UNIVERSITY OF
CAMBRIDGE

Practical Optimisation
Simulated Annealing and Evolution Strategies analysis

5568D

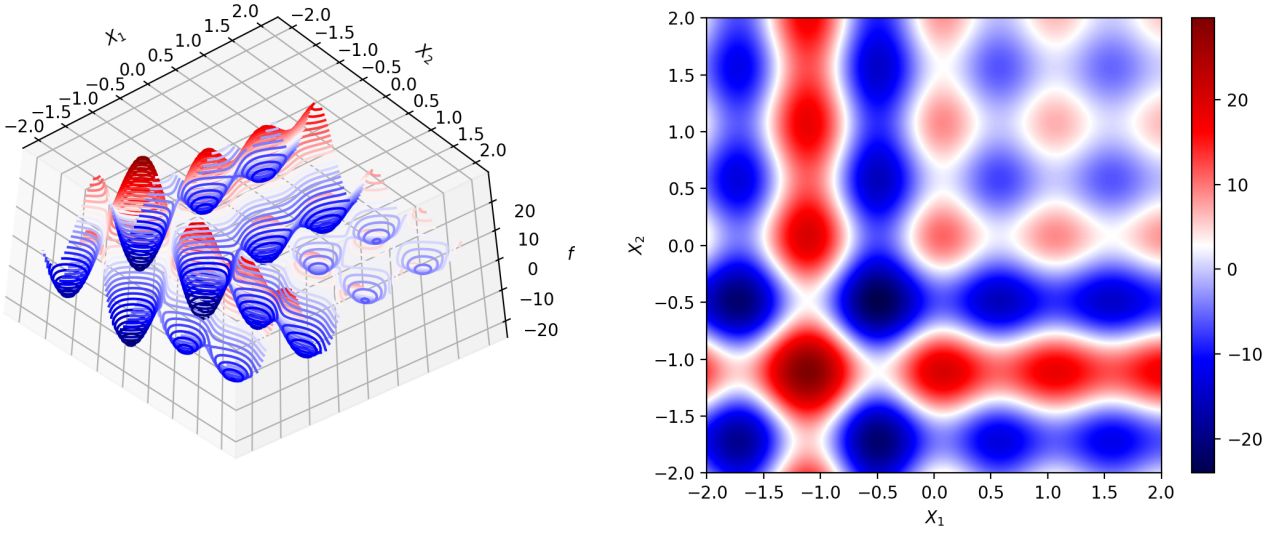
14th January 2020

1 Introduction

The aim of this coursework is to compare the performance of two optimisation algorithms that were studied in 4M17. Because of my interests are in the area of image processing and machine learning, **Simulated Annealing** and **Evolution Strategies** have been selected as they are widely used in these domains[1]. The objective is to use these strategies to optimise a n-dimensional constrained optimisation called *Shubert's Functions*. For the purpose of this coursework only the 5 dimensional and the 2 dimensional function will be studied and optimised. The *Shubert's Function* is displayed below:

$$\begin{aligned} &\text{Minimise} \quad f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{5} j \sin((j+1)x_i + j) \\ &\text{Subject to} \quad x_i \in [-2, 2] \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{1}$$

For the *Shubert's Function* described above, the global minimum in a 2D version is found to be-24.0625 at $[-0.4914, -0.4914]$. In the 5D version the global minimum is found to be -60.5. Due to its simple visualisation the 2D Shubert's Function is displayed below:



(a) 3D representation

(b) Contour representation

Figure 1: 2 dimensional (2D) visualisation of *Shubert's Function*

2 Simulated Annealing

2.1 Description

Simulated Annealing(SA) is a probabilistic method for finding a global optimum of a function that may posses several local minima [2]. It involves a random search which does not only accept changes that decrease the objective function but it also accepts some changes that increase the objective function with a certain probability [3]. Essentially the **SA** algorithm tries to simulate the physical process of cooling a solid which when eventually it will become 'frozen' it will happen at the minimum energy configuration[2]. This analogy comes from the acceptance probability which depends on the temperature, the increase in the function and the change in the step size. Out of these parameters, the most important one is the temperature which dictates how much can the function be increased. Therefore this parameter controls the space function is explored and how quickly it will converge to a minimum.

2.2 Implementation Details

The algorithm can be split into two parts: *Solution Representation & Generation* and *Annealing Schedule.*

2.2.1 Solution Representation & Generation

Solutions are generated according to a strategy suggested by Parks [4] which is showed below:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{D}\mathbf{u} \tag{2}$$

where $\mathbf{x}_i$ is the current solution, $\mathbf{u}$ is a vector of uniform random numbers in the range $[-1, 1]$ and $\mathbf{D}$ is a diagonal matrix which defines the maximum change allowed in each variable. If the new solution is accepted, $\mathbf{D}$ is updated according to:

$$\mathbf{D}_{j+1} = (1 - \alpha)\mathbf{D}_j + \alpha\omega\mathbf{R} \tag{3}$$

where $\mathbf{R}$ is a diagonal matrix the elements of which consists of the magnitude of the successful changes made to each of the control variable:

$$\mathbf{R}_{kk} = |\mathbf{D}_{kk}\mathbf{u}_k| \tag{4}$$

and the damping constant α controls the rate at which information from $\mathbf{R}$ is folded into $\mathbf{D}$ with weighting ω. For the purpose of this optimisation the values of α and ω are set to the suggested values 0.1 and 2.1 [3].

The probability of accepting an increase in the objective function is defined as:

$$\begin{aligned} p &= \exp(-\frac{\delta f}{T\bar{d}}) \\ \text{where} \quad \bar{d} &= \sqrt{\sum_{k=1}^{N} \mathbf{R}_{kk}^2} \end{aligned} \tag{5}$$

2.2.2 Annealing Schedule

The first point that has to be taken into account is the initial temperature. This is found by performing an initial search where all solutions are accepted. σ_0 represents the standard deviation of the variation in the objective function found by using all the accepted solutions. This standard deviation can be used as an initial temperature $T_0 = \sigma_0$.

The next part of the Annealing process is the length of the Markov Chain used at each temperature and minimum accepted trials. The length of the Markov Chain, L_k, depends on the size of the problem. Therefore for this coursework it will be parameter which will be optimised. The minimum number of trials that should be accepted $\eta_{\min}$ is set to the suggested value of $\eta_{\min=0.6L_k}$.

In order for the algorithm to work it is required to decrease the temperature. In these coursework, 2 rules for decreasing the algorithm will be studied. These are described below:

$$T_{k+1} = \alpha T_k \quad \text{where}\, \alpha = 0.95 \tag{6}$$

$$T_{k+1} = \alpha_k T_k \quad \text{where}\, \alpha_k = \max[0.5, exp(-\frac{0.7T_k}{\sigma_k})] \tag{7}$$

where σ_k is the standard deviation of the objective function values accepted at temperature T_k

The process of the *Annealing Schedule* can be summarised by the following figure below:

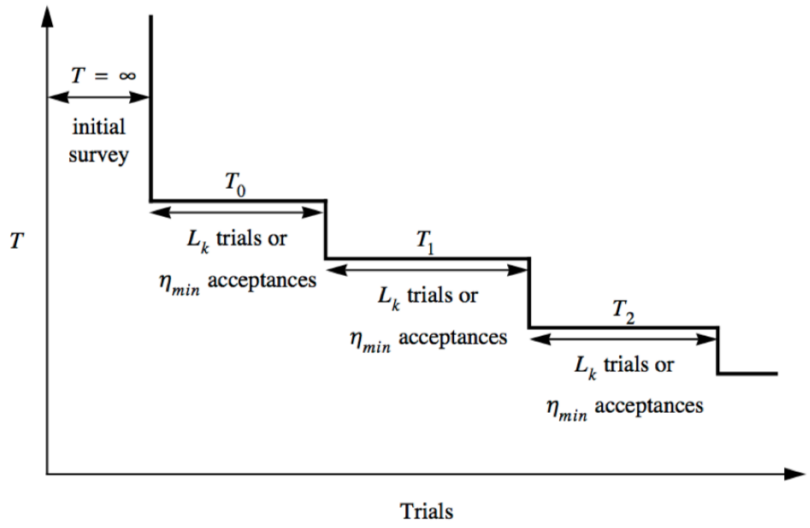


Figure 2: Graphical Representation of the Annealing Process [3]

2.3 Stopping and Restarting Strategies

Ideally, the **Simulated Annealing** algorithm will be terminated once the temperature will become zero. However, due to the fact, that temperature decreasing strategies prevent that from happening it is required to terminate the algorithm after a certain number of evaluations. For the algorithm implementation, 10 000 evaluations are selected.

Because of the way the **Simulated Annealing** algorithm is structured, it sometimes gets stuck in a local optima. To help it escape a restart is required while keeping the temperature constant. For this coursework, two different strategies are are implemented. The first one is to restart from the best solution found so far. As observed these could cause a potential problem if algorithm gets stuck in a local optima very early. The second strategy is to restart from a random location.

2.4 Application and Analysis of SA

By running the algorithm 30 times with different seed with the following strategies: adaptive temperature strategy and best solution restart strategy, but by varying the values of the L_k for 2D and 5D the mean of global minimum is found. All L_k are scaled by the dimension. The results are displayed below.

Iteration	2D	5D	Iteration	2D	5D	Iteration	2D	5D
50	−24.0222	−56.2697	550	−23.9964	−55.2973	1050	−23.9923	−55.9224
100	−24.0496	−56.7897	600	−24.0277	−54.9648	1100	−23.9628	−56.0182
150	−24.0568	−56.6069	650	−24.0068	−55.0483	1150	−23.9604	−56.4226
200	−24.0556	−56.3202	700	−24.0054	−55.4692	1200	−23.9633	−55.6031
250	−24.0589	−56.2697	750	−24.0183	−55.5839	1250	−23.9587	−55.6469
300	−24.0610	−57.5416	800	−24.0285	−55.9624	1300	−23.9953	−55.8324
350	−24.0616	−56.9441	850	−24.0086	−55.7439	1350	−23.9909	−55.2831
400	−24.0615	−55.7399	900	−24.0059	−56.2384	1400	−23.9923	−55.4137
450	−24.0623	−56.3530	950	−23.9719	−55.8773	1450	−24.0041	−55.7991
500	−24.0174	−56.5048	1000	−56.0585	0.5750	1500	−23.9981	−55.4137

Table 1: Values of minimisation point found with different values of L_k

As observed at low values of L_k(light grey) the values found are very not at all accurate. This is because due to the low number of trials, the temperature decreases quickly, which could limit the amount of space explored by the function. Therefore it could get stuck in local minima. On the other hand, at large values of L_k(dark grey) it is observed that the minimum found is very similar between the results. This is because at large temperature the space around the global minimum will be not be explored into detail. Therefore a medium value of L_k should be good for finding the minimum.

So now, by looking at different strategies for both temperature and restart and by varying L_k to a sensible range. The best values found the respective strategies are displayed below.

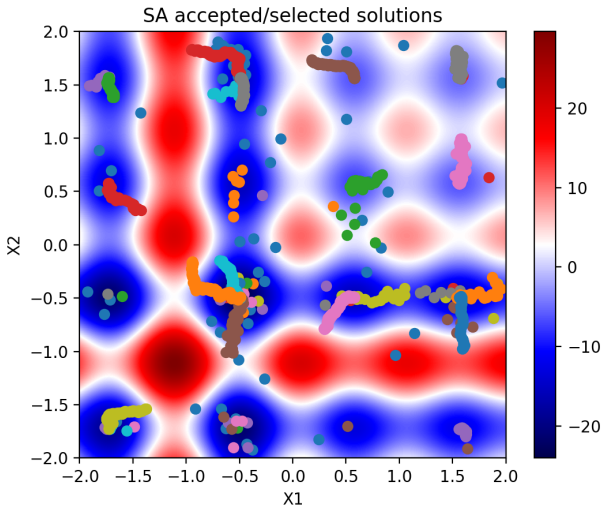
Dimension	Temperature Strategy	Restart Strategy	L_k	Global Minimum Mean	**Ratio**
2D	Adaptive	Best Solution	300	-24.05780	0.9
	Constant	Best Solution	150	-24.0624	0.8
	Adaptive	Random	150	-24.0600	0.93
	Constant	Random	150	-24.0564	0.6
5D	Adaptive	Best Solution	1125	-57.9535	0.4
	Constant	Best Solution	250	-57.7136	0.4
	Adaptive	Random	1250	-57.8055	0.27
	Constant	Random	1250	-57.7722	0.33

Table 2: Best Performance of different Strategies

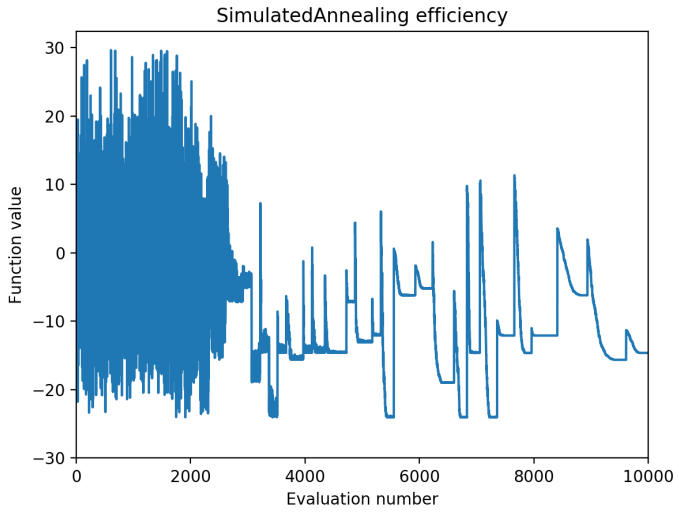
It is important to look at the ratio as this is the parameter which looks at the reliability of the algorithm as it measures the percent of the total 30 runs in which the algorithm gets very close to the optimum value.As observed having an decreasing temperature strategy which takes into account the accepted value at a certain temperature is much better than having a fixed decreasing constant. It would have been expected the way the decrease in temperature is controlled by the standard deviation of the accepted which will influence if a region is needed to be explored or not.

In terms of the restarting strategy it is observed that for a 5D evaluation the algorithm gets less stuck in local minima and therefore restarting from the best solution found is a much better choice. For the 2D evaluation it is observed that restarting from a random position is a much better approach as it would let the algorithm escape the local minima.

To observe how the **Simulated Annealing** performs a 2D representation containing all the accepted solutions is observed in the plot below:



(a) Accepted solutions on the 2D *Shubert's Function*



(b) **SA** evaluations

It is observed from *Figure 3a* that the **SA** is performing as expected. The 2D **SA** has been run with $L_k = 75$, an adaptive temperature strategy and a random restart strategy, each colour changing in the points is due to a restart. Because of the low number of max trials the temperature decreases very fast and therefore each region is explored thoroughly. Because of the random restarts strategy, it prevents the algorithm from getting stuck in local minima. However, it is observed that most region looked at is the one with the global minimum. *Figure 3b* reiterates the fact that the algorithm does explore each region with a potential of global minimum before it does a restart.

3 Evolution Strategies

3.1 Description

Evolution Strategies (ES) is an algorithm which imitates the principles of organic evolution as a method to find the global minimum of a cost function. Therefore, based on the modern synthetic theory of evolution [5] the fittest individuals are selected as parents, their genes are recombined and mutated to produce a new set of population. This process is repeated until the population converges to a minimum solution. The advantage of the **ES** algorithm is that it allows for self-adaptation which means that not only the values are mutated but also the parameters for each member of the population.

3.2 Implementation Details

Firstly, it is important to note what the individual member of the population is being used to be transmitted forward to its "children".The values of each member are n, where n is dimension studied. Apart from these, there are the strategies parameters which are split into n parameters that represent the standard deviation of the mutation and there can also be an additional *n(n-1)/2* parameters which define the rotation of the distribution.

The core process of the **ES** algorithm can be split into three parts: *Selection*, *Recombination* and *Mutation*. In these steps, some different strategies will be tried and tested so that the best performance is found.

3.2.1 Selection

It will be noted that λ will be the number of offspring and μ will be the number of parents. Two strategies will be studied. The first one is called $(\mu + \lambda) - ES$. This strategy consists of the best μ parents producing λ offspring which are reduced again to the μ parents of the next generation. Parents survive until they are superseded by a better offspring [5]. The other strategy called $(\mu, \lambda) - ES$ involves only the offspring undergoing selection and therefore limiting the lifetime of a parent to one generation.

3.2.2 Recombination

Following the suggestion in the lecture notes [6] it is recommended that the control variables be updated using discrete recombination where the offspring inherits its control variable from one or other two randomly selected parents, the parent to contribute each component being chosen by a series of 'coin tosses'. At the same time, strategy parameters are updated using intermediate recombination where the offspring solution (O) inherits components which are a weighted average of the components from two randomly selected parents (1 and 2):

$$u_{Ok} = \omega u_{1k} + (1 - \omega) u_{2k} \tag{8}$$

where $u_k$is a strategy parameter.

3.2.3 Mutation

Again, here two strategies are implemented which will be tested to observed their performance. These are either introducing a rotation or not such as the covariance matrix will be build according to:

$$\mathbf{A}_{ij}^{-1} = \frac{\sigma_i^2 - \sigma_j^2}{2} \tan(2\alpha_{ij}) \quad \text{with rotation} \tag{9}$$

$$\mathbf{A}_{ii}^{-1} = \sigma_i^2 \quad \text{without rotation} \tag{10}$$

The standard deviation and the rotation angles are mutated according to:

$$\sigma_i' = \sigma_i \exp(\tau' \times \chi_0 + \tau \times \chi_i) \tag{11}$$

$$\alpha_{ij}' = \alpha_{ij} + \beta \times \chi_{ij} \tag{12}$$

where the χs are (different) random numbers sampled from a normally distributed one- dimensional random variable with zero mean and unity standard deviation. τ,τ' and β are control variable which are given the following recommended values according to the notes [6]:

$$\tau = \frac{1}{\sqrt{2\sqrt{n}}} \tag{13}$$

$$\tau' = \frac{1}{\sqrt{2n}} \tag{14}$$

$$\beta = 0.0873 \tag{15}$$

Now that the mutation of the strategy parameters has been defined it is time to look at the control variables. These are mutated according to:

$$\mathbf{x}' = \mathbf{x} + \mathbf{n} \tag{16}$$

where $\mathbf{n}$ is a vector of random numbers sampled from the n-dimensional normal distribution with zero means and the probability density function equal to:

$$p(\mathbf{z}) = \sqrt{\frac{|\mathbf{A}|}{(2\pi)^n}} \exp(-\frac{1}{2}\mathbf{z}^T\mathbf{A}\mathbf{z}) \tag{17}$$

3.3 Stopping Criterion

The algorithm will be terminated once a certain number of evaluations will be achieved. This will depend on the population size and therefore different it will limit how many generations will be available.

3.4 Application and Analysis of ES

Firstly, it is looked at how the size of the population affects the algorithm. By running the **ES** algorithm 30 times each time with a different ratio of the global min over the number of runs is calculated. From some early runs, it has been observed that **ES** algorithm is very reliable achieving a ratio of 1.0 most of the times, so therefore, it is now important to take into account how fast will the algorithm will converge. Therefore the average evaluation at which the algorithm reaches the global min is also taken into account. The results for different population sizes with a rotation strategy and $(\mu, \lambda) - ES$ strategy are displayed below for 5D and a 2D *Shubert Function.*

	2D		5D	
Iteration	Ratio	Avg. End Iteration	Ratio	Avg. End Iteration
56	0.43	350.9	0.010	115.7
112	0.77	709.3	0.020	253.9
168	0.97	1187.2	0.025	369.6
224	1.0	1582.9	0.033	604.8
280	1.0	2053.3	0.033	914.7
336	1.0	2542.4	0.033	1321.6
⋮	⋮	⋮	⋮	⋮
784	1.0	6376.5	0.83	6507.2
840	1.0	7112.0	0.77	6832.0
896	1.0	7556.3	0.9	7675.7
952	1.0	7933.3	0.9	8028.5
⋮	⋮	⋮	⋮	⋮
2072	0.97	10000.0	0.033	10000.0
2128	1.0	10000.0	0.067	10000.0
2184	1.0	10000.0	0.025	10000.0
2240	1.0	10000.0	0.020	10000.0

Table 3: Values of the global ratio found with different values of population sizes

As observed the population size proportional to the dimensionality of the problem. It is observed that the **ES** algorithm performs worse at small and large population sizes. Not only this but also the average end iteration increases linearly with the population size independently of the dimensionality of the problem. This is because at low population sizes the algorithm converges quickly on a slope and therefore does not allow for wide initial surface exploration. At large population sizes, the algorithm takes a long time before it converges to a region where the global minimum is found.

Now, that a suitable range of population sizes have been selected it is time to at how different strategies perform. The results of the best ratios are found below:

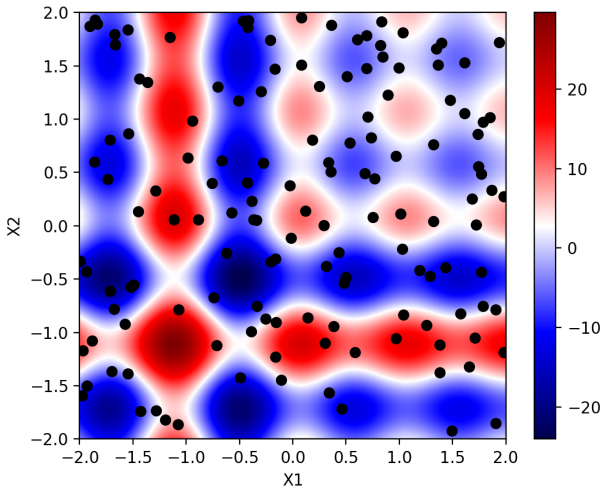
Dim.	Rot. Strategy	Selection Strategy	Pop. Size	Global Min Mean	**Ratio**	**Avg. End Eval**
2D	Yes	$(\mu+\lambda)-ES$	140	-24.0601	1.0	1433.6
	Yes	$(\mu,\lambda)-ES$	182	-24.0589	1.0	1579.76
	No	$(\mu+\lambda)-ES$	140	-24.0598	1.0	1131.2
	No	$(\mu,\lambda)-ES$	182	-24.0577	1.0	1230.32
5D	Yes	$(\mu+\lambda)-ES$	742	-58.4633	0.15	3534.6
	Yes	$(\mu,\lambda)-ES$	994	-59.2144	0.2	3954.9
	No	$(\mu+\lambda)-ES$	966	-59.4167	1.0	9146.6
	No	$(\mu,\lambda)-ES$	966	-59.2324	0.93	8309.4

Table 4: Best Performance of different strategies

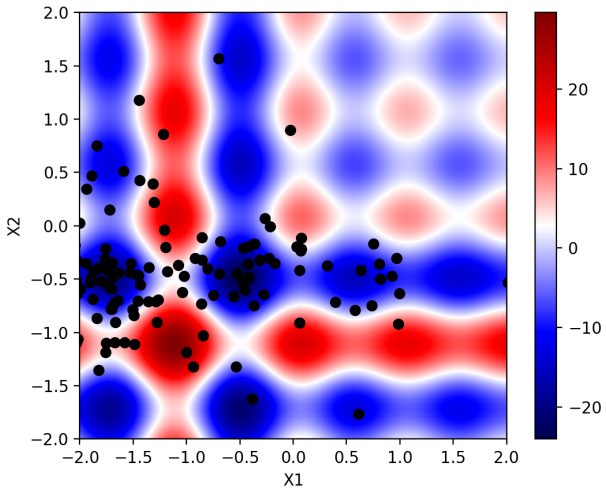
Firstly, the Selection Strategy is discussed. As observed, $(\mu+\lambda)-ES$ does perform better in a 2D problem while in a 5D problem $(\mu,\lambda)-ES$ strategy could perform better if the efficiency is taken more into account. The deficiencies of $(\mu+\lambda)-ES$ lies in the fact that it is possible to get stuck in an out-dated good location if the internal parameter setting becomes unsuitable to jump to the new field of possible improvements [5]. By using $(\mu,\lambda)-ES$ strategy this is avoided due to the limited lifetime even if it may result in short phases of recession but it avoid long term stagnation phases due to mis-adapted strategy parameters [5].

Looking at the Rotation Strategy it is observed that by allowing it, the algorithm performs worse both in 2D and 5D as it would take longer for the algorithm to reach the end evaluation. The reason why might a rotation strategy be useful is the fact that mutation performs a hill-climbing search procedure. Because of σ_i for each control variable x_i the preferred directions of search can only be along the main axes [5]. Therefore because this might not be the best search direction. Rotation is introduced so that mutations are correlated to one another which will allow movement not only in axis direction. However it is observed that for this algorithm having correlated control variables makes it harder for the algorithm to find it's global minimum.

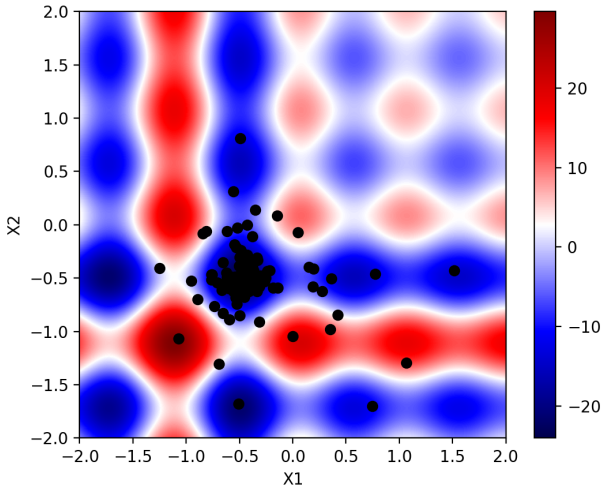
To observe how the **Evolution Strategies** performs, a 2D representation of the algorithm at different generation of 'kids' can be observed in the figure below along with how the algorithm performs with the number of evaluations.



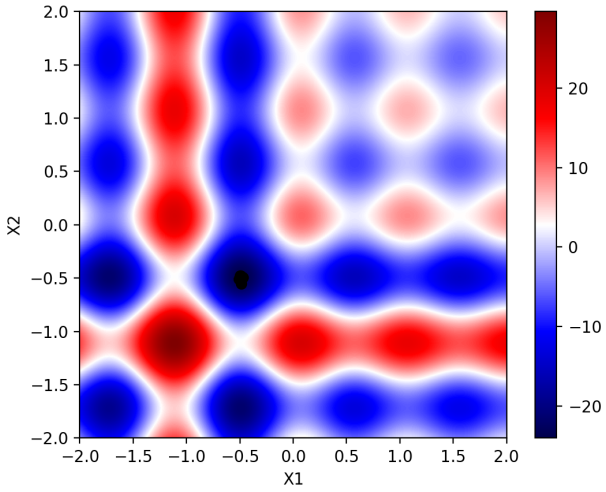
(a) 1st Generation



(b) 3rd Generation



(c) 5th Generation



(d) 15th Generation

Figure 4: Visual representation of how **Evolution Strategies** works

As observed in the figure above the **ES** algorithm performs as expected. It is seen how the process of mutation affects the population such that it will cause an uphill movement. The 2D global min was found at the correct value of -24.0625 at $[-0.4914, -0.4914]$.

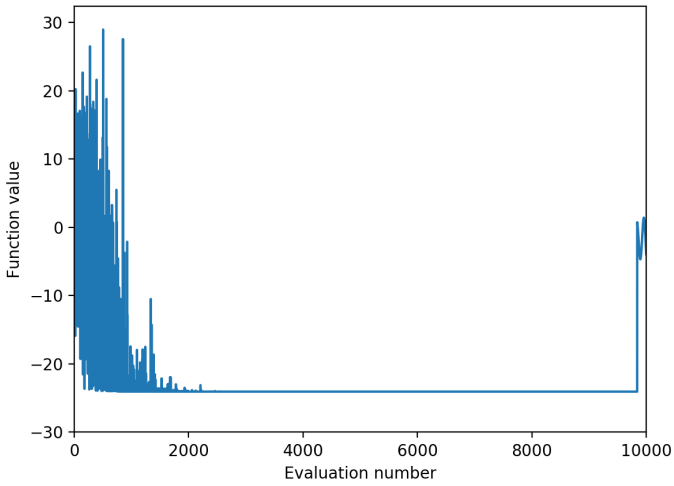


Figure 5: **Evolution Strategies** performance based on the number of evaluations

4 Comparison

4.1 Reliability

As already established and clearly observed from previous sections the ratio of **Evolution Strategies** algorithm is generally higher than the **Simulated Annealing** algorithm. The reason for this represents one of the key differences between these algorithms. **Simulated Annealing** is working with one solution at a time while **Evolution Strategies** works with a population of solution at a time.

The reliability of the algorithms was also measured by running the algorithm with the same set of seeds each run for 200 runs and storing the best values found. The results can be seen in the figure below:

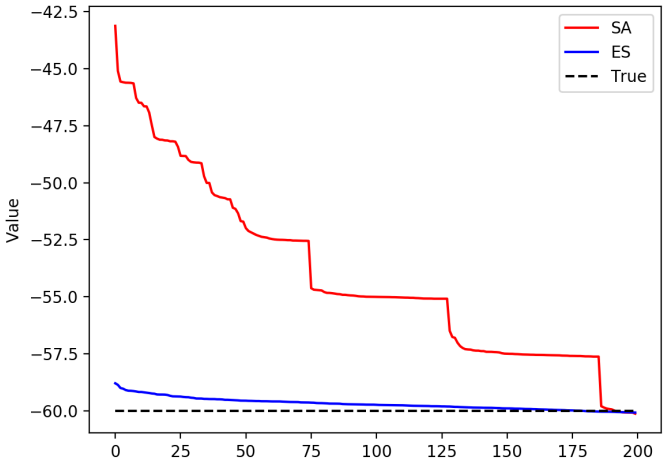
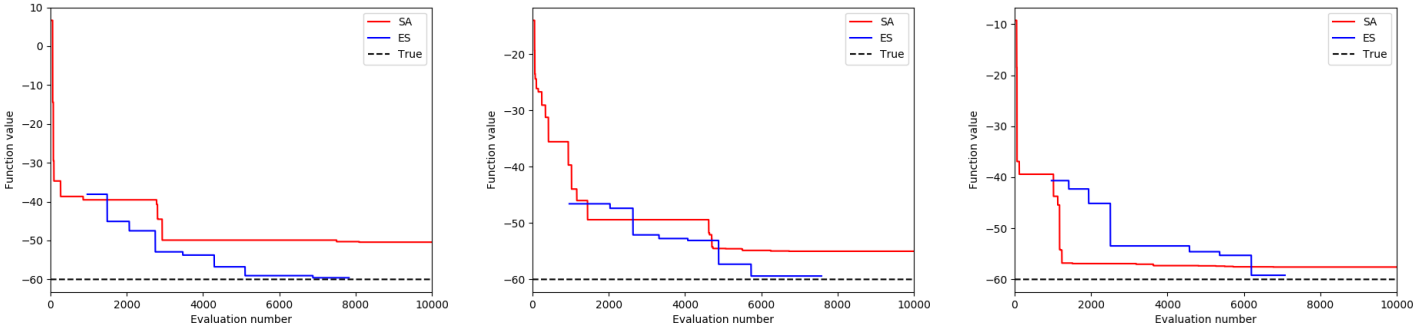


Figure 6: **Simulated Annealing** vs **Evolution Strategies** reliability

It is clear from the figure above that **Evolution Strategies** is a much more reliable algorithm than the **Simulated Annealing**.

4.2 Efficiency

The efficiency of both algorithms was measured by running the algorithm using the same seed and looking at how fast the algorithm convergences to a global minimum. By looking at the figures below it is observed that most of the time the **Evolution Strategies** algorithm will converge faster however by using some particular seed the **Simulated Annealing** algorithm converges faster.



(a) Seed 1 (b) Seed 2 (c) Seed 3

Figure 7: **Simulated Annealing** vs **Evolution Strategy** efficiency

5 Appendix

5.1 Appendix A: Coding

5.1.1 Simulated Annealing

5.1.2 Evolution Strategies

```python
import numpy as np
from Function import Common

class SimulatedAnnealing(Common):
 """
    Contains all the requirements to run a Simulated Annealing
    Optimisation with different strategies:
    -rotation strategy
    -selection strategy
    """

    def __init__(self, dim=5, L_k=1000, temp_adapt=True, random_restart=True):
        super(SimulatedAnnealing, self).__init__()
        self.temp_adapt, self.random_restart = temp_adapt, random_restart
        self.dimension = dim
        """Generator parameters"""
        self.alpha = 0.1
        self.omega = 2.1
        """Temperature parameters"""
        self.initial_search = 50  # fixed
        self.L_k = L_k
        self.accepted_min = int(L_k*0.6)
        """Restart acceptance ratio"""
        self.ratio = 0.01

        # Algorithm values to save
        self.best_solution = None
        self.best_value = float("inf")
        self.store_accepted_values = []
        self.archive_temp = []
        self.restart = False
        self.end = False
        self.temp_values = []
        self.new_best = False

    def run(self, evaluations=10000, seed=None):
        """ Main function that runs the algorithm"""

        np.random.seed(seed) # Set the seed for random generator
        solution = np.random.uniform(-1, 1, self.dimension) #intial solution
        value = self.func(solution*2)
        self.archive_solution(solution, value)
        """Initialisation"""
        D = np.eye(self.dimension)
        temp = self.initial_search(D)
```

```
        trials, accepted = 0, 0 # Initialise parameters

        """Start of main loop"""
        for identity in range(self.initial_search, evaluations+1):
            new_solution, delta = self.generate_solution(solution, D)
            new_value = self.func(new_solution*2)
            trials += 1
            if new_value > value: # Check if inceasing solution is accepted
                prob = self.probability(new_value-value, temp, delta)
                if prob >= np.random.rand(1):
                    solution, value = new_solution, new_value
                    D = self.update_generator(D, delta)
                    self.archive_solution(solution, value)
                    accepted += 1

            else: # All decreasing solution accepted
                solution, value = new_solution, new_value
                D = self.update_generator(D, delta)
                self.archive_solution(solution, value)
                accepted += 1

            """Check if restart is required"""

            if accepted >= self.accepted_min or trials >= self.L_k:
                if not self.new_best and accepted/trials < self.ratio:
                    if self.random_restart:
                        self.restart = True
                        solution = np.random.uniform(-1, 1, self.dimension)
                    else:

                        self.restart = True
                        solution = self.best_solution/2
                    value = self.func(solution*2)
                    trials, accepted = 0, 0
                    continue
                temp, trials, accepted = self._update_temperature(temp)


        self._archive.append(self._archive_temp)

        return self.best_value, self.best_solution

    def archive_solution(self, solution, value):
        """Update storage"""
        if self.restart:
            self.store_accepted_values.append(self.archive_temp)
            self.restart = False
```

```
            self.archive_temp = []

        self.archive_temp.append((solution*2).tolist())

        if value < self.best_value:
            self.best_solution = solution*2
            self.best_value = value
            self.new_best = True
        self.temp_values.append(value)

    def generate_solution(self, x, D):
        """Generate solution"""
        delta = D.dot(np.random.uniform(-1, 1, self.dimension))
        while not self.check_solution((x + delta)*2):

            delta = D.dot(np.random.uniform(-1, 1, self.dimension))
        return x + delta, delta

    def update_generator(self, D, delta):
        """Update the solution generator"""
        R = np.diag(abs(delta))
        D = (1-self.alpha) * D + self.alpha * self.omega * R
        return D

    def update_temperature(self, temp):
        """Update the temperature """
        if not self.temp_adapt:
            temp_param = 0.95
        elif len(self.temp_values) < 2:
            temp_param = 0.5
        else:
            temp_param = max(0.5, np.exp(-0.7*temp/np.std(self.temp_values)))
        self.temp_values = []
        self.new_best = False
        return temp_param*temp, 0, 0

    def probability(self, delta, temp, delta):
        """Calculate the acceptance probability"""
        step_size = np.sqrt(np.sum(np.square(delta)))
        return np.exp(-delta/(temp*step_size))

    def initial_search(self, D):
        """Perform the initial search"""
        solution = np.random.uniform(-1, 1, self.dimension)
        values = [self.func(solution*2)]
        for i in range(self.initial_search):
            solution, delta = self.generate_solution(solution, D)
```

```
            D = self.update_generator(D, delta)
            values.append(self.func(solution*2))
        return np.std(values)
```

```python
import numpy as np
from Function import Common

class EvolutionStrategies(Common):
    """
    Contains all the requirements to run an Evolution Strategies
    Optimisation with different strategies:
    -rotation strategy
    -selection strategy
    """


    def __init__(self, dim=5, population_size=1000, rot=True, elitist=False,
                 stop=False):
        super(EvolutionStrategies, self).__init__()

        """ General Parameters"""
        self.rot, self.elitist = rot, elitist
        self.dimension = dim
        """Selection parameters"""
        self.population_size = population_size
        self.parents_size = int(population_size/7)
        """Mutation parameters"""
        self.tau = 1 / np.sqrt(2*np.sqrt(dim))
        self.tau_dash = 1 / np.sqrt(2*dim)
        self.beta = 0.0873


        self.best_solution = None
        self.best_value = float("inf")
        self.archive_generation = []
        self.current_generation =[]

        """Stopping criteria"""
        if stop:
            self.stop = 15*dim
        else:
            self.stop = 0

        if rot:
            self.rot_dim = int(dim*(dim-1)/2)

    def run(self, evaluations=10000, seed=None):
        """ Main function that runs the algorithm"""

        np.random.seed(seed) # Set the seed for the random generator
        initial = []
```

```
        for i in range(self.population_size):
            if self.rot:
                initial.append(np.concatenate(
                    (np.random.uniform(-2, 2, self.dimension),
                     np.random.rand(self.dimension),
                     np.random.uniform(-np.pi, np.pi, self.rot_dim))))
            else:
                initial.append(np.concatenate(
                    (np.random.uniform(-2, 2, self.dimension),
                     np.random.rand(self.dimension))))
        self.current_generation.extend(initial)
        initial_assesments = self.assess(initial)
        parents, parent_assesments, stop = self.select(initial_assesments)
        evals_done = self.population_size
        self.archive_generation.append(self.current_generation)

        """Main loop"""
        while evals <= evaluations-self.population_size:
            self.current_generation=[]
            kids = self.recombine(parents)
            kids = self.mutate(kids)
            self.current_generation.extend(kids)
            kids_assesments = self.assess(kids)
            evals += len(kids)
            self.archive_generation.append(self.current_generation)
            if self.elitist:
                parents, parent_assesments, stop = self.select(
                    kids_assesments+parent_assesments)
            else:
                parents, parent_assesments, stop = self.select(
                    kids_assesments)

            if stop:
                return evals_done, None

        return self.best_value, self.best_solution

    def recombine(self, population):
        """Recombination part"""
        new_population = []
        for i in range(self.population_size):
            """Choosing the parents"""
            [i1, i2] = np.random.choice(len(population), 2)
            individual_length = len(population[0])
            kid = np.empty(individual_length)
            """Discrete recombination for control variables"""
            for j in range(self.dimension):
```

```
                choice = np.random.randint(2)
                kid[j] = (choice*population[i1][j] +
                          (1-choice)*population[i2][j])
            """Intermediate recombination for strategy variables"""
            for j in range(self.dimension, individual_length):
                choice = np.random.rand()
                kid[j] = (choice*population[i1][j] +
                          (1-choice)*population[i2][j])
            new_population.append(kid)
        return new_population

    def mutate(self, population):
        """Mutation part"""
        new_population = []
        dim = self.dimension
        if self.rot:
            dim2 = self.rot_dim

        for a in population:
            n = np.empty(len(a))
            """Standard deviations"""
            n[dim:dim*2] = a[dim:dim*2] * np.exp(
                              self.tau_dash * np.random.normal() +
                              self.tau * np.random.normal(size=dim))
            """Rotations"""
            if self.rot:
                """Matrix needs to be positive semidefinite for inversal so different methods
                are used to transform the matrix into the required form"""
                for i in range(1000):
                    n[-dim2:] = (a[-dim2:] +
                                 self.beta * np.random.normal(size=dim2))
                    cov = self.covariance_matrix(n)
                    if cov is not None:
                        break
                else:
                    cov = self.covariance_matrix(n, detect_err=False)
            else:
                cov = self.covariance_matrix(n)
            """Control variables"""
            mean = np.zeros(dim)
            n[:dim] = a[:dim] + np.random.multivariate_normal(mean, cov)

            new_population.append(n)
        return new_population

    def covariance_matrix(self, solution, detect_err=True):
        """Build a covariance matrix from the solution"""
```

```
        dim = self.dimension
        """If no rotation strategy is used, a simply a diagonal with the variance will be built"""
        var = np.square(solution[dim:dim*2])
        if not self.rot:
            return np.diag(var)
        """If the rotation strategy is selected, a symmetric matrix is built"""
        A = np.zeros((dim, dim))
        rot = solution[-self.rot_dim:]
        idx = 0
        for i in range(dim):
            for j in range(i):
                A[i, j] = 0.5 * (var[i] - var[j]) * np.tan(2*rot[idx])
                idx += 1
            A[i, i] = var[i]
        A = (np.maximum(A, A.T))
        """Try to make the matrix positive definite"""
        if detect_err and not np.all(np.linalg.eigvals(A) > 0):
            return None
        return np.linalg.inv(A)

    def assess(self, population):
        """Assement of the value used for the current generation"""
        assessments = []
        for idx, individual in enumerate(population):
            value = self.func(individual[:self.dimension])
            if value is not None:
                assessments.append((individual, value))
        return assessments

    def select(self, assessments):
        """Selection part of the algorithm"""
        val = sorted(assessments, key=lambda c: c[1])
        if self.stop != 0:
            terminate = abs(val[-1][1] - val[0][1]) <= self.stop
        else:
            terminate = False

        if val[0][1] < self.best_value:
            self.best_value = val[0][1]
            self.best_solution = val[0][0][:self.dimension]
        selected = []
        for i in range(self.parents_size):
            if i == len(val):
                break
            selected.append(val[i][0])
        return selected, val[:self.parents_size], terminate
```

References

[1] Andrej KarpathyTim SalimansJonathan HoPeter ChenIlya SutskeverJohn SchulmanGreg BrockmanSzymon Sidor. Evolution strategies as a scalable alternative to reinforcement learning. https://openai.com/blog/evolution-strategies, march 24, 2017. [Online; accessed 4-January-2020].

[2] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[3] Dr Geoff Parks. Simulated annealing. moodle Website, 7 January 2020. [Online; accessed 7-January-2020].

[4] Geoffrey Thomas Parks. An intelligent stochastic optimization routine for nuclear fuel cycle design. *Nuclear Technology*, 89(2):233–246, 1990.

[5] Thomas Back, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*, volume 2. Morgan Kaufmann Publishers San Mateo, CA, 1991.

[6] Dr Geoff Parks. Evolution strategies. moodle Website, 7 January 2020. [Online; accessed 7-January-2020].