CUED - Engineering Tripos Part IIB 2021-2022 **Module Coursework**

Module	**4M17**	Title of report	**Assignment 2: Optimisation Algorithm Performance Comparison Study**

UNDERGRADUATE STUDENTS ONLY	
Candidate number:	5585G

Assessment for this module is ☑ 100% / ☐ 25% coursework of which this assignment forms **50%**

MPHIL STUDENTS ONLY	
Candidate number:	

Date submitted: 18/12/2021

OTHER POSTGRADUATE STUDENTS	
Name:	
CRSid:	

Feedback to the student

☐ **See also comments in the text**

		Very good	**Good**	Needs imprvmt
CONTENT	**Completeness, quantity of content:** Has the report covered all aspects of the assignment? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
PRESENTATION	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Marker: Date:

1 Introduction

The aim of this report is to compare and contrast the performance of two stochastic optimisation methods on a difficult test case. I chose to compare Simulated Annealing and Evolution Strategies since they differ in the sense that one is based on a Markov chain approach, whilst the other is population based. The full algorithms (as well as the required *backend* for each) were built in python with extensive use of *numpy* for speed and *matplotlib* for visualisation. Various parameter configurations are considered for each algorithm in order to explore their effect on performance. Additionally, care is taken to reduce the effect of random sampling when measuring and discussing the quality of each method by varying the seed and averaging over multiple runs.

Throughout this report, the optimisation problem referred to is a minimisation problem and, as such, decreases in the objective function are considered an improvement. The test objective function in this report is the 6-dimensional Eggholder function - though for visualisation purposes the 2-dimensional Eggholder function will also be considered. The specific form of the N-dimensional Eggholder function is (for reference)

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} -(x_i + 47)\sin\sqrt{|x_{i+1} + 0.5x_i + 47|} - x_i \sin\sqrt{|x_i - x_{i+1} - 47|}\,, \quad -512 < x_i < 512 \tag{1}$$

This optimisation problem is hard as the function is littered with local optima, and the global optimum is on a constraint boundary. Despite this, it has many attractive features including a non-disjoint output space, no equality constraints and identically scaled input features.

2 Implementation Details

I will start by outlining some details of my specific implementation of these methods that I feel are pertinent to the overall discussion in this report, or that required some thinking beyond the lecture notes on my part. This includes explanation of some coding decisions as well any choices made regarding certain methods in the handout (e.g. choice of recombination methods).

2.1 Evaluating the Objective

Since the two chosen methods are not gradient based (and hence we do not know in which direction to step to decrease our objective), potential solutions are compared at each iteration by direct evaluation of the objective function at each location. Given that these algorithms often require upwards of ten thousand evaluations for a full run, it is very important that the objective is evaluated as efficiently as possible to avoid excessive run-time. A naïve approach to evaluating this objective is to iterate over each index in the control variable $\mathbf{x}$ (bar the last) in turn and accumulate the sum.

More careful consideration suggests that for long control variables a vectorised approach is favourable, particularly on a parallelised computer. This is easily achieved by noting that at each iteration we consider only a pair of adjacent values, x_i and x_{i+1}, and hence (using the 3-dimensional case as an example) by *rolling* our control variable

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \quad \text{rolled to} \quad \mathbf{x}_r = \begin{bmatrix} x_2 & x_3 & x_1 \end{bmatrix}^T$$

Then truncating

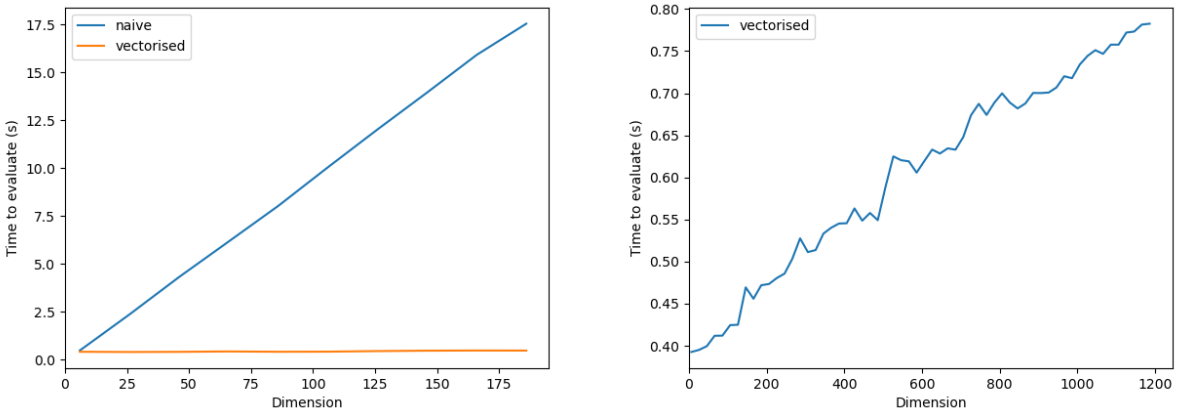
$$\mathbf{x}_t = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \quad \text{and} \quad \mathbf{x}_r = \begin{bmatrix} x_2 & x_3 \end{bmatrix}^T$$

We have collected each adjacent pair of coordinates. Defining the summand vector as

$$\mathbf{g}(\mathbf{x}) = -(\mathbf{x}_r + 47\mathbb{1}_n)\sin\sqrt{|\mathbf{x}_r + 0.5\mathbf{x}_t + 47\mathbb{1}_n|} - \mathbf{x}_t \sin\sqrt{|\mathbf{x}_t - \mathbf{x}_r - 47\mathbb{1}_n|} \tag{2}$$

Where $\mathbb{1}_n$ is the length n vector of ones. We are then able to sum over this vector $\mathbf{g}$. Note that this method has removed the need for a Python `for` loop in favour of vectorised *numpy* methods. The full implementation of this method (and the naïve method) is included in `function.py`.

The resulting speed-up of this method is visualised in figure (1) by comparing the run-times for 15,000 evaluations of random samples each drawn from unit Gaussian for increasing dimension. Both methods have $\mathcal{O}(n)$ complexity but the vectorised method is roughly two orders of magnitude faster, as can be seen by comparing the scaling of each y-axis. The vectorised times are short enough at the dimensions considered to still be significantly affected by random fluctuations in background processes so the values plotted are averaged over 20 seperate runs.



(a) Both together

(b) Vectorised alone

Figure 1: Running times for 15,000 evaluations of the objectives for each method

2.2 The Solution Class

Underlying the implementation of each method is a representation of the candidate solutions. For this, a `Solution` base object is used which stores the coordinates and dimension of the problem, as well as the corresponding value of the objective function - this is intended to reduce objective function evaluations since any methods operating on instances of `Solution` can find the objective in memory rather than re-evaluating it.

2.3 Simulated Annealing

The Simulated Annealing method is initialised using shotgun initialisation by taking a specified number of samples uniformly in the feasible region, evaluating their objective and choosing the best as a starting point for the chain.

In the chains, steps are proposed as $D\mathbf{u}$ where $\mathbf{u}$ is a length n uniform random variable with each value in $[-1, 1]$ and D is a diagonal matrix that defines the maximum step size. D is initialised as $\kappa\mathcal{I}$ ($\mathcal{I}$ is the identity matrix) and defines the scaling of the steps for each coordinate. D is updated dynamically after each successful step according to the step sizes where $R_{kk} = |D_{kk}u_k|$

$$D_{j+1} = (1 - \gamma)D_j + \gamma\omega R$$

with $\gamma = 0.1$ and $\omega = 2.1$. Additionally, limits are imposed on the possible values of D so that steps are not allowed to become too small or too large. As suggested in the notes, the acceptance probability for a step that increases the objective function incorporates the step size using $\hat{d}$.

The initial temperature is estimated by running the chain for a short while and accepting every move (i.e. infinite temperature) then calculating the mean increase in the objective function $\bar{\delta f^+}$ scaled by the euclidean norm of the step size

$$\bar{\delta f^+} = \frac{1}{N} \sum_{i=1}^{N} \frac{\delta f^{+(i)}}{\hat{d}^{(i)}}, \quad \text{where} \quad \hat{d}^2 = \sum_{k=1}^{N} R_{kk}^2$$

This can be used to estimate the required temperature for which positive moves are accepted with probability 0.8 in the early stages of the actual annealing schedule.

The chains are run for either a specified number of iterations L, or after a specified number of acceptances, η, occur - L is varied for comparison and $\eta = 0.6L$. After this happens the temperature is decremented using an exponential cooling scheme such that $T_{l+1} = \alpha T_l$ with α allowed to vary for comparisons. The algorithm terminates once eight consecutive chains run with no improvement in the best solution (or after 15,000 evaluations).

I found it quite effective to re-initialise the chain from the current best solution found after each temperature decrement, particularly as the temperatures dropped. This is likely to work less well in more disjoint spaces, though a similar approach could be taken by considering multiple chains initialised at a few of the best solutions found.

The eggholder function is also extended such that should any of it's input coordinates lie outside the feasible region, it will return infinity. Therefore, whenever a step takes the chain into the infeasible region, the move will be accepted with probability zero.

2.4 Evolution Strategies

Solutions in an Evolution Strategy are extended to include strategy parameters by creating a class `esSolution` which inherits from `Solution`. These parameters are used for mutating the solutions, and are updated during recombination. The covariances (or equivalent rotation angles) are all initialised to zero and the variances are some specified value σ.

The algorithm is initialised by generating an initial population uniformly over the feasible region and immediately selecting a set of parents from the best solutions. Mutation is implemented by mutating the strategy parameters as outlined in the handout, then sampling randomly from a zero mean multivariate normal with covariance matrix corresponding to the new strategy parameters (after checking that the new matrix is positive definite) to mutate the coordinates.

It was common during mutation for candidate solutions near the boundary of the feasible region to step out after mutation, in which case the mutation of the newly infeasible coordinate was dropped, but steps in the other dimensions were kept. This is particularly important for this problem since the global optimum lies on the constraint boundary, so if we simply threw away newly feasible solutions then solutions near the global optimum would be routinely thrown away. Another option would be to simply not move at all if any step becomes infeasible, but this would likely reduce the quality of the search.

New generations are bred by recombination, which generates a new set of offspring from the parents. In this report discrete recombination was used for coordinates whilst intermediate recombination (with weight 0.5) was used for strategy parameters. In each case, the recombined strategy parameters were resampled until they gave an offspring with a positive definite covariance matrix.

The Evolution Strategy was deemed to have converged once the absolute difference between the best and worst parent solutions was less than some specified limit, $|f_b - f_w| < \epsilon$.

2.5 The Archive Class

Each instance of a strategy class (`SimulatedAnnealing` or `EvolutionStrategy`) contains an instance of the `Archive` class. The purpose of this is to store the best solutions found thus far, and is implemented on a best S dissimilar solutions principle. The dissimilarity of two solutions is measured using the euclidean norm of the vector joining their coordinates, and is compared using thresholds $D_{\min}$ and D_{sim}. The best solution found at each iteration of either method is considered as a candidate for archiving - the full logic of this can be seen in the `Archive.check_candidate()` method within `archive.py`. The purpose of archiving is to store information about the topology of the search space, and so that the best solution found can be retrieved when the search terminates (as there is no guarantee that the current location upon termination will be the best location found).

3 Visualising in 2D

This section will run through a representative successful example of each algorithm optimising the 2-dimensional eggholder function to validate that they are doing what is expected. The figures below are taken from a single run and so are not meant to indicate the quality of performance of each algorithm, only to show some salient features of their progression.

3.1 Simulated Annealing

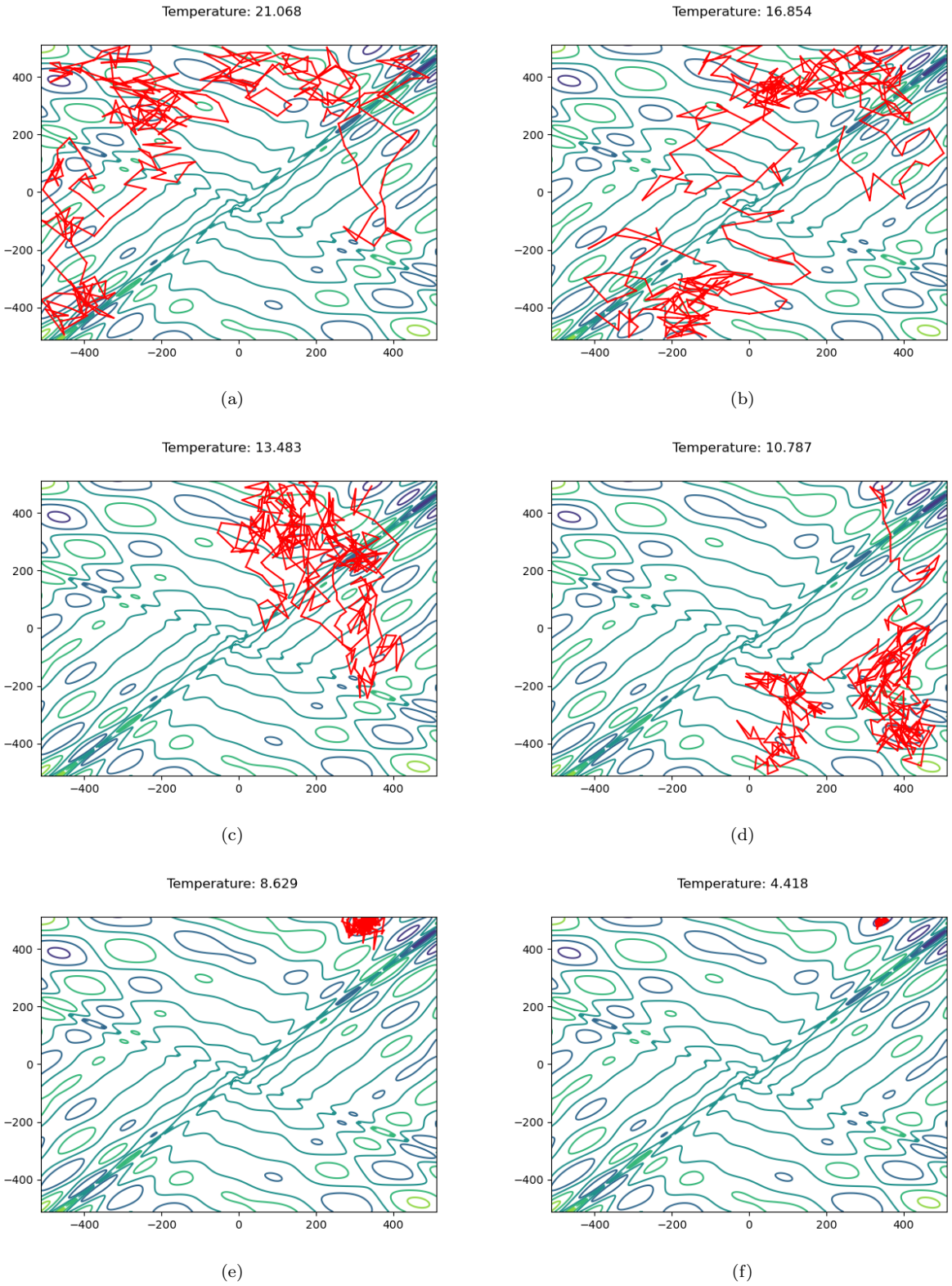
The sample run of Simulated Annealing used a minimum step size of 1, a maximum step size of 100 and initialised $D = 100\mathcal{I}$. 10 shotgun samples were used for initialisation, the initial temperature was estimated from 1000 samples and the maximum chain length was 500. The temperature was decremented using $\alpha = 0.85$ and the chain was run for 5,000 objective function evaluations.

Figures (2a) through (2d) show the first four chains. Initially the system takes large steps - it explores the function topology and stores important points in the archive, so that as the temperature decreases the chain knows in which areas to perform narrower searches. Figure (2e) shows a low temperature search which starts in a local optimum but is able to escape and explore some of the nearby peaks. Figure (2f) shows the final search which settles in this local optimum. The algorithm is reliant on being able to locate the global optimum in the high temperature searches, however, in this case the chain hopped over the global optimum.

3.2 Evolution Strategy

The test run of the Evolution strategy used 140 offspring and 20 parents, with a convergence limit $\epsilon = 10$ and the first generation was initialised with high variances to reflect the initial lack of confidence in the solution ($\sigma = 500$).

Figure (3a) shows the result of selecting the best 20 offspring from the initial generation, which has roughly located some troughs in the objective function. By generation 8, the solutions have clustered around the set of troughs near the global optimum. Figure (3d) shows the algorithm branching out and still exploring quite widely as the solution variances are still high. After 25 iterations, the solutions are tightly spread on



(a) (b) (c) (d) (e) (f)

Figure 2: An example run of simulated annealing

two peaks until convergence in the 29^{th} generation on a local optimum. The global optimum was stored in the archive from previous generations. Note that in other runs, the algorithms did settle on the global optimum.

4 Performance in 6D

To start with we consider the effect of varying a few parameters of each strategy on their behaviour and performance.

4.1 Simulated Annealing

4.1.1 Decrementing the temperature

The rate at which temperature is decremented is critical to the performance of simulated annealing - too slow and the algorithm spends needlessly long exploring the space at high temperatures, too fast and the system is likely to get stuck in a local optimum. For a fixed chain length of 500, and maximum and minimum steps of 100 and 1 respectively, the system was tested on 50 different seeds and the results are shown in table (4).

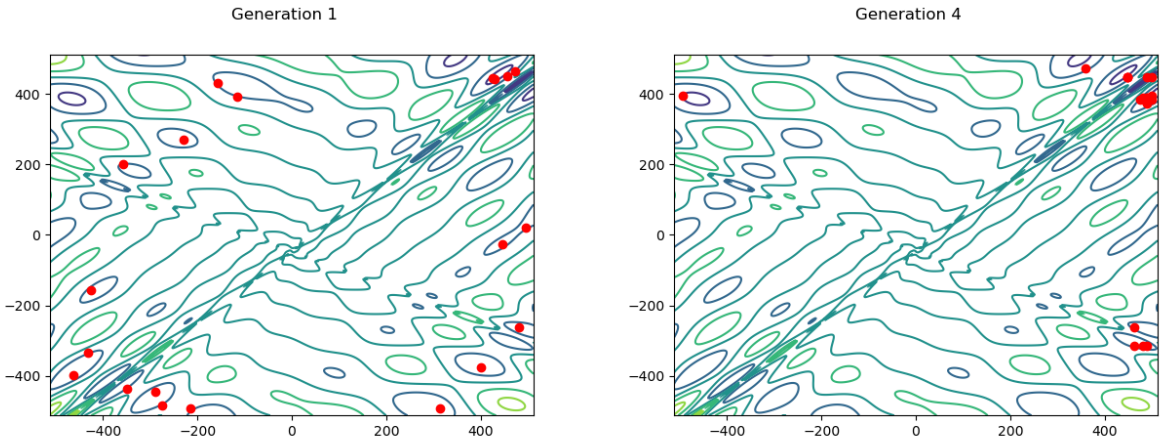
α	Mean Optimum	Best Optimum	Std. Dev.	Mean Evaluations
0.50	-3,414.0	-4,222.7	458.9	9,801.6
0.75	-3,645.0	-4,227.4	332.7	11,338.2
0.85	-3,751.9	-4,579.8	379.9	12,405.5
0.95	-3,185.6	-3,959.0	406.6	8,344.6
0.99	-3,140.9	-4,000.5	339.1	7,724.3

Figure 4: Results from varying α

Interestingly, high values of α led to earlier termination of the chain as the temperature is decremented so slowly that the searches are happening at similar temperatures, and so they are exploring the space at the same scale. I'd previously expected high values of α to not terminate early as they might be able to find minor improvements in the solution very slowly, but this was not the case. Very rapid decrements also led to early termination - the high variance suggest this is due to the chains getting stuck in local optima and not being able to escape. $\alpha = 0.85$ gave the best solution on average and still terminated early in some cases meaning the best optimum found had been explored well.

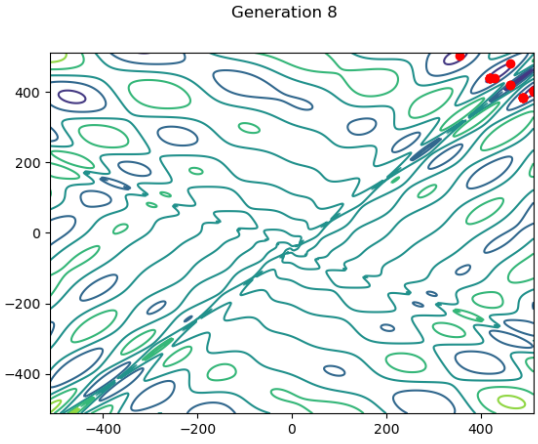
4.1.2 The D matrix

As stated previously, D controls the maximum step size allowed at each iteration. D is updated dynamically, and is subject to a maximum (and minimum) limit on its values. This is intended to stop the steps becoming too large and erratically hopping across the space, and also to avoid the situation where very few steps are accepted at low temperatures because all proposed steps are so big. Experiments were carried out with $\alpha = 0.85$ and $L = 500$.

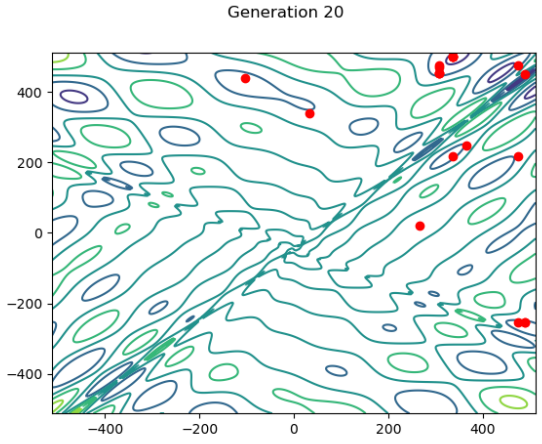


(a)

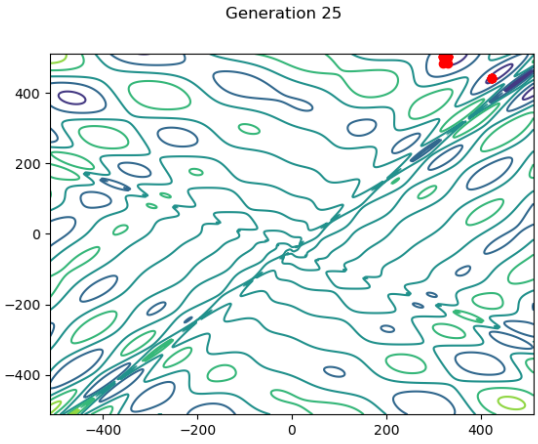
(b)



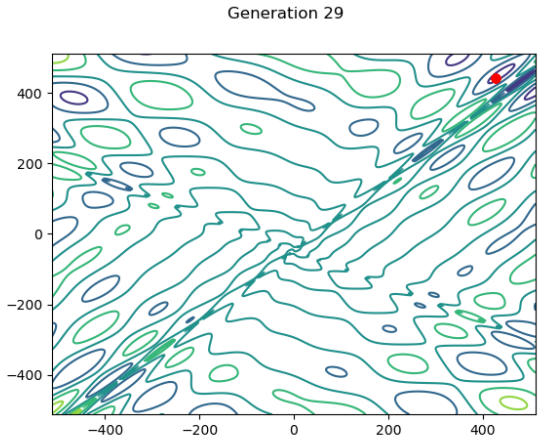
(c)



(d)



(e)



(f)

Figure 3: An example run of an evolution strategy

Maximum Step	Mean Optimum	Best Optimum	Std. Dev.	Mean Evaluations
25	-3508.3	-4244.4	341.9	11980.3
50	-3640.4	-4358.6	336.9	12462.8
100	-3755.1	-4592.2	397.5	12128.6
150	-3533.7	-4445.0	541.6	10627.3
200	-3534.5	-4545.2	523.5	11634.1

Figure 5: Results from varying maximum step size

The effect of the maximum step size is shown in table (5). Using a maximum step size of 100 appeared to be the best of the values considered - it had a lower variance than higher values and provided a better mean optimum than the lower values. An example evolution of the size of D is plotted in figure (6) by plotting $\frac{1}{6}\text{Trace}(D)$ as a function of evaluations. For all values there is an initial drop, which then recovers as the

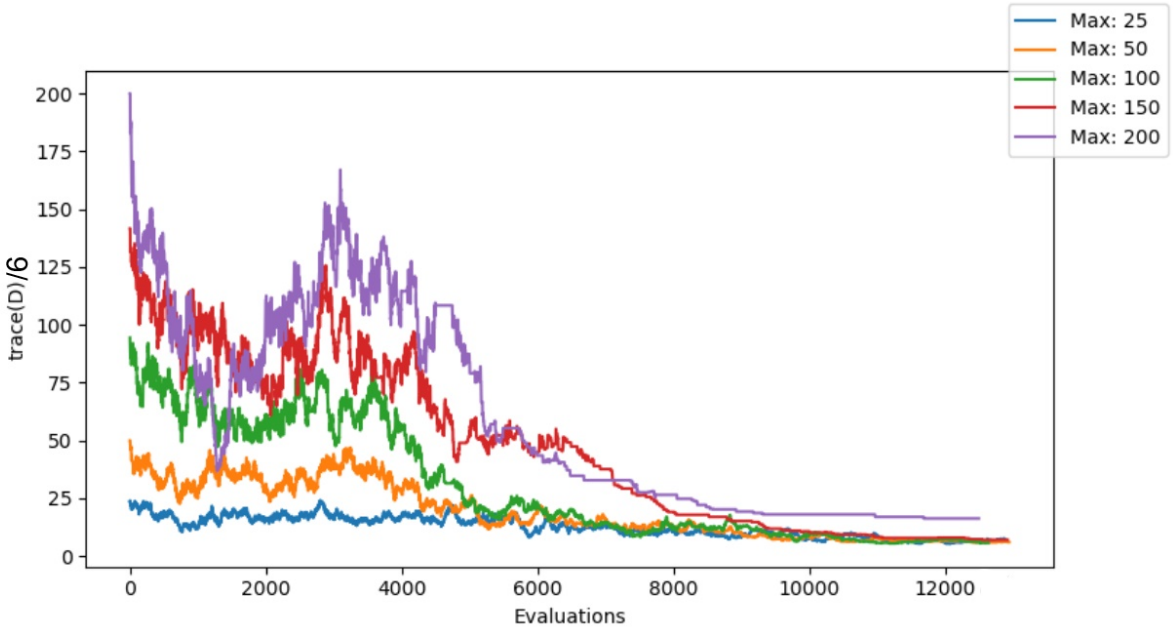


Figure 6: Evolution of D

algorithm runs. The step sizes then taper off as the algorithm starts to only accept smaller moves as the temperature decreases. Figure (7) shows these trace values scaled by the maximum specified step size (with the less important values as thinner lines). In all cases, the steps drop to about 0.6 of the allowed maximum and then fluctuate between about 0.4 and 0.8 before dropping to about 0.1. Maximum steps of 100 and 150 are the first to drop to low temperatures which suggests that they are having the most acceptances in the early chains and are therefore exploring the space the best.

4.1.3 Chain length

Choosing the length of the chain is a balancing act between giving the system enough time to explore the whole space well at high temperatures, whilst also not spending too long exploring a single trough at a short length scale. Experiments were carried out with $\alpha = 0.85$ and maximum step of 100.

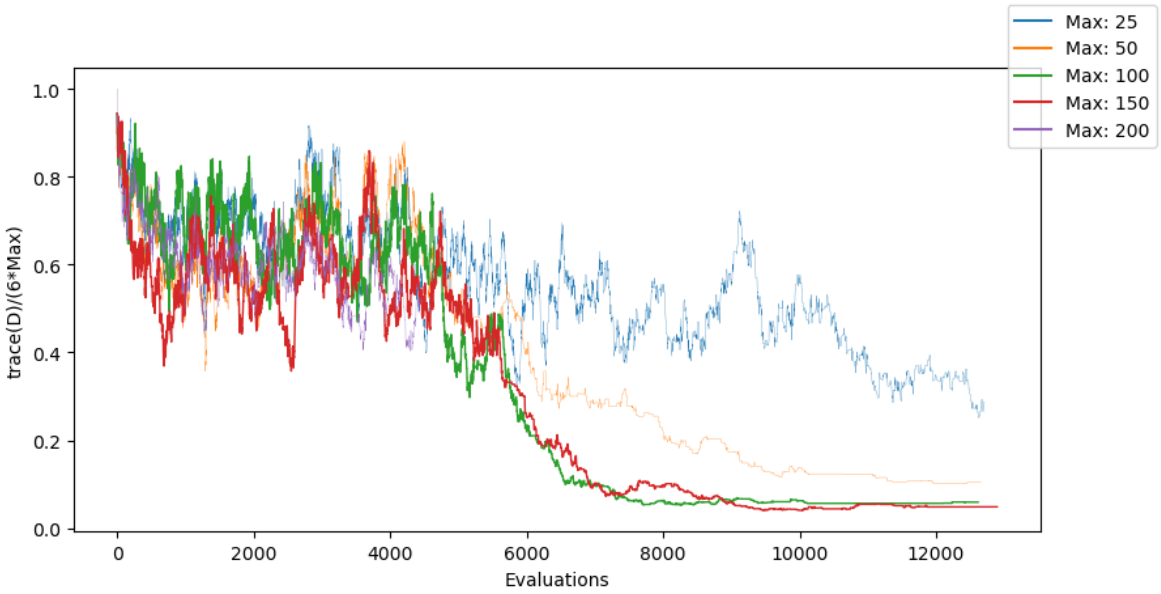


Figure 7: Evolution of D normalised by the specified maximum step size

Chain Length	Mean Optimum	Best Optimum	Std. Dev.	Mean Evaluations
50	-2880.0	-3812.2	434.5	975.16
150	-3305.0	-4413.38	503.2	4054.52
250	-3395.2	-4272.4	448.9	8044.82
350	-3584.9	-4402.9	519.7	10253.6
450	-3771.9	-4516.6	369.4	11674.2
550	-3647.7	-4299.7	375.3	12412.5
650	-3678.4	-4570.1	373.6	12557.1
750	-3763.6	-4584.9	368.4	12392.5

Figure 8: Results from varying chain length

Table (8) shows the results. Short chains gave poor solutions or had high variance so were much less reliable, meaning the system was consistently finding different local minima. The longer chains offered little to no decrease in solution variance for the extra computation required, suggesting that chains longer than 450 were a waste of resources.

4.2 Evolution Strategy

4.2.1 Number of offspring

Each time a new generation of offspring are generated, their objectives must be evaluated and hence large population sizes represent a more costly procedure. Therefore the population should be large enough that the offspring are bred from a 'genetically diverse' population, without excessive computation at each generation. For a fixed convergence limit $\epsilon = 10$ and using 20 parents, the parent:child ratio was varied.

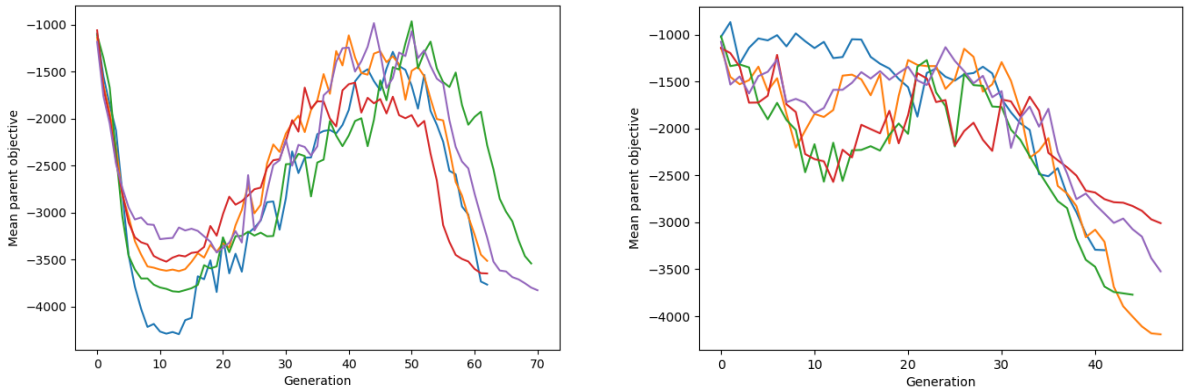
Ratio	Mean Optimum	Best Optimum	Std. Dev.	Mean Evaluations
1:2	-3248.1	-4422.7	403.6	1208.0
1:3	-3546.2	-4428.4	314.4	2422.0
1:4	-3662.1	-4216.9	266.2	3664.2
1:5	-3752.2	-4533.4	337.0	4020.6
1:6	-3792.7	-4465.7	267.6	5179.2
1:7	-3946.2	-4546.9	268.7	6280.4
1:8	-3911.6	-4536.9	302.4	7377.6
1:9	-3955.9	-4512.4	279.6	9246.4
1:10	-3932.0	-4519.6	301.0	10784.0

Figure 9: Results from varying the ratio of parents to offspring

Table (9) shows solution statistics for these ratios - the initial population was generated with $\sigma = 500$. The results show that the 1:7 ratio suggested in the handout is a good trade-off between computation time and solution quality as there is a big rise in mean optimum with roughly the same variance when increasing from 1:6 to 1:7. Lower ratios had a worse mean optimum and higher ratios were more computationally expensive.

4.2.2 Initial variance

The variance strategy parameters of the initial population should reflect an initial lack of confidence that the parents are located at optima, since the space has not been well searched. I investigated the effect of initialising the first population with a low variance on the evolution of the parents.



(a) $\sigma^2 = 10$ (b) $\sigma^2 = 2500$

Figure 10: Mean parent objective as a function of generation

The effect of this low initial variance is clear in figure (10). The initial population starts off with high confidence that it is near a global optimum, and therefore subsequent generations search the space local to the selected parents. As the search stagnates in this local optima the variances suddenly increase and the search opens up to the whole space (around 10 generations in figure (10a)), before eventually settling on new optimum. This initial local search can bypassed by setting the initial variance to a high value, which in turn reduces the number of generations required and speeds up the algorithm.

4.3 Global recombination

Finally, one possible method to alleviate the lack of genetic diversity in lower population counts is to use global recombination. In contrast to the form of recombination used previously, global recombination allows each offspring to inherit features from many different parents rather than just four (two for the coordinates, two for the strategy parameters).

Ratio	Mean Optimum	Best Optimum	Std. Dev.	Mean Evaluations
1:2	-2896.9	-3779.2	294.7	2360.0
1:3	-3285.8	-4161.1	350.8	3165.6
1:4	-3497.9	-4181.8	350.2	3908.8
1:5	-3643.0	-4479.7	329.1	4724.0
1:6	-3703.5	-4432.9	335.3	5488.8
1:7	-3756.6	-4371.9	305.8	6384.0
1:8	-3869.2	-4468.9	314.7	7084.8
1:9	-3836.6	-4425.0	310.8	7495.2
1:10	-3914.8	-4527.1	316.1	8452.0

Figure 11: Results from using global recombination

In fact it appears across the board that this has slightly reduced the quality of the solutions in both in terms of mean optimum and standard deviation and so the two parent methods of recombination should be preferred. There is a slight improvement in number of evaluations, but this is negligible over the timescales considered (plus the sampling procedure for global recombination is slightly more time consuming).

5 Overall Performance

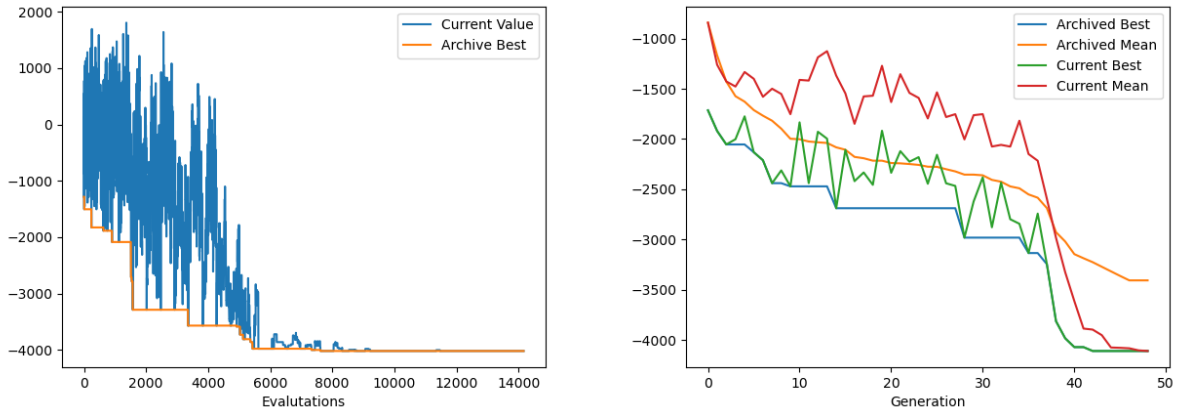
In this final section we compare the performance of the algorithms using the best parameters found in the previous section against the `scipy` implementation of the *dual annealing* algorithm.

The plots in figure (12) visualise an example convergence in value of each algorithm. The evolution strategy appears to be running for the right amount of time as the best objective and mean objective converge near to the end of the running; whereas the simulated annealing seems to be running for too long - there are nearly 6000 iterations with minimal improvement in objective. Therefore if I were to repeat these experiments, I would relax the convergence criterion for simulated annealing possibly from 8 to 4 chains with no improvement.

Algorithm	Mean Optimum	Mean Execution Time (s)
Simulated Annealing	-3760.42	6.17
Evolution Strategy	-3952.65	1.92
Dual Annealing	-4465.99	23.30

Figure 13: Mean optimum found and execution time for each algorithm over 50 seeds

As shown in table (13), of the two methods I implemented, the evolution strategy was more performant both in terms of the mean optimum found and in the fact it ran over three times faster on average (though this would be reduced by relaxing the convergence criterion for simulated annealing). The library implementation of dual annealing outperformed both my implementations by some margin in terms of the quality of the solutions. It is important to consider, however, that the dual annealing algorithm is significantly more



(a) Simulated Annealing

(b) Evolution Strategy

Figure 12: Convergence of both methods on the same seed (30)

complex than the algorithms I implemented, as reflected in the running times of each algorithm. Also, with more time to code and experiment there are likely a few minor tweaks in parameters or implementation details for my algorithms that would give a healthy increase in solution quality for minimal extra computational cost.

6 Conclusion

In this coursework I have implemented two global optimisation methods using only the basic available mathematical computing framework. Throughout the work, a key consideration has been maintaining computational efficiency, particularly in the evaluation of the objective function as in higher dimensions this can dominate the running time of the algorithm. The behaviour of each algorithm was validated in two dimensions before proceeding to evaluate their performance in six dimensions. I experimented with the settings of system parameters and internal methods of the algorithms to try and find a rough set of optimal parameters so that I could suggest which of the two algorithms might be preferred in practice. Finally, I compared their performance to an out-of-the-box optimiser from the `scipy` library to see how much room for improvement there was in each implementation. Of the two methods I considered, Evolution Strategies came out on top both in terms of computational cost and solution quality.

My submission includes the required source code to run an example of each algorithm in `test.py`.