibility to search in all directions by defining a multi-dimensional shape around the design point. Vertices on this shape are evaluated and the simplex is reflected in the best direction or shrunk if there is no improvement. Direct methods are intuitive and extremely simple to implement. However they are very sensitive to topology of design space, with poor performance on discontinuous problems. Additionally, convergence can be very slow for more complex problems and may not even converge to a stationary point.

Evolutionary Methods - Whereas gradient and direct methods calculate the same path every time they are re-run, these incorporate randomness in the search. The philosophy is rather than take few expensive steps, take many quick steps far out to quickly search the design space and slowly evolve better solutions from previous ones. A number of creative approaches achieve this: Evol takes a simple approach applying a normal distribution to each design variable and slowly reducing the mean and standard deviation around the most successful point. Adaptive DOE instead runs an Optimal Latin Hypercube of the whole design space and gradually reduces the DOE bounds in promising regions. Adaptive simulated annealing (ASA) takes a different approach and simulates many particles moving around the design space and slowly reduces their energy to trap them in minima. Particle swarm optimisation (PSO) instead gives particles intelligence to communicate with each other about the best regions and directions found by the population. These methods achieve far better coverage of design space - therefore much more likely to find multiple local minima and the global minimum. Additionally they require no derivative information and can deal with any problem, whether discontinuous or even integer or binary. However a huge number of iterations are required, relying on parallelisation to be fully effective. Also, whilst an algorithm may find the general area of the global optimum, it may take many more iterations to accurately locate the exact minimum point.

Genetic Algorithms - Also inspired by nature, in these methods a population of design points is gradually evolved through successive generations, with the DNA of each individual holding the design variables. They are characterised by three main stages at each generation. Selection: from the population a certain proportion of best performing individuals selected to reproduce. Crossover: genes are crossed over between pairs of points, giving children a mix of design variable values. Mutation: additional randomness is incorporated by applying a probability to mutate a design variable in an individual, to another random value. Many variations on these methods have been explored - varying population distributions in the design space, elitism of selection, variable mutation or more involved methods such as the Teaching-Learning Based Method. As with evolutionary algorithms, these can be applied to any problem. They are excellent at finding global optima - however achieving quicker convergence than evolutionary methods. Finally, they are simple to implement and configure for reasonably robust results on a range of problems. For good results and efficiency though, there is huge variability in performance for different configurations. As such, they require a lot of effort to tune appropriately for complex problems, and for very simple problems, can be extremely inefficient.

Combined Methods - These methods control multiple algorithms of different classes to combine benefits and usually exist in proprietary software. Two common algorithms exist. Multifunction Optimisation System Tool (MOST) approximates a solution to the problem using an SQP gradient method (discontinuous variables treated as continuous). Then each variable taken away in turn and reduced problem solved to seek improvements (analogous to direct methods but with dimensionality). Pointer on the other hand controls a selection of different sub-algorithms of different classes and runs them all on the problem. It dynamically assesses which are failing and reconfigures or terminates them. As expected with a combination of algorithms, these are good for robust application to many problems, however are rarely the most efficient methods.

1.1.2 Decision Process

In order to understand and compare the different classes of algorithm, 1 from each class was to be selected for detailed analysis. Initially, a comparison of all 16 algorithms performed on simple structural test case. This was a plate with elliptical cut-out, for which 2x design variables (width and height of cut-out) were optimised for maximum buckling load (details can be found in appendix). Simulations were conducted in Abaqus under same boundary conditions as mono-string model, and iterations to converge to single global optimum recorded.

This selection was also done based on prevalence in literature, software and industry applications so that conclusions could be widely applicable. Suitability to continuous and constrained problems was considered, as well as CPU performance to aid with analysis workflow. A table summarising findings used to make decisions is recorded below.

Table 1.1: Algorithm decision matrix (prevalence scored from lowest-1 to highest-5). Green are selected, yellow are multi-objective methods

Class	Acronym	Iteration Benchmark	Well suited for			Prevalence	Notes
			Non-Linear	Constraints	Long-Running Simulations		
Gradient	MISQP	13	Yes			1	Poor global convergence
	NLPQLP	13	Yes		Yes	5	Poor global convergence
	MMFD	14	Yes	No	Yes	2	Excellent parallelisation
	LSGRG	15	Yes	Yes	Yes	5	
Direct	Downhill Simplex	20		No	Yes	4	
	Hooke-Jeeves	37		No	Yes	4	Excellent for linear problems
Exploratory	Evol	52	Yes		Yes	2	
	Particle Swarm	21			No	5	
	ASA	74			No	3	Excellent global convergence
	Adaptive DOE	54				1	Good for very large number of DVs
GA	NSGA2	116	Yes		No	5	
	AMGA	136	Yes	Yes	No	4	Good for many local optima
	NCGA	272			No	3	
	Multi-Island GA	167			No	4	High performance consistency
Combined	MOST	10		Yes	Yes	2	
	Pointer	34	Yes	Yes	Yes	4	

The following algorithms were selected (a detailed description of methods provided in later section to better understand results).

- **LSGRG** - A version of the commonly used reduced gradient methods optimised for high number of design variables. It is very widely used, being the default non-linear method for Microsoft Excel solver. NLPQLP was a good candidate, and Isight default method but known to converge only to local optima and not very good at dealing with constraints. Hence despite similar performance for all gradient methods, LSGRG was selected as the only gradient method that is well optimised for constraints
- **Simplex** - This had much better performance than Hooke-Jeeves, with half the number of iterations needed. Both are expected to perform poorly with constraints but this is a fundamental problem of the class of methods. Hooke-Jeeves is widely used but typically better performing for linear rather than non-linear problems, so simplex was selected.
- **PSO** - This algorithm had by far best performance in class, almost 2.5 times fewer iterations than the second best. It is not well suited for long-running simulations due to large CPU usage but by far the most prevalent and used in an increasing number of studies. Evol was a good candidate, likely to be well suited in performance to a mono-string problem but in the interests of useful comparison for wide applicability, PSO was selected.

- **MIGA** - This had slightly worse performance than NSGA2/AMGA. Additionally none of the GA methods are suited for long-running simulations - a problem of highly parallelisable exploratory methods that are CPU intense. However, a differentiating factor was that the main focus of this comparative study is scalarised multi-objective optimisation. The other 3 methods were developed mainly for different purpose - pareto front optimisation. Hence MIGA chosen as comparative GA for other classes.
- **Pointer** - This algorithm had worse performance for the simple problem but is reported to be better for more complex problems. It is extremely well suited to the target problem and much more prevalent in industry application. Additionally it benefits from requiring little configuration. Finally, running a combination of most other classes serves as a good example of the potentials of combining methods, so this algorithm was selected.

1.2 Performance Overview

1.2.1 Testing Methodology

In the context of probabilistic optimisation. algorithms were tested on the mono-stringer problem. The constrained formulation (with an upper limit on mass) was chosen as it is the most applicable to practical applications and would ensure no bias against constraint-driven approaches. Algorithms were compared across 3 levels of optimisation complexity: 2 objective functions (deterministic method), 3 objective functions (robust method), and 4 objective functions (hybrid 2:1 method). Results were compared across 3 target masses for each method and averaged for a rigorous comparison of method performance for a range of designer needs and objective function topologies. For ease of understanding this breakdown, gridpoints are defined as a combination of method and target mass as shown below.

Optimisations were implemented in Isight for each algorithm, looping through gridpoints. Each gridpoint represents a unique optimisation problem with a unique global optimum but there is possibility for unfair performance if the start point happens to be close to the optimum. Hence each gridpoint was put inside a third loop, running 10x with different starting points for LSGRG/Simplex (determined by Optimal Latin Hypercube for maximum variety), and 10x random seeds for other methods subject to randomness. Averages were taken to determine performance. An initial study of exploratory methods indicated software would keep running for extremely long time unless stopped. Convergence was assessed for iteration caps in factors of 10, showing limited improvement between 10k and 100k and almost no improvement between 100k + 1M iterations. Finally all algorithms were tuned for optimal performance on gridpoint 2 (deterministic) beforehand for an unbiased comparison of each algorithm at its best.

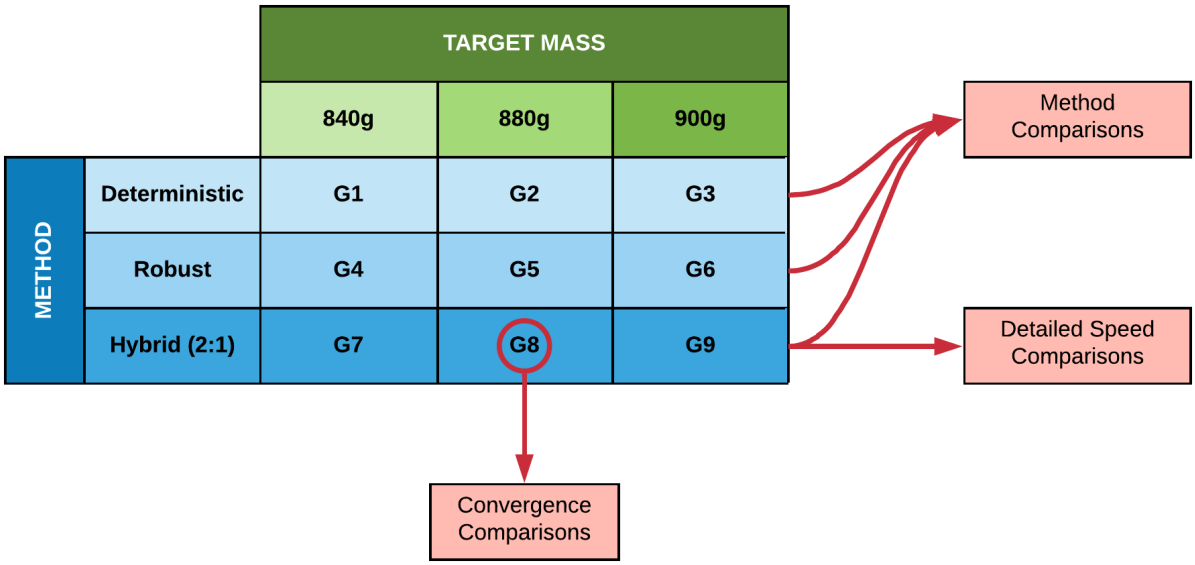


Figure 1.1: Explanation of Gridpoints

1.2.2 Speed Comparison

Performance

Using different combinations of gridpoints as shown Figure 4.1, a range of different comparisons were carried out. For those that compare against global optimum, this was found by running MIGA for 1 million iterations on each gridpoint. The following figure demonstrates performance for the hybrid cases.

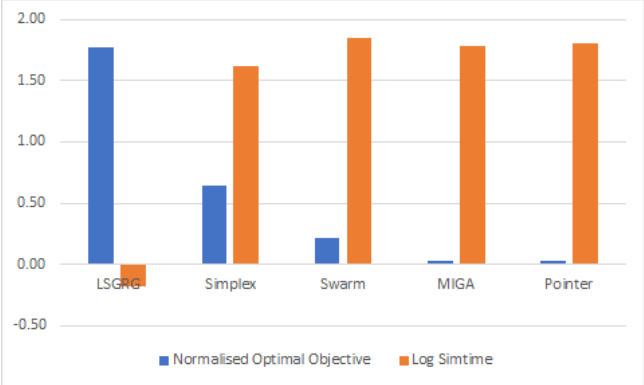


Figure 1.2: Comparison of objective and optimization time (sim-time) by method

Comparing optimal objective values in Figure 4.1, it is clear that both gradient + direct methods found local minima (note the gradient solution was much worse, likely just the closest minimum to the starting point). PSO performed better but still missed the global optimum. MIGA and Pointer on the other hand achieved extremely low objectives and in fact both found the global optimum. This trend is as expected from literature - contrasting a focus on exploration (last 3 algorithms) vs exploitation (first 2 algorithms).

Considering simulation time however, LSGRG was the fastest by a huge margin (factor of 60 from closest method). This makes a drawback of exploratory methods clear - better solutions come at a much higher computational cost. Simplex was surprisingly slow considering it only takes a single path - it is therefore worth investigating the algorithms journey to explain why this happens.

These results suggest a possible way to utilise gradient methods' speed is by running inside a DOE loop - if there exist in the order of 60 local optima, spaced apart, then this could become more efficient. Nevertheless genetic algorithms/combined methods still appear to be the most robust methods to ensure a global optimum is found.

Method

Next, results are compared by method, to see how different classes perform at different complexities. For a clearer visual comparison in Figure 4.2, performance metrics were ranked instead, with rank 1 = best objective/simulation time and rank 5 = worst.

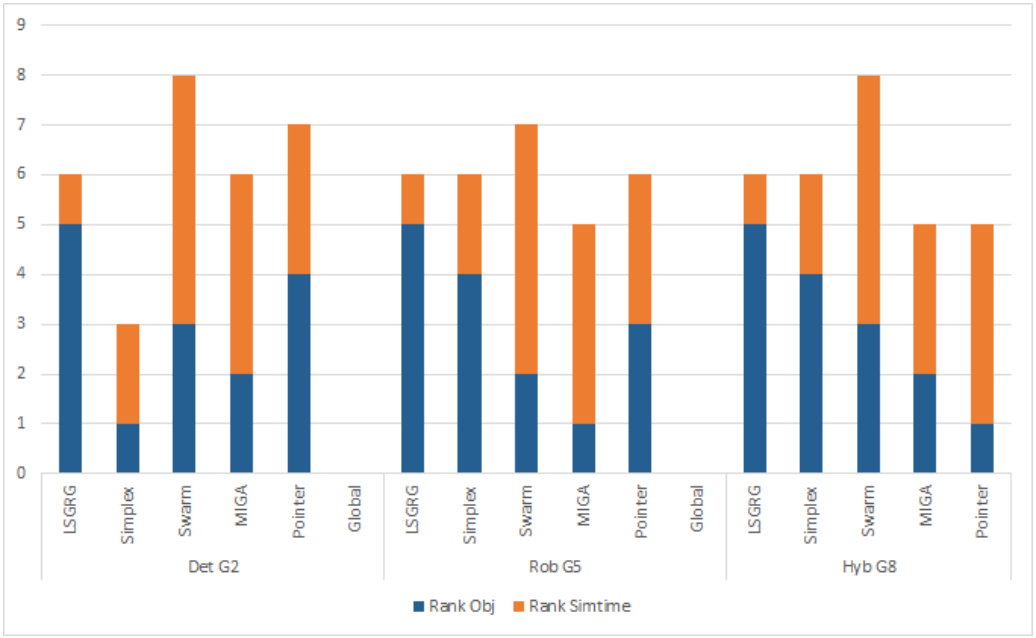


Figure 1.3: Comparison of algorithm rank by method

Firstly, LSGRG was both the fastest and achieved the worst objective across all methods, whilst the exploratory, GA and combined methods were consistently the slowest 3. This indicates gradient method performance is unaffected by complexity - likely since it sees only a small region of design space regardless of the problem.

For the simplest deterministic problem, the direct method was by far the best, successfully finding the global optimum and in a time only slower than the gradient method. Exploratory methods took much longer and produced inferior solutions. Adding complexity through introduction of uncertainty-based objectives in the robust case, there is a shift to PSO/MIGA finding better solutions than the gradient and direct methods. Surprisingly, whilst pointer was quicker than these, it achieved a poorer solution. Finally for the hybrid case, the results were as detailed previously, with a clear trend towards the GA class dominating traditional methods in location of global optima. Here pointer was the best performing but slower than the GA - overall equal in performance considering speed and accuracy.

From this we can conclude that direct methods are excellent for very simple problems but quickly become slow and inaccurate as complexity increases. Genetic algorithms and combined methods become dominant as dimensionality increases (with Pointer likely leveraging the internal GA most here). Interestingly, the particle swarm method was consistently a poor in-between choice with the worst of both worlds - a long run-time with many function calls but also not as good at finding optima as a GA.

Function Calls

Next, a breakdown of simulation time by number of function calls and calculation time per step is compared for the different algorithms. Figure 4.3 presents results for the deterministic case which most clearly demonstrates the differences.

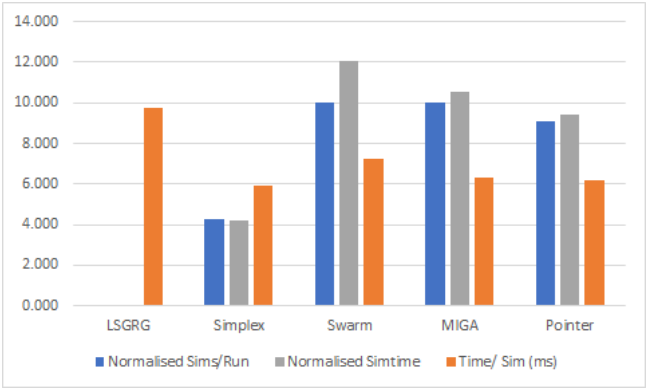


Figure 1.4: Breakdown of function calls by method

For the gradient method, firstly note that iterations/runtime data are not omitted, just negligibly small compared to the others. However their ratio is more telling with the highest time per iteration by about 50% over the 2nd highest algorithm. This can be explained as each step is highly optimised using gradient information and calculation of gradients/hessians is expensive. From this information it is likely significantly more expensive if run multiple times from different points in the earlier proposed DOE method.

Simplex achieved the lowest time per iteration, which is characteristic of direct methods. These are simplest to evaluate at each step, with either binary decisions or simple averages needed. This perhaps suggests good scaling with more simplexes run from different points.

With PSO, previously it was seen that it had consistently highest runtime. Now it is clear that this is not due to the high number of function calls, as these are comparable to GA/Pointer. Instead there is a significantly higher calculation time per iteration due to many exchanges needed at each iteration between all particles. Additionally this is not favoured for parallelisation as information from the entire population is needed at each step before even a single particle can be calculated.

Whilst pointer had lower runtimes, we can now see that time/iteration is not much different. The reason for this is most runs needed all 10000 runs to converge, but some runs happened to start close to GA so used other methods to converge faster. Therefore on average, fewer iterations were needed than MIGA but since smaller runs used gradient methods, these were relatively more expensive. Hence Pointer demonstrates comparable efficiency but potential for much faster runs depending on starting conditions.

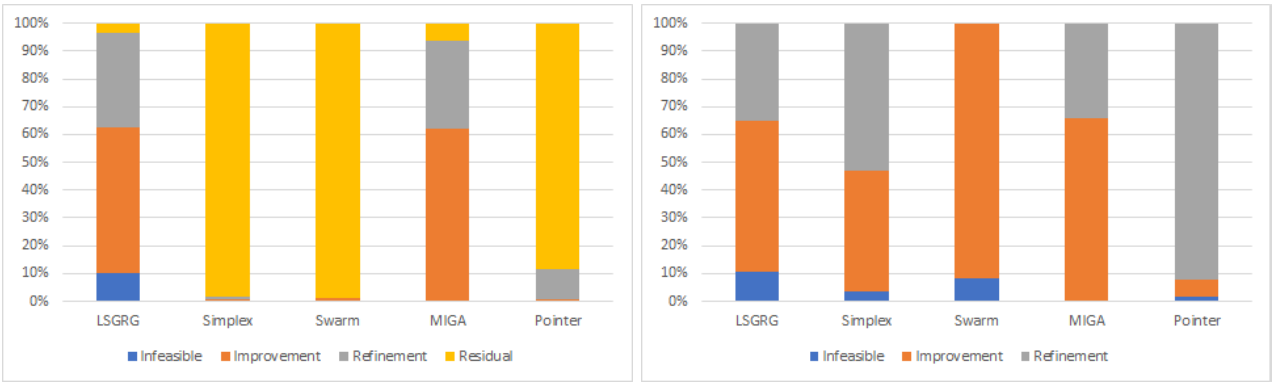
1.2.3 Convergence Comparison

Now we know high level performance comparison, it is worth investigating how efficiently each algorithm converges to optimum and how solutions compare. Clearly averaging journeys for individual runs does not make sense here, so results are shown for the gridpoint 8 case starting from the infeasible base design.

Convergence Shape

Firstly to clearly quantify convergence of algorithms, iterations are split into a number of phases: An infeasible phase, where the algorithm tries to get into the feasible region, an improvement phase to find where the local/global optimum is in design space (defined by a solution within 1% of final optimum). A refinement phase follows this, where the algorithm finds the exact bottom of local/global optimum and finally a residual phase, where extra iterations are run by the program with no improvement before it is satisfied and terminates.

Fig 4.4a below shows the split of total iterations, Figure 4.4b the split up to optimum point (excluding residual stage), and Figure 4.5 shows how the best solution found so far changes up to optimum point.



(a) Including residual phase

(b) Excluding residual phase

Figure 1.5: Iteration split for each method

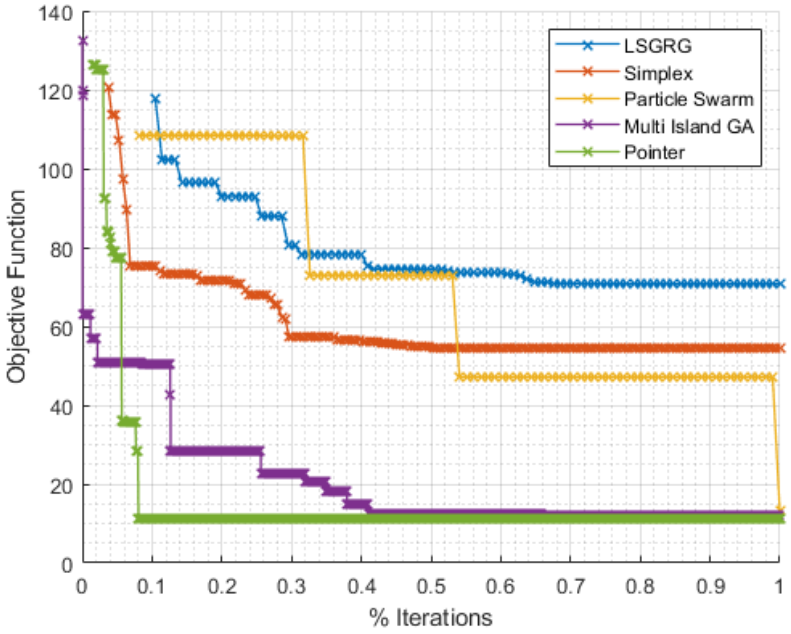


Figure 1.6: Iteration histories for different algorithms

Firstly, from Figure 4.4 we can see that LSGRG and MIGA had very small residual phases, wasting under 5% of iterations after the optimum was found. However this was for different reasons: LSGRG has a convergence criterion that stops it once it reaches a stationary point, but MIGA achieved this by continuing to find slightly better solutions as long as it continued running. The other methods struggle to define convergence so just keep trying points until the artificial cap was reached.

Figure 4.5 reveals that all algorithms are quick to escape the infeasible region. LSGRG spends highest proportion of time doing so but in actual iterations it is very few steps. Interestingly, MIGA found the feasible solution instantly but this is somewhat expected with randomised starting positions. Consider the Improvement/Refinement phase split, which represents how quickly algorithm decided to stick with potential optimum area rather than continue searching. One would expect gradient and direct methods to have lower proportion of improvement phase as they are designed to stick with the first minimum they find; again a relatively small number of iterations could explain why this is not the case. Exploratory and GA methods have much higher proportion of improvement as they are designed to continually improve the solution, not by digging deeper into local optimum but by exploring to find better optima. Finally the most surprising case is pointer, which found its optimum extremely fast, within 10% of total iterations. However it wasted a large proportion of these with only an additional 1% improvement to the result.

Considering the shape of the improvement curve in figure 4.5, LSGRG and Simplex gradually improve the solution with diminishing returns as expected - simplex with particularly strong improvements early on. Furthermore the difference between exploratory and GA methods becomes clear here: MIGA found a number of improved local optima along way and made small but quick improvements, whereas Particle Swarm found only 4, with large jumps and flat regions where algorithm searched with no success. Importantly, this means that PSO cannot be stopped early - stopping at 30% completion would have resulted in a worse solution than even LSGRG and would still be worse than Simplex after 50%

These results suggest the need to understand why it is so difficult to terminate convergence for constrained problems. An idea to investigate here would be whether algorithms return to infeasible region after initially leaving it. Finally with striking results, Pointer can perform best by a long margin if convergence and termination conditions can be properly specified.

Optimal Geometry

Finally, since these results are all placed in the context of stiffened panel design, it is interesting to see the geometry configurations of solutions. Here again we look at the hybrid G8 problem (optimising for buckling load, robustness and reliability considering uncertainties, with maximum mass 880g). The results, visualised below, show a few different designs - clear evidence of convergence to different local optima.

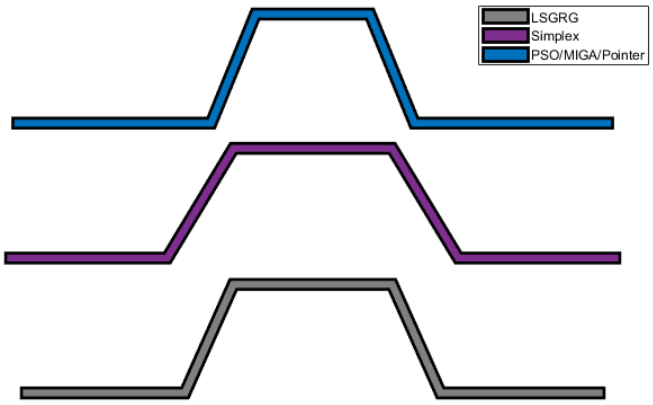


Figure 1.7: Optimal geometry for each algorithm

Figure 4.6 shows that LSGRG converged to solution with wide flange, resulting in the worst performance of the algorithms. Simplex found a solution with surprisingly large value of B at this mass considering what we now understand about the effect of slope on performance variability. This does explain the poor performance of the optimum point though. However, all the Exploratory, GA and Combined methods all converged to same solution, likely the global optimum, which has a significantly smaller middle section width but larger feet. Surprisingly this configuration also has a slightly larger height - unexpected for typical deterministic design but highlights that the probabilistic trade-off can produce numerous closely performing but different geometries.

It is interesting to note that the NSGA2 multi-objective Pareto optimisation used in section X found a solution similar in design to global opt. but slightly worse performing for this mass. This is expected to be quite optimal since it is a GA but shows that single-objective optimisation will still dominate for any particularly defined problem.

1.3 Detailed Algorithm Explanations

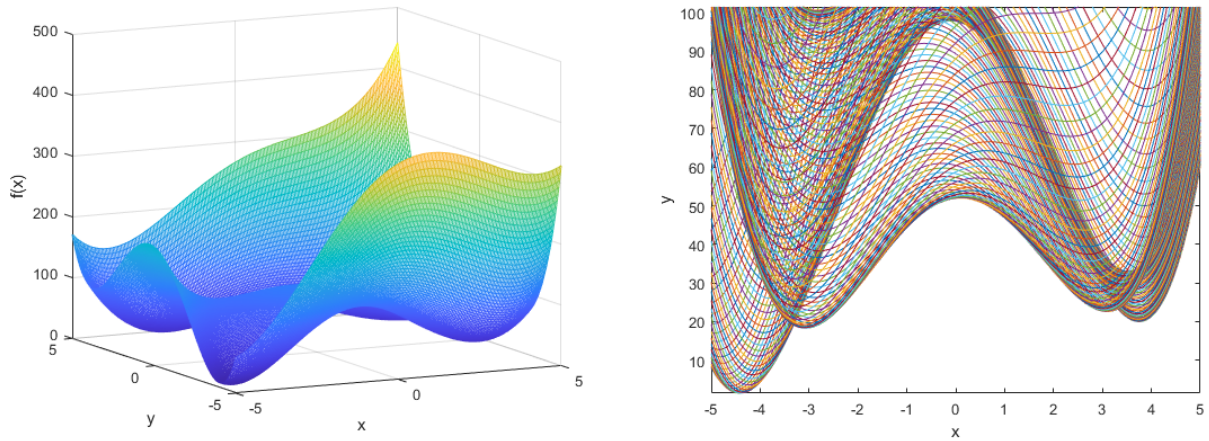
The performance of different classes of algorithms has been presented and discussed. General convergence properties indicate the most obvious features of each algorithm. However the best way to rigorously assess algorithms is to understand fundamental principles behind them and chart how they search the design space. The following sections seek to explain their behaviour from a more mathematical/analytical basis and highlight how differences in formulation lead to differences in results.

1.3.1 Simplified Problem Description

For the purpose of understanding the journey of each algorithm, there are several issues with using only the real problem. The primary problem is that it is 4-dimensional in design variables. One can take a slice of local region, keeping some variables constant for understanding an optimal design point, but for comparison of algorithms across the entire design space in all variables, it becomes impossible to visualise. A second important problem is the high non-linearity; many crowded local optima makes understanding decisions for the path of an algorithm difficult.

Therefore, a simpler test function is used to aid comparisons, with an aim to select a function which mimics real structural problems in a 3D space. For validity to the real problem, the objective function had to be continuous, well scaled and non-linear. It also should be non-convex, with multiple local optima and a global optimum not closest to the starting point to ensure a non-trivial solution. The Himmelblau function was found to be a popular test case in literature, however it has identical minima so not quite suitable here. Therefore the objective function was adjusted to closely mimic the form of a real scalarised objective function + give unique minima, as shown below. Finally, the problem chosen to be unconstrained to allow informative comparison between unconstrained/constrained performance.

$$f(x,y) = \frac{2}{5}\left(x^2 + y - 11\right)^2 + \frac{3}{4}\left(x + y^2 - 7\right)^2 - \frac{1}{1000}\left(x^2 + y^2\right)^3 + 25 \tag{1.1}$$



(a) 3D surface view

(b) Side view

Figure 1.8: Modified Himmelblau function

1.3.2 Large-scale Generalised Reduced Gradient Method (LSGRG)

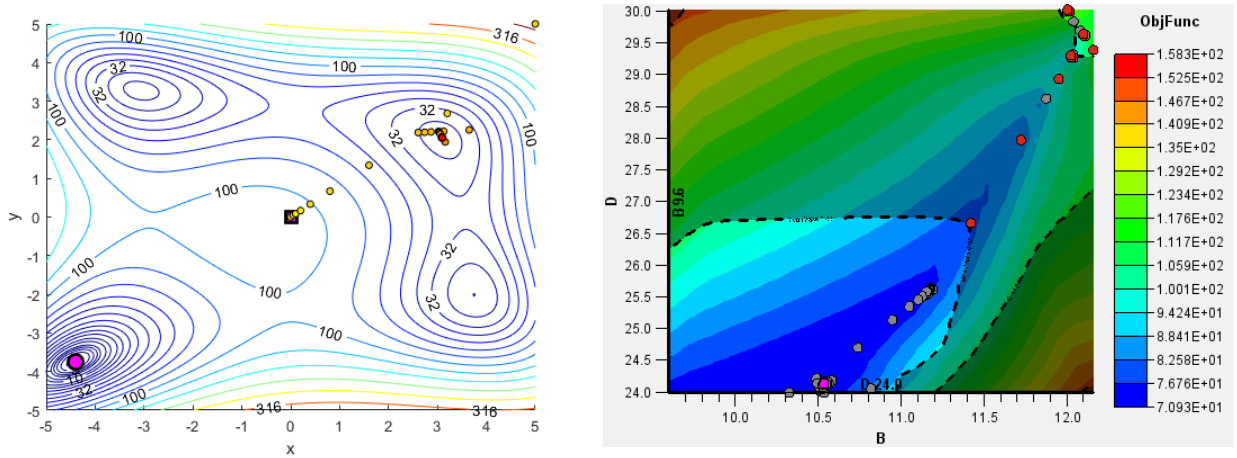
The Large-scale Generalised Reduced Gradient method is fundamentally based on the conjugate gradient method; however a number of advanced features have been added over the years to produce a fast, widely applicable algorithm, each giving rise to parts of the name:

- Reduced gradient - For each equality constraint, we can solve for one design variable in terms of others (a dependent variable), reducing dimensionality of problem. The algorithm tailors the search direction to ensure that active constraints remain precisely active for some small move in that direction.
- Generalised - This builds on an earlier reduced gradient method that solved for equality constraints only. It converts each inequality constraint to equality using slack variables, as in equation 4.2. However this no longer reduces dimensionality for inequalities as slack variables become dependent:

$$g_j(x) < 0 \Rightarrow g_j(x) + s = 0 \tag{1.2}$$

- Large-scale - Unlike typical gradient methods, this algorithm is highly optimised for parallelisation, separating potential search direction gradient calculations. It also uses sparse matrices stored in less memory rather than dense matrices for gradients / Hessians to speed up computation when they are large due to a large number of DVs.

Finally the algorithm operates in two phases: Phase 1 - minimises a constraint penalty plus a fraction of the objective to obtain a feasible solution. Phase 2 then minimises the objective.



(a) Simple unconstrained problem

(b) Real constrained problem

Figure 1.9: LSGRG algorithm iteration steps

Figures 4.8 show paths of algorithm implemented on the simple (a) and real (b) problems. From both plots it can be seen that the algorithm is generally travelling down path of steepest descent throughout. Additionally it doesn't fully minimise on the line search, in order to prevent partially un-doing any previous step - this is a clear example of conjugacy of search direction gradients. Next, we see an increase in step size in proportion to magnitude of gradient of search direction (i.e. larger steps where contour lines are closer), which speeds up design improvement where it can. Notably for constrained regions, the algorithm does not move to closest feasible region in top corner of (b), but instead comes out in line with the steepest gradient descent, so has improved the objective considerably before even becoming feasible. This explains LSGRG's fast convergence to closest local optimum, and exceptional performance dealing with constraints.

1.3.3 Downhill Simplex Method

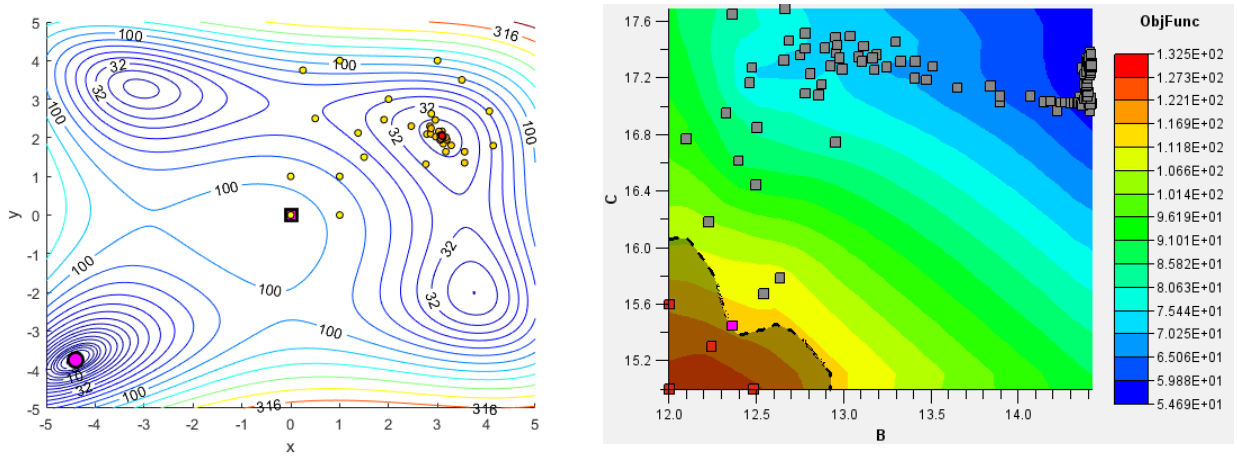
The simplex method (also called the Nealder-Mead method, not to be confused with the Simplex method for linear programming), works with a comparatively simple set of instructions:

For a problem with n design variables, a simplex is defined as a regular polytope (polyhedron extended to more dimensions) with n+1 points, given by:

$$\Delta_c^n = \left\{ (t_1\ ,\ \dots\ ,\ t_n) \in R^n \mid \sum_{i=1}^{n} t_i \leq 1 \; and \; t_i \geq 0 \; for \; all \; i \right\} \tag{1.3}$$

At each step, the following calculations are performed: Ordering, where vertices are evaluated and ordered in terms of performance. Next the centroid c of the best side is calculated (this is opposite the worst vertex x_j and calculated as: $c := \frac{1}{n} \sum_{j \neq h} x_j$). Finally a transformation maps the old simplex to a new one to improve the worst point.

There are 3 possible transformations at every stage, characterised by a reflection through the best side, replacing the worst point with $x_{new} = c + a(c - x_{old})$. Most of the time a regular reflection occurs, where $a = 1$. If results in a direction are particularly promising, an expansion occurs with a typically raised to 2. If results are converging however, a contraction happens instead with $a = 1/2$ typically, reducing the size of the simplex.



(a) Simple unconstrained problem

(b) Real constrained problem

Figure 1.10: Simplex algorithm iteration steps

Figure 4.9 shows the performance of the simplex algorithm. The first thing to notice is that the algorithm is highly predictable - no spurious points are evaluated and the path can clearly be tracked. The results in the previous sections showed quick improvements early on. This can be seen in large initial steps quickly reducing objective function by large amounts. Additionally for the real problem, this means the algorithm escapes the infeasible region in just 5 steps. From the theory, the simplex algorithm is based on reflection and so it is expected to be prone to zig-zagging movements. This is indeed seen with the unconstrained optimiser circling the minimum very slowly, whilst the constrained optimiser does not take the direct steepest gradient path - it travels up to the north-west corner before very steep gradient forces it towards the local minimum.

Convergence in both cases is only to local minima. Incidentally, both problems start with the problem chasing an immediate downhill gradient towards North East. However from (a) it becomes clear that the algorithm

never even sees the landscape outside of this quadrant, which explains particularly poor performance for higher-dimensional spaces. Finally, there is very slow convergence once near minimum indicating contractions do not happen regularly or strongly enough.

1.3.4 Particle Swarm Method

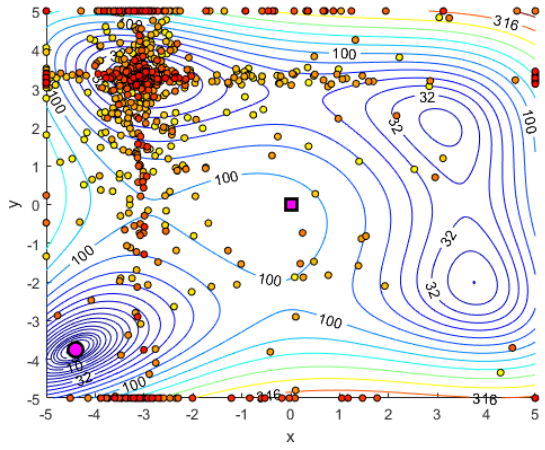
This method, inspired by nature, mimics the social behaviour of a group of animals foraging food. An initial population of design points (particles) is generated with randomly generated positions in the design space (states) and velocities. At each stage, the new position of a particle is given by its initial position plus velocity. This velocity is calculated by the following equation:

$$v_{ij}(t+1) = w\, v_{ij}(t) + r_1 c_1 \left[P_{ij}(t) - x_{ij}(t)\right] + r_2 c_2 \left[g_j(t) - x_{ij}(t)\right] \tag{1.4}$$

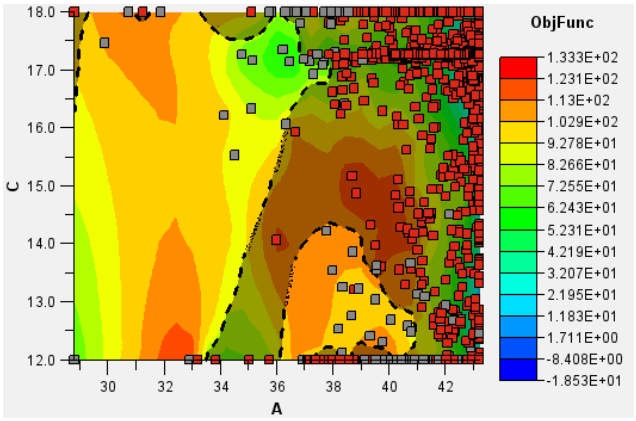
Here a cognitive component (2nd term) attracts a particle to the best position it has encountered so far, simulating independent thought. A social component is added, attracting each particle to the best design found so far by the entire population. This step is expensive as communication is needed between all the particles. Finally, an inertia term tries to keep particles moving in the same direction. This avoids particles repeatedly changing their mind and wiggling around a single area. Acceleration coefficients c1 and c2 are used to tune the relative effects of all these components, and are related by the following equation:

$$\chi = \frac{2\kappa}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}, \quad w = \chi, \quad c_1 = \chi\,\phi_1, \quad c_2 = \chi\,\phi_2 \tag{1.5}$$

For ϕ_1,ϕ_2 and k, Clerc + Kennedy suggested typical numbers 2.05, 2.05 and 1, which are commonly used.



(a) Simple unconstrained problem



(b) Real constrained problem

Figure 1.11: PSO algorithm iteration steps

From the results displayed above, a number of behaviour patterns can be identified. Firstly for the simple problem, this exploratory method encourages convergence to better minimum than closest, but still the global optimum was not found. This is exactly as was seen in the real problem results, with an optimal point performance much better than gradient/direct methods, but not global as achieved by GA/combined methods.

Communication can clearly be seen in action for the unconstrained case. At the start of the optimisation process, yellow circles are spread out around the initial point. As the optimiser ran, some particles found the NW minimum, after which clearly all other points travelled over to that position. The algorithm skirted around the global optimum but missed it - this shows that an inherently random method can fail sometimes, and the likelihood of finding global minima decreases as non-linearity increases and more local optima are available.

For the real constrained optimisation case in (b), we can see that the algorithm does come out of the infeasible region quickly, but does not explore a lot of the feasible space. In fact, most of the feasible region to the west is uncovered, missing other potential minima. Generally high velocities in the algorithm formulation makes accurate convergence to exact minimum hard - this explains more scattered points around the minima in (a) and jumps in solution quality with no convergence phase identified in section 3. With a large residual

phase, we would expect this to be concentrated deviations around the minimum that fail to actually improve it. However the illustration above confirms it was actually just continued wasted sampling of infeasible space - a reformulation of penalty functions could be suggested but the conclusion is that PSO generally does not handle constrained optimisation well.

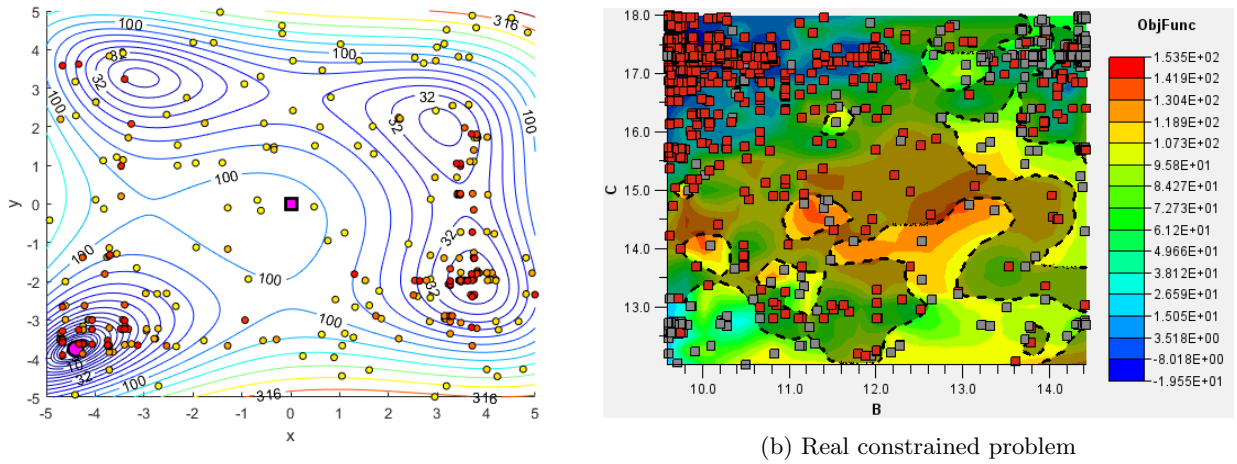
1.3.5 Genetic Algorithm

Again inspired by nature, genetic algorithms are widely used and reported in literature. Basic principles have already been mentioned earlier so more focus will be on what differentiates MIGA and its implementation in software. More detailed information can be found in [30]

In this algorithm, design variables are first encoded into a single binary string representing chromosome, each number being a gene. MIGA uses tournament selection, where best individuals are chosen not from the entire population but rather from a random sub-set. This allows for duplicates to better favour exceptional designs, however tuning the tournament size can reduce this - in the limit as tournament size goes to 1, this becomes regular selection.

The algorithm also implements elitism, where the best individuals are immune from alteration to the ensure the best result each generation never gets worse. At the crossover stage, this is carried out by crossing each chromosome at 2 points (so it is split into 3 sections). Genes between these points are swapped, resulting in 2 new individuals to maintain population size each generation. Finally a constant probability of swapping each binary digit is applied to the whole chromosomes for mutation.

The biggest difference in a multi-island genetic algorithm is its management of the population. Rather than have one large population, it is split into several sub-populations called islands. Periodically individuals migrate in between islands, introducing variation. This encourages a range of converged optima, not just the whole population converging to the same value.



(a) Simple unconstrained problem

(b) Real constrained problem

Figure 1.12: MIGA algorithm iteration steps

For the simple problem in (a), initial samples (yellow) are very spread out and cover the entire design space - this highlights a known strength of genetic algorithms. What is particularly promising is that this is true even for the higher-dimensional complex real problem - note the scales on (b) span the full range of allowed values. Considering iterations later in the optimisation process, unlike particle swarm and indeed pointer, even the final samples are spread across all 4 minima. This indicates that the multi-island approach works well as intended, allowing for a more gradual improved design even at a late stage in optimisation process. Likely aided by the elitism approach, this explains the high improvement and low convergence and residual phases that represented a distinct advantage over the other methods.

MIGA was the only algorithm to find the global optimum for the unconstrained problem. In fact, even for the constrained real problem with the existence of multiple small islands of feasible solutions, it was able to sample at least one point in each, a significant improvement over PSO.

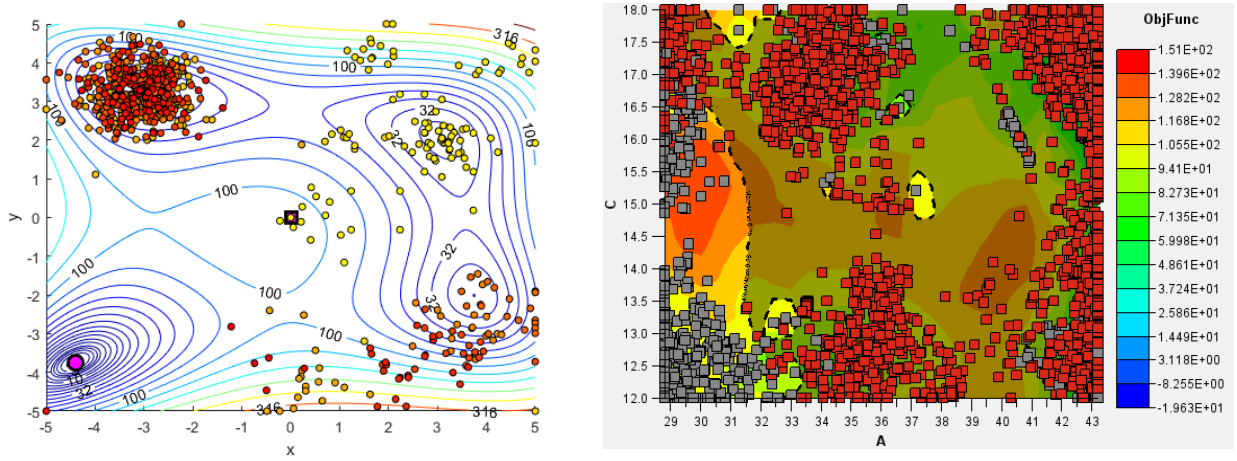
This evidence points to the outstanding exploratory capabilities of genetic algorithms, which are significantly more likely to find a global optimum than other classes. This makes them a highly robust optimisation method for range of problems. The key drawback here is inefficiency - as with PSO, a high number of points continue

to be sampled in infeasible region since there is no understanding of local gradients and how to quickly move to constraint boundaries.

1.3.6 Combined Method

The philosophy behind Pointer is that it is designed to be a black-box tool for non-experts to use with minimal configuration. This unfortunately results in limited available details on the specifics of implementation. That said it is known to consist of an evolutionary algorithm, the Simplex method, sequential quadratic programming (NLPQLP) and a linear solver. This selection of methods is designed to combat each other's weaknesses with certain design space topography features.

Pointer starts all algorithms on a given problem simultaneously and monitors them closely for progress. Based on this it terminates or adjusts parameter settings for the sub-algorithms in real time. It can therefore use beneficial information from one algorithm to improve another, e.g. location of a found minimum or gradient information.



(a) Simple unconstrained problem

(b) Real constrained problem

Figure 1.13: Pointer algorithm iteration steps

From the results in figures 4.12a and 4.12b, the clearest feature of this algorithm is convergence to clusters of design points. In the unconstrained case, it found the nearest minimum and quickly converged and explored around it. Pointer then found and explored the SE local optimum before finally discovering the NE minimum point. This shows a very intelligent exploration of design space with very few sampled points in non-promising regions.

Looking at the clusters in detail, they are relatively spread out and circular in shape. The spread indicates use of an exploratory method, whilst the circular shape suggests use of simplex near optima. However, objective function values indicate excellent resolution of local optima especially in the constrained problem, which can only indicate leveraging of the gradient method

Between the two problems there is a contrast in coverage however. In the simple case, the Pointer missed most of the design space and could not find the global optimum. This echoes results for the deterministic study - poor performance for simpler problems and can further be explained by a reliance on exploratory methods for high-level searching of the design space. Based on randomness, there is always a possibility for regions to be missed.

For the constrained problem however, figure (b) shows excellent coverage rivalling the Genetic algorithm. Pointer also identified all locally feasible regions, explaining comparable success in finding global optima as seen previously. This likely demonstrates the internal evolutionary algorithm at work, although this brings with it the side effect of a large number of infeasible points. Similar results were found in a comparative study [40], comparing pointer with a range of benchmark problems and their respective best algorithms. It found that Pointer was usually two times as expensive on average, so could not be used as a general optimisation solution.

1.4 Synthesis

The comparison of performance for these five algorithms indicates a clear effect of complexity on the best choice of algorithm class for a particular problem. Firstly gradient methods appear to be unaffected by problem complexity, running fast since they only see a small portion of the design space anyway. However in comparison to other methods they converge to much worse local optima. Similarly for the deterministic optimisation problem, the direct method achieves excellent results quickly, however once uncertainty-based objectives are added for the robust optimisation case, performance drops off. For the probabilistic methods, exploratory methods begin to dominate more, with genetic algorithms performing particularly well at finding global optima in the most complicated robust and reliability-based design case.

From detailed analysis of the performance of gradient and direct methods, these are shown to favour careful improvement of design in each step with fewer iterations. However, a key difference was that the gradient method had the most expensive calculations per step, processing gradient information, whilst Simplex evaluated iterations quickest. Nevertheless, LSGRG converged 60 times faster than the others and was best formulated to terminate quickly after this. By contrast, whilst Simplex quickly improved the design, convergence for a constrained problem proved to be very difficult.

The other three exploratory methods on the other hand were much better at exploring the design space, particularly for the constrained problem. Because of this, MIGA was the only algorithm to find a global optimum for both problems, supporting the consensus in literature that genetic algorithms are reliably superior methods as problem complexity increases. However the combined method, Pointer, converged much faster than any other method, in around 10% of the time it took MIGA. Analysing the optimisation journey it becomes clear that a combination of methods allows much more intelligent exploration of the design space, however at the cost of lower efficiency and difficulty dealing with highly-constrained design spaces.

For complex design space topographies typical of uncertainty-based optimisation methods involving constraints, direct, exploratory and combined methods are all shown to pose problems with termination conditions, particularly particle-swarm optimisation. Better methods need to be developed to specify convergence conditions carefully to avoid unnecessary sampling of infeasible points with no improvement.

2 Conclusions

In this study, deterministic and probabilistic optimisation methods have been compared for the geometry optimisation of a composite stiffened panel, and efficient optimisation algorithms explored for this purpose. Uncertainties in structural response due to variations in material properties and geometry have been simultaneously considered for a more realistic assessment of design performance. In line with common practice, expensive structural simulations were approximated using design of experiments techniques and surrogate modelling. However, for the purposes of running a large number of optimisation studies, a second layer of approximation was applied to capture the uncertainty responses of each design. This resulted in computational savings in the order of a million times faster with total accuracies maintained within 5% on average. This was aided by an improvement on the standard Monte-Carlo simulation, utilising an iterative convergence method and Sobol sampling to minimise required samples for an accurate uncertainty response distribution.

A deterministic optimisation approach was shown to produce designs with a significantly higher buckling load than the base design, however these had very poor robustness. Notably, when designs were constrained to a mass lower than 884g and mean buckling load greater than 49.5kN, a high correlation emerged between mean and reliable buckling load, resulting in comparable performance for deterministic and reliability-based design. Considering the simpler deterministic optimisation and an example unconstrained optimisation formulated to aid comparison, gradient and direct methods were found to be most efficient at optimising geometry. These ran considerably faster than evolutionary methods and in the case of the Simplex method, converged to a better local optimum solution. Therefore, for less complex problems, these methods have been recommended.

In the case of probabilistic optimisation, RDO and RBDO methods were found to optimise for superior robustness and reliability respectively. The benefits of these methods were exploited with a combined hybrid approach which produced stiffener designs that were comparably reliable but much more robust than deterministic solutions. The effect of modifying weightings given to standard deviation and minimum reliable load objectives was investigated. From this a clear advantage over previous methods was the ability for a designer to tune the problem formulation to output more robust or reliable designs based on requirements, with a predictable linear variation in response.

In the more complex hybrid design optimisation case, exploratory algorithms were found to be superior in locating global optimum solutions. Despite an increased runtime, Particle Swarm optimisation was found to locate much better solutions than either of the gradient or direct methods. The genetic algorithm improved on this and was found to be the best class of algorithm to reliably find global optimum solutions for higher complexity problems. Combined methods were shown to be more efficient, converging to similar designs in a tenth of the time, however they were more prone to missing global optima, particularly for highly-constrained problems.

Future Work

Based on the conclusions of this study, there are a number of possible avenues for future research. Furthering structural design for aircraft, the optimisation problem can next be expanded to a multi-stringer optimisation. Based on observations for single-stiffener design, it would be recommended to simply maximise flange width and minimise height from the outset to gain the largest improvement in buckling load resistance and halve the number of design variables per stringer. The limitation of this study on reliability-based design can be overcome by including more objective functions such as maximum deflection or material cost. Finally, to aid this a comparison of other reliability methods such as FORM/SORM could yield more efficient optimisations.

This report introduced the potential of dual-layer surrogate modelling, which was crucial for the high optimisation complexity of this study. In future studies, application of this methodology can be leveraged to enable combined robust and reliable optimisation with a more accurate non-linear model. Alternatively, more uncertainties could be simultaneously applied such as internal stresses due to variable loading conditions, or simultaneous optimisation of both geometry and lay-up could be explored for a more global design optimisation.

Finally, techniques to improve efficiency of optimisation algorithms could be further explored. The results of this study directly suggest an investigation into running gradient or direct methods inside a design of experiments loop to leverage their speed and improve exploratory capabilities. Finally, efficient methods for multi-objective Pareto front optimisation should be investigated, with a more focused comparison of tailored methods such as NSGA2, NCGA or AMGA. Optimisation is now a fundamental part of aircraft design and developing efficient methods will ultimately enable lighter, more robust and reliable design solutions for next-generation aircraft.

Bibliography

[1] Johan Andersson. *A survey of multiobjective optimization in engineering design.* PhD thesis, 2000.

[2] Satchi Venkataraman and Raphael Haftka. Optimization of composite panels- a review. pages 479–488. American Society for Composites, 1999.

[3] Fabrizio Sbaraglia, Hamed Farokhi, and Ferri M. H. Aliabadi. *Robust and Reliability-Based Design Optimization of a Composite Floor Beam.* PhD thesis, 2018.

[4] Mark W. Hilburger, W. Jefferson Stroud, Thomas A. Zang, Peiman Maghami, Michael J. Hemsch, Sharon L. Padula, James M. Luckring, and Sean P. Kenny. Needs and opportunities for uncertainty-based multidisciplinary design methods for aerospace vehicles. Technical report, Jul 1, 2002.

[5] Wen Yao, Xiaoqian Chen, Wencai Luo, Michel van Tooren, and Jian Guo. Review of uncertainty-based multidisciplinary design optimization methods for aerospace vehicles. *Progress in Aerospace Sciences*, 47(6):450–479, 2011.

[6] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.

[7] Wei Wang, S. Guo, Nan Chang, and Wei Yang. Optimum buckling design of composite stiffened panels using ant colony algorithm. *Composite Structures*, 92(3):712–719, 2010.

[8] G. Sun and Renjie Mao. Optimization of stiffened laminated-composite circular-cylindrical shells for buckling. *Composite Structures*, 23(1):53–60, 1993.

[9] G. I. Schuëller and H. A. Jensen. Computational methods in optimization considering uncertainties – an overview. *Computer Methods in Applied Mechanics and Engineering*, 198(1):2–13, 2008.

[10] R. Rikards, H. Abramovich, K. Kalnins, and J. Auzins. Surrogate modeling in design optimization of stiffened composite shells. *Composite Structures*, 73(2):244–251, 2006.

[11] Ricardo M. Paiva, Curran Crawford, and Afzal Suleman. Robust and reliability-based design optimization framework for wing design. *AIAA Journal*, 52(4):711–724, Apr 2014.

[12] M. Papadrakakis, N. D. Lagaros, and V. Plevris. Design optimization of steel structures considering uncertainties. *Engineering Structures*, 27(9):1408–1418, 2005.

[13] M. W. Long and J. D. Narciso. Probabilistic design methodology for composite aircraft structures. Technical report, Jun 1999.

[14] Peng Hao, Bo Wang, and Gang Li. Surrogate-based optimum design for stiffened shells with adaptive sampling. *AIAA Journal*, 50(11):2389–2407, Nov 2012.

[15] Peng Hao, Chen Liu, Xiaojie Yuan, Bo Wang, Gang Li, Tianyu Zhu, and Fei Niu. Buckling optimization of variable-stiffness composite panels based on flow field function. *Composite Structures*, 181(1):240–255, Dec 1, 2017.

[16] Carlos López, Carlos López, Omar Bacarreza, Omar Bacarreza, Aitor Baldomir, Aitor Baldomir, Santiago Hernández, Santiago Hernández, M. H. Ferri Aliabadi, and M. H. Ferri Aliabadi. Reliability-based design optimization of composite stiffened panels in post-buckling regime. *Structural and Multidisciplinary Optimization*, 55(3):1121–1141, Mar 2017.

[17] Sharon Padula, Clyde Gumbert, and Wu Li. Aerospace applications of optimization under uncertainty. *Optimization and Engineering*, 7(3):317–328, Sep 2006.

[18] S. Nikbakt, S. Kamarian, and M. Shakeri. A review on optimization of composite structures part i: Laminated composites. *Composite Structures*, 195(1):158–185, Jul 1, 2018.

[19] Guan-Chun Luh and Chun-Yi Lin. Optimal design of truss-structures using particle swarm optimization. *Computers and Structures*, 89(23):2221–2232, 2011.

[20] Arora and Jasbir Singh. *Introduction to Optimum Design.* Academic Press, US, 3 edition, Aug 12, 2011.

[21] Manoj Kumar Buragohain and Manoj Kumar. *Composite Structures.* CRC Press, London, 1 edition, Aug 15, 2017.

[22] Bo Liu, Slawomir Koziel, and Qingfu Zhang. A multi-fidelity surrogate-model-assisted evolutionary algorithm for computationally expensive optimization problems. *Journal of Computational Science*, 12(1):28–37, Jan 2016.

[23] Nikolaos D. Lagaros, Manolis Papadrakakis, and George Kokossalakis. Structural optimization using evolutionary algorithms. *Computers and Structures*, 80(7):571–589, 2002.

[24] Luciano Lamberti, Satchi Venkataraman, Raphael T. Haftka, and Theodore F. Johnson. Preliminary design optimization of stiffened panels using approximate analysis models. *International Journal for Numerical Methods in Engineering*, 57(10):1351–1380, Jul 14, 2003.

[25] Kim Cheol and Yoon In-Se. The stacking sequence optimization of stiffened laminated curved panels with different loading and stiffener spacing. *Journal of mechanical science and technology*, 20(10):1541–1547, 2006.

[26] J. Herencia, Paul Weaver, and Michael Friswell. Optimization of anisotropic composite panels with t-shaped stiffeners including transverse shear effects and out-of-plane loading. *Structural and Multidisciplinary Optimization*, 37(2):165–184, Dec 2008.

[27] Hamed Farokhi, Omar Bacarreza, and M. H. Ferri Aliabadi. Probabilistic optimisation of mono-stringer composite stiffened panels in post-buckling regime. *Structural and Multidisciplinary Optimization*, 1(1):1–23, May 15, 2020.

[28] Warren Hare, Julie Nutini, and Solomon Tesfamariam. A survey of non-gradient optimization methods in structural engineering. *Advances in Engineering Software*, 59(1):19–28, May 2013.

[29] Hossein Ghiasi, Damiano Pasini, and Larry Lessard. Optimum stacking sequence design of composite materials part i: Constant stiffness design. *Composite Structures*, 90(1):1–11, 2009.

[30] J. Díaz, M. Cid Montoya, and S. Hernández. Efficient methodologies for reliability-based design optimization of composite panels. *Advances in Engineering Software*, 93(1):9–21, Mar 2016.

[31] B. Gaspar, A. P. Teixeira, and C. Guedes Soares. Adaptive surrogate model with active refinement combining kriging and a trust region method. *Reliability Engineering and System Safety*, 165(1):277–291, Sep 2017.

[32] G. Chiandussi, M. Codegone, S. Ferrero, and F. E. Varesio. Comparison of multi-objective optimization methodologies for engineering applications. *Computers and Mathematics with Applications*, 63(5):912–942, Mar 2012.

[33] Jared L. Cohon and David H. Marks. A review and evaluation of multiobjective programing techniques. *Water Resources Research*, 11(2):208–220, Apr 1975.

[34] Omar Bacarreza, M. H. Aliabadi, and Alfonso Apicella. Robust design and optimization of composite stiffened panels in post-buckling. *Structural and Multidisciplinary Optimization*, 51(2):409–422, Feb 1, 2015.

[35] E. Barkanov, O. Ozoliņš, E. Eglītis, F. Almeida, M. C. Bowering, and G. Watson. Optimal design of composite lateral wing upper covers. part i: Linear buckling analysis. *Aerospace Science and Technology*, 38(1):1–8, Oct 2014.

[36] Venkata M. K. Akula. Multiscale reliability analysis of a composite stiffened panel. *Composite Structures*, 116(1):432–440, Sep 2014.

[37] H. Arda Deveci, Levent Aydin, and H. Seçil Artem. Buckling optimization of composite laminates using a hybrid algorithm under puck failure criterion constraint. *Journal of Reinforced Plastics and Composites*, 35(16):1233–1247, Aug 2016.

[38] Christos Kassapoglou. *Design and analysis of composite structures.* Wiley, Chichester, West Sussex, U.K, 2010.

[39] Brian G Falzon and M H Ferri Aliabadi. *Buckling and Postbuckling Structures II.* WORLD SCIENTIFIC (EUROPE), 2018.

[40] Alex Velden and Pat Koch. Isight design optimization methodologies. 2020.