NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Implementation of the Sine Cosine Algorithm and its variants for solving the tension compression spring design problem

Aris K. Tsilifonis

Supervisor: **Panagiotis Stamatopoulos, Assistant Professor**

ATHENS

JUNE 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση του αλγόριθμου ημιτόνου συνημίτονου και των παραλλαγών του για την επίλυση του προβλήματος σχεδιασμού τάσης ελατηρίου

Άρης Κ. Τσιλιφώνης

Επιβλέπων: **Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής**

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2023

BSc THESIS

Implementation of the sine cosine algorithm and its variants for solving the tension compression spring design problem

Aris K. Tsilifonis

S.N.: 1115201700170

Supervisor: **Panagiotis Stamatopoulos, Assistant Professor**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση του αλγόριθμου ημιτόνου συνημίτονου και των παραλλαγών του για την επίλυση του προβλήματος σχεδιασμού τάσης ελατηρίου

Άρης Κ. Τσιλιφώνης

Α.Μ.: 1115201700170

Επιβλέπων : **Παναγιώτης Σταματόπουλος**, Επίκουρος Καθηγητής

ABSTRACT

The Sine and Cosine Algorithm was created by Seyedali Mirjalili in 2015. It uses sine and cosine to solve various optimisation problems precisely. It belongs to a category of metaheuristics, which includes population-based strategies for obtaining the optimal result by mimicking natural phenomena. This thesis elaborates on a wide variety of its mutants. Specifically, fuzzy, chaotic, opposite-based-learning, greedy levy flight and adaptive multi-objective aquila are some of the variants the work focuses on.

This work is based on both theoretical and practical aspects of the algorithm. First, tests of efficiency were pursued on multiple benchmark functions. The research on the topic was expanded by the solution of a widely known engineering problem, the tension/compression spring design. It can be observed that the algorithm has relevance to various engineering, mathematical and medical issues when other deterministic ways fail. Many variants of the procedure were introduced to balance its weaknesses. Finally, diagrams are presented to improve our understanding of the SCA's accuracy.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: Swarm Intelligence, optimisation, sine cosine, mutation, spring design problem

ΠΕΡΙΛΗΨΗ

Ο αλγόριθμος ημιτόνου και συνημίτονου εφευρέθηκε από τον Mirjalili το 2016. Χρησιμοποιεί τις συναρτήσεις ημιτόνου και συνημίτονου για να επιλύσει ένα μεγάλο εύρος προβλημάτων βελτιστοποίησης. Ανήκει σε μια κατηγορία μεταευρετικών διαδικασιών, που περιλαμβάνει στρατηγικές βασισμένες σε πληθυσμό, για επίτευξη βέλτιστου αποτελέσματος μιμούμενο φαινόμενα στη φύση. Έπειτα, έγινε εμβάθυνση σε ένα μεγάλο εύρος παραλλαγών του αλγορίθμου. Ειδικότερα, ασαφής, χαοτικός, βασισμένος σε αντίθετη μάθηση, άπληστος levy, προσαρμοστικός και πολλαπλών στόχων aquila είναι κάποιες από τις μεταλλάξεις του αλγορίθμου που βασίστηκε η εργασία και βελτιώνουν την απόδοση του σημαντικά.

Η εργασία είναι στηριγμένη τόσο στο θεωρητικό όσο και στο πρακτικό κομμάτι του αλγορίθμου καθώς επιδιώχθηκε να ελεγχτεί η αποδοτικότητα του με πολλαπλές συναρτήσεις κριτηρίου. Επεκτείνεται η έρευνα στο αντικείμενο επιλύοντας ένα ευρέως γνωστό πρόβλημα μηχανικής, του σχεδιασμού τάσης ελατηρίου. Παρατηρείται ότι ο αλγόριθμος έχει εφαρμογή σε ποικιλία μηχανικών, μαθηματικών και ιατρικών θεμάτων. Είναι αντιληπτό ότι βρίσκει λύση εκεί που άλλες ντετερμινιστικές διαδικασίες δεν μπορούν να εφαρμοστούν. Πολλές παραλλαγές του αλγορίθμου ημιτόνου συνημίτονου έχουν εμφανιστεί για να ισορροπήσουν τις αδυναμίες του. Τέλος, παρουσιάζονται διαγράμματα για υπάρχει καλύτερη αντίληψη της απόδοσης του SCA.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: αλγόριθμος σμήνους, βελτιστοποίηση, παραλλαγές, ημίτονο συνημίτονο, σχεδιασμός τάσης ελατηρίου

Contents

1. INTRODUCTION 13

2. THEORY 14

2.1 Sine Cosine Algorithm 14

2.2 Fuzzy Sine Cosine Algorithm 19

2.3 Adaptive Chaotic Sine Cosine Algorithm 21

2.4 Aquila Algorithm 25

2.5 General SCA real-life applications 31

3. TENSION COMPRESSION SPRING DESIGN 32

3.1 Definition 32

3.2 Application of SCA 35

4. METHODOLOGY 36

5. METRICS 38

6. RESULTS AND DISCUSSION 44

6.1 Minimum Fitness Diagram (First 2000-6000 iterations) 45

6.2 Parameters 49

6.3 Basic SCA Results 51

6.3.1 Dimension=10 52

6.3.2 Dimension=100 54

6.3.3 Dimension=500 57

6.4 Chaotic SCA Tables 60

6.4.1 Dimension=10 60

6.4.2 Dimension=100 63

6.4.3 Dimension= 500 66

6.5 Modified Aquila Tables 69

6.5.1 Dimension=10 69

6.5.2 Dimension=100 72

6.5.3 Dimension= 500 75

6.6 Fuzzy SCA Tables 78

6.6.1 Dimension= 500 78

6.7 TCSD Problem Results 81

7. CONCLUSIONS AND FUTURE WORK.....84

TABLE OF TERMINOLOGY 85

ABBREVIATIONS ACRONYMS 86

ANNEX I: HARDWARE SPECS AND TOOLS.....87

REFERENCES 88

LIST OF FIGURES

Figure 1: Approximation of solution depending on the destination's radius 15
Figure 2: Decision of radius based on values of Sine and Cosine.....16
Figure 3: Decreasing amplitude of the trigonometric functions.....16
Figure 4: Bifurcation diagram.....21
Figure 5: Logistic map (first 100 terms) 23
Figure 6: Spring example.....31
Figure 7: Spring example 2.....34
Figure 8: Schwefels 7 3-dimension graph.....39
Figure 9: Griewank 3-dimension graph.....40
Figure 10: Ackley's 3-dimension graph.....41
Figure 11: Basic SCA's Min Fitness/iteration diagram (dimension=100) 45
Figure 12: Chaotic SCA's Min Fitness/iteration diagram (dimension=100).....45
Figure 13: Aquila's Min Fitness/iteration diagram (dimension=100) 46
Figure 14: Basic SCA's Min Fitness/iteration diagram (dimension=500) 46
Figure 15: Chaotic SCA's Min Fitness/iteration diagram (dimension=500).....47
Figure 16: Aquila's Min Fitness/iteration diagram (dimension=500) 47

LIST OF TABLES

Table 1: Parameters of experiment 49
Table 2: Parameters of experiment 2 50
Table 3: Basic SCA Results (dimension=10).....51
Table 4: Basic SCA Results (dimension=100).....54
Table 5: Basic SCA Results (dimension=500).....57
Table 6: Chaotic SCA Results (dimension=10) 60
Table 7: Chaotic SCA Results (dimension=100) 62
Table 8: Chaotic SCA Results (dimension=500) 66
Table 9: Modified Aquila Results (dimension=10) 69
Table 10: Modified Aquila Results (dimension=100).....72
Table 11: Modified Aquila Results (dimension=500).....75
Table 12: Fuzzy Sine Cosine Algorithm (dimension=500).....78
Table 13: Tension/Compression spring design Results.....81

LIST OF IMAGES

Image 1: Basic Sine Cosine Algorithm 17
Image 2: Fuzzy Sine Cosine Algorithm.....20
Image 3: Chaotic Sine Cosine Algorithm 24
Image 4: Modified Aquila Optimizer Algorithm.....27
Image 5: Quasi-opposite based learning Algorithm 28

1.INTRODUCTION

In this thesis, optimisation is at the centre of attention. Maximising or minimising the output is essential in reaching the desired results. In recent years, datasets have become quite large and more demanding for computer scientists to manipulate. As a result, problems are turning more complex for conventional algorithms to solve, and a generalised approach that could be implemented in almost every situation is invaluable.

Many artificial intelligence procedures cannot provide satisfying results because the agent is trapped inside a locally optimal solution, ignoring the one that is globally the best. Furthermore, the agent might be unable to approximate the optimal solution quickly enough, which can be concerning on many occasions. Modern systems need solutions to be on the spot and produced rapidly to operate correctly. Real-life challenges propose constraints which are nonlinear, constituting the computation of the gradient perplexing. As a result, derivative-free solutions with black box designs have gained a lot of ground. Also, by treating problems randomly and stochastically, scientists can broaden their horizons because they can face a larger set of problems than before. However, it is worth noting that one solution to every problem is impossible (No Free Lunch Theorem) [1]. Different algorithms could be more effective than others, depending on the situation. There is room for a lot of improvements in this research field.

This thesis focused on both the theoretical and practical aspects of the Sine Cosine Algorithm, a probabilistic problem-solving approach that capitalised on the trigonometric functions of sine and cosine to produce a satisfying outcome. However, even though it is a process that was invented recently, it lacks effectiveness in dealing with problems of higher dimensions. Because of this fact, a lot of improvements in these algorithms have attracted the interest of many scientists and will be presented in this thesis [7], [8]. More specifically, a procedure that mimics the behaviour of eagle species, named Aquila, will be thoroughly discussed in this research. Chaos, a phenomenon that takes place in real life numerous times, will be elaborated on in this analysis as it can enhance SCA's effect greatly. Other difficulties that can occur in complex problems are when the global optimum is located on the limits of the search space, when the function has a large number of global optima or when the optimal solution is located at a very steep point in space [1]. A thorough research regarding the efficiency of SCA on multiple test functions will verify the results of the widely known literature. These results will be compared with the ones of the other variants to have a complete view of the improvements in the basic algorithm. In addition, the research will be expanded on a mathematical engineering problem, the so-called tension-compression spring problem [11], [12]. The reader will be able to view how the constraints that this problem proposes can be applied to SCA as well as observe SCA's efficacy on it.

SCA is population-based, belonging to the swarm intelligence family of algorithms. This can augment its area of effect because more agents can try to approximate the global optima, as they can better explore the search space. Although population-based solutions need more function evaluations [1], this results in a higher chance of reaching the optimal solution. Regarding the test functions, both multimodal (multi-peaks) and unimodal (single peak) were taken into consideration as well as fixed dimension ones (CEC).

2.THEORY

2.1 Sine cosine Algorithm

The SCA population-based algorithm commonly attains results by setting values randomly to solutions in the initial stage. Then, by proper update rule, the algorithm will obtain the best value out of that set. Due to the stochastic approach of SCA, the solution is almost impossible to be reached in a single step [1]. Several updates need to occur so that SCA can converge on the best solution.

Probabilistic processes, which form the population of solutions, usually share some common characteristics. Their optimisation process is divided into two stages: exploration and exploitation. In the former stage, the solutions are changing substantially to explore a wider area of possible global optima [6]. In the latter one, SCA fluctuates around the found peak that could potentially lead to the desired value.

Below, the equations that the procedure is based on are illustrated:

$$x_i^{t+1} = \begin{cases} x_i^t + r1 * \sin(r2) * \left| r3P_i^t - x_i^t \right|, r4 < 0{,}5 \\ x_i^t + r1 * \cos(r2) * \left| r3P_i^t - x_i^t \right|, r4 \geq 0{,}5 \end{cases} \quad (1)$$

$$r_1 = a - \alpha \frac{t}{T},$$

Variable t equals the current number of iterations, while threshold T is the maximum number of iterations. The value of a is a constant (a=2 in this implementation). P_i^t, in the t-th iteration and i-th dimension, represents the optimal individual of the current population. r1, r2, r3 and r4 are random values defined by the inventor of SCA (Mirjali et al.). x_i^t shows the current element of the population in the t-th iteration and the i-th dimension. The individual solution in the t+1-th iteration is the result of the mathematical equation.

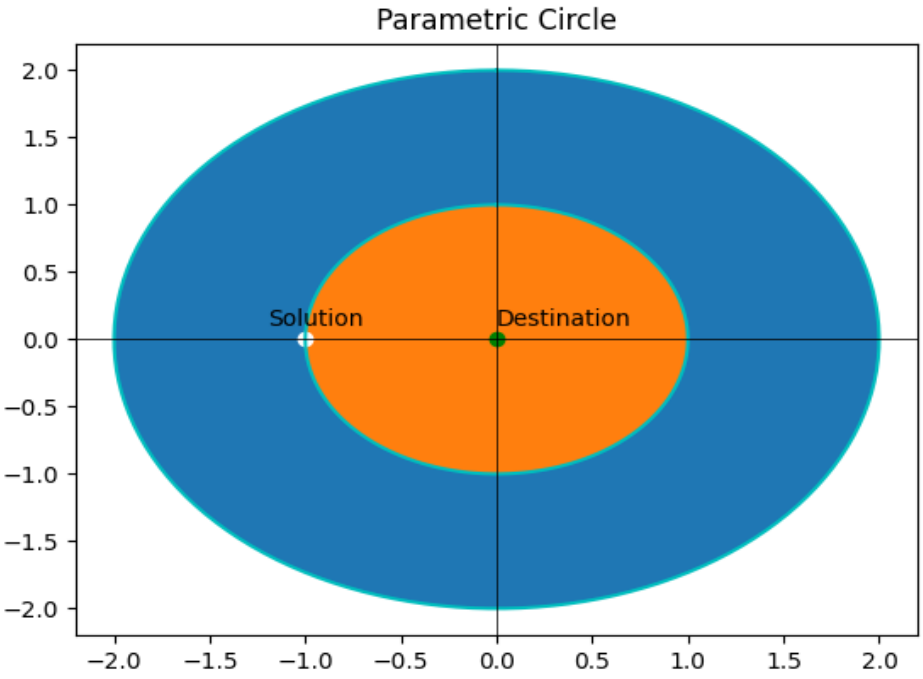

Figure 1: Approximation of solution depending on the destination radius.

The reader can picture the problem in a parametric circle. The solution is placed at a specific radius far from the optimal destination. By using the aforementioned update rule, the solution can either be placed towards the orange area (closer to the destination) or the blue area (outside the scope of the optimal solution).

By observing equation (1), someone can understand that sine and cosine play a major role in the final outcome. It should be made clear that when the ranges of trigonometric functions 2*sin(x) and 2*cos(x) are in the interval of [1, 2] or [-1, -2], the solution avoids the destination by moving away from it (blue area). These functions are based on the hypothesis that r1=2. On the other hand, when the trigonometric functions are limited in the range (-1, 1), then the swarm(population) will move towards the destination (orange area).

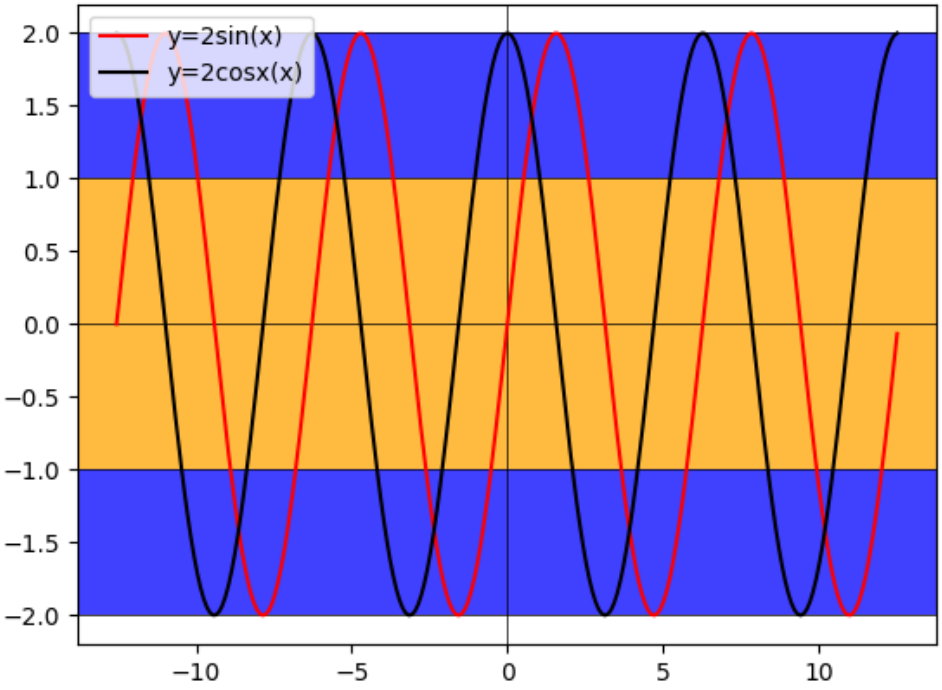

Figure 2: Decision of radius based on values of Sine and Cosine

SCA is fundamentally based on randomness. As a result, r1, r2, r3 and r4 play a critical role in the behaviour of the whole procedure. These random properties have different targets. Value r4 defines which equation the program will use (the upper or the lower bracket). It is distributed in the range [0, 1]. The role of r1 is to define the amplitude of sine and cosine.

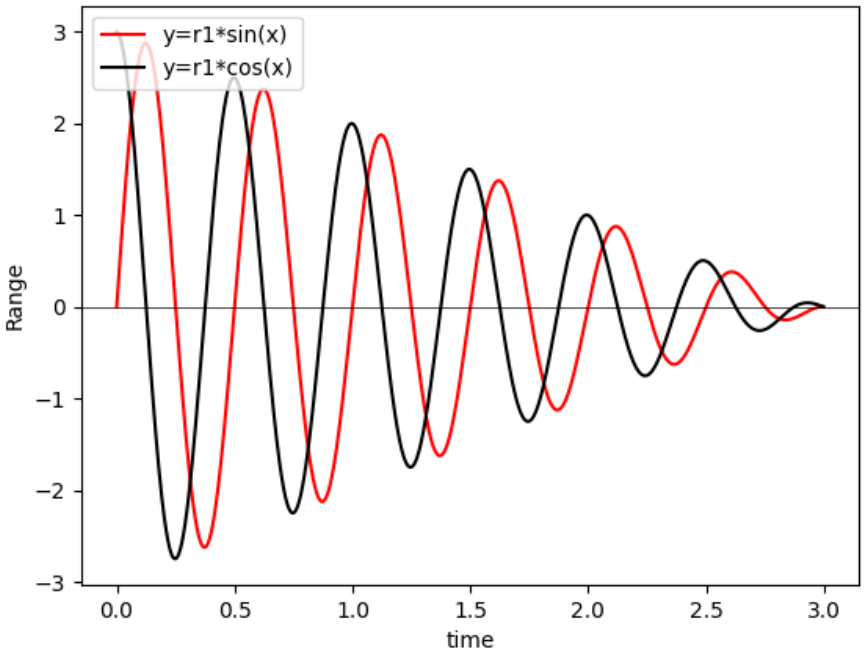

Figure 3: Decreasing amplitude of the trigonometric functions

As it is shown in equation (1), r1 is decreasing. Therefore, the range of sine and cosine will be reducing steadily (Figure (3)), which results in approaching closer to the destination. T defines the maximum number of iterations, and t is the iteration that happens currently. By decreasing r1, in the way it is presented in equation (1), the agent is able to shift smoothly rather than abruptly from exploration to the exploitation stage. In contradiction to that, the amplification of it can lead to an expanded distance between the solution and the destination. In addition, r2 affects the period of sine and cosine. It ranges between [0, 2π] and depending on the waveform, it can have either greater or lower values. For instance, sin (0) =0, cos (0) =1. r3, which belongs in the range [0, 2], defines how much the optimal solution will affect the outcome of the equation. r3>1 increases importance, and r3<1 lowers it. Even though the scenario that was depicted in Figure (2) considers two dimensions, it can be extended to further dimensions if the problem requires it. Figure (1) concerns the exploitation stage of the algorithm.

Pseudocode sums up all the theory that was mentioned previously. A set of agents are initialized randomly at the start, and the fitness of each element as well as the optimal individual is computed. On top of that, an evaluation of the objective function will happen afterwards. The agents are shifted from exploration to the exploitation stage at some point, which will result in better convergence to global optima.

Algorithm 1 *basic Sine Cosine Algorithm*

Initialize random values r_1, r_2, r_3 and r_4
for $i = 1$, i++, $i <= n$ **do**
    Initialize individuals randomly $X_i(i = 1, 2, ..., n)$
    Calculate the fitness value for each individual X_i
    Compute the optimal individual P based on the minimum fitness value
end for
while $t_{var} \leq T$ **do**
    Update random values r_1
    for $i = 1$, i++, $i <= n$ **do**
        for $j = 1$, j++, $j <= d$ **do**
            Update random values r_2, r_3 and r_4
            Calculate objective's function value X_{var} eq.(1)
            Update solutions X_i in each dimension
        end for
    end for
    for $i = 1$, i++, $i <= n$ **do**
        Calculate the fitness for each individual X_i
        Update P (Optimal individual)
    end for
end while
    return optimal individual P

Image 1: Basic Sine Cosine Algorithm

Regarding the complexity of SCA, it can be computed easily if it is divided into stages. One way to accomplish that is by considering as initial stage the first loop. Then, the second stage is the objective function computation, the while and the two following loops. The third stage is the calculation of the optimal individual. O(SCA) = 3*O(N) + T * O (N * D) + T* 2O (N) = (3 + 2T) *O(N) + T*O(N*D) where N is the size of the population and D is the dimension each solution.

In conclusion, the SCA forms solutions around optimal values. This creates a better chance of reaching a global optimal. By treating problems as black boxes (no adjustments on the algorithm according to the input), it can have numerous applications in different fields. Based on the original algorithm, many interesting variations have arisen from it.

2.2 Fuzzy Sine Cosine Algorithm

This modification was created to improve the design of PID controllers responsible for thermal heating [10]. Fuzziness is a mathematical concept where several elements in the set are assigned two different values of more instead of one. It is quite similar to SCA, but it differs from it after the computation of the objective's function value.

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,D} \\ \vdots & \ddots & \vdots \\ x_{NP,1} & \cdots & x_{NP,D} \end{pmatrix}$$

Suppose that there is a population of size N, and its element of it has dimension D. This matrix illustrates the updated solutions after SCA's computation as well. A certain mutation stage will be applied to the original SCA. More specifically, the program produces N number of random values, which represent the index of the dimension that will be altered for each solution.

$$r_{mut} = [r_{mut,1} \quad r_{mut,2} \cdots r_{mut,NP-1} \quad r_{mut,NP}] \text{ (2)}$$

Every r_{mut} is an integer in the range of [1, D]. For instance, consider a scenario that $r_{mut,1} = 2$. This means that the program will change the second dimension's value from the first element of the population randomly inside the domain space. The process will continue in the same way until the following vector is produced.

$$x_{mut} = [x_{mut,1} \quad x_{mut,2} \cdots x_{mut,NP-1} \quad x_{mut,NP}] \text{ (3)}$$

Afterwards, there will be two vectors, each one representing the same element of the population (fuzziness). Two different fitness values need to be calculated. One for the old vector of the element and the other for the new one.

$$f_{n1} = [f_{n1,1} \quad f_{n1,2} \cdots f_{n1,NP-1} \quad f_{n1,NP}] \, for \, x_{old} \text{ (4)}$$

$$f_{n2} = [f_{n2,1} \quad f_{n2,2} \cdots f_{n2,NP-1} \quad f_{n2,NP}] \, for \, x_{new} \text{ (5)}$$

The program holds the element of the population that has the better fitness out of the two.

After the calculation of the objective function, which is essentially the same as SCA's, the computations x's and r's follow. The program copies the old population to a new one to manipulate it properly. Subsequently, the program updates the population as described above and then proceeds by computing the optimal solution based on the fitness of the resulting population. It is a greedy and simple approach that improves the SCA on many occasions. The complexity of fuzzy SCA has the same order of magnitude as basic SCA's but is quite larger. O(Fuzzy-SCA) = (3 + 2T) *O(N) + T*O(N*D) + T*(O(N*D) + 2*O(N)) = (3 + 4T) *O(N) +2*T*O(N*D).

The operation that was described could be applied only to the optimal global individual instead of the whole population but affecting every element was thought to have a greater influence on the outcome of the algorithm. The following pseudocode shows the steps that the algorithm consists of:

Algorithm 1 .2 *Fuzzy Sine Cosine Algorithm*

Initialize random values r_1, r_2, r_3 and r_4
for $i = 1$, i++, $i <= n$ **do**
    Initialize individuals randomly $X_i(i = 1, 2, ..., n)$
    Calculate the fitness value for each individual X_i
    Compute the optimal individual P based on the minimum fitness value
end for
while $t_{var} \leq T$ **do**
    Update random values r_1
    for $i = 1$, i++, $i <= n$ **do**
        for $j = 1$, j++, $j <= d$ **do**
            Update random values r_2, r_3 and r_4
            Calculate objective's function value X_{var} eq.(1)
            Update solutions X_i in each dimension
        end for
    end for
    for $i = 1$, i++, $i <= n$ **do**
        Compute r_i based on eq.(2)
        Compute x_i based on eq.(3)
        for $j = 1$, j++, $j <= d$ **do**
            Copy old population to a new one x_i'(of same size)
        end for
    end for
    for $i = 1$, i++, $i <= n$ **do**
        Update new population based on eq.(4)
        if $fitness(x_{new}) \leq fitness(x_{old})$ **then**
            Update old individual with a new one
        end if
    end for
    for $i = 1$, i++, $i <=^{1} n$ **do**
        Calculate the fitness for each individual X_i
        Update P (Optimal individual)
    end for
end while
    return optimal individual P

Image 2: Fuzzy Sine Cosine Algorithm

2.3 Adaptive Chaotic Sine Cosine Algorithm

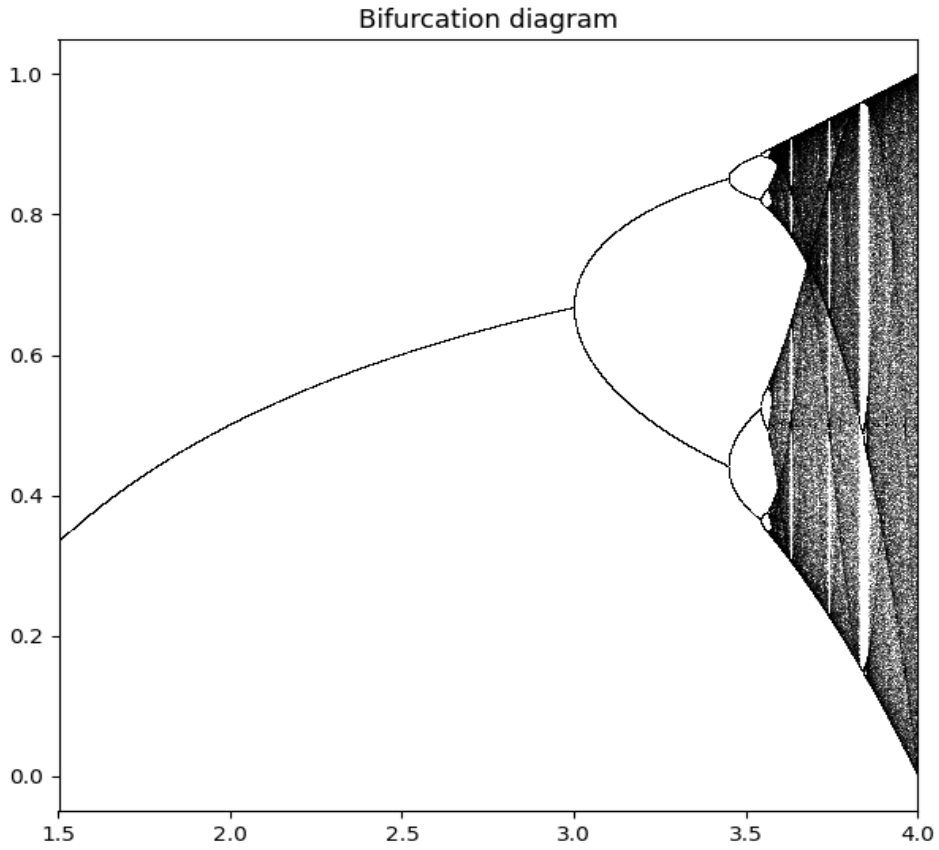

Figure 4: Bifurcation diagram

Chaos refers to situations which are primarily predictable. Chaotic systems are predictable at initial stages, and they turn random after a while. A classic example is the butterfly effect. Just a small change in a deterministic nonlinear system can result in significant change at a later stage. The bifurcation diagram pictures the resulting population after many iterations of the logistic map in equation 8 as the value of a is rising. When a is higher, the population increases. When a surpasses 3, the population splits into two parts. The population in this situation oscillates instead of stabilizing. As the a is becoming larger, the period is doubling. This means that the population oscillates among four values instead of two when a was 3. After a is increasing further, around 3.5, the period increases in an unpredictable manner. This phenomenon happens in the grey zone of the diagram above (after a>3.5). However, not all values between 3.5 and 4 lead to chaos. There are some white areas across that domain, meaning that chaos and order are swapping places. That is, by definition, chaos [14].

It is worth mentioning that the bifurcation diagram is part of an over fractal, which are very complicated geometric shapes and is impossible to describe with Euclidean geometry. The ratio, when those bifurcations occur, approaches a constant value and is commonly known as the Feigenbaum constant. It is a number close to 4.669. This constant is ubiquitous as it has been found in many equations which are attempting to simulate events in nature. Simple equations can have very complex behaviours. Also, scientists experimented with fibrillation disease and found that by applying an electrical

shot to the heart at times based on chaos theory, the heart can pump blood in a periodic way again [20].

Since SCA is entirely random, chaos can promote its global search ability by traversing the whole area of search space more effectively. Sine Cosine Algorithm experience a drop of diversity at the later stages, something that leads to slow convergence and locally oriented optimization when dealing with complex problems. To prevent this, Chaotic Sine Cosine uses adaptation and Chaotic exploitation. Firstly, an adaptive transformation parameter boosts the algorithm by allowing the switch between the exploration and exploitation stage to be more balanced [6].

$$r_1 = 4 * \left(1 - \frac{t}{T}\right) * \left(1 - 2^{\left(\frac{t}{T}-1\right)}\right) \quad (6)$$

In the original SCA algorithm, r1 was declining in a linear way. In the proposed equation (6), r1 is decreasing in a smoother way compared to SCA's case, allowing the switch from exploration to the exploitation stage to happen in an accurate time [6]. T is the total number of iterations. The current t and the constant T will be utilized in the calculation of r1. On SCA's occasion, r1 implementation risked premature convergence. Therefore, a more significant peak could be missed, constituting the algorithm's inefficiency.

The Lambda constant, which is used below, can be computed in the following way:

$$\lambda = \frac{T-t+1}{T} \quad (7)$$

The Chaotic algorithm augments the search for the solution localities.

To be more specific, a Chaotic logistic map:

$$y_{k+1} = a * y_k(1 - y_k) \quad (8)$$

is proposed by a research paper to render the population more accurate. New individuals are created randomly according to the Chaotic sequence that is shown above. When $a = 4, y_1 \notin \{0.25,0.5,0.75,1\}$ this equation is a Chaotic system. A plot of the equation is depicted below. You will observe that there is no pattern in that graph. Every peak is different, which helps generate chaos and unpredictability for SCA.

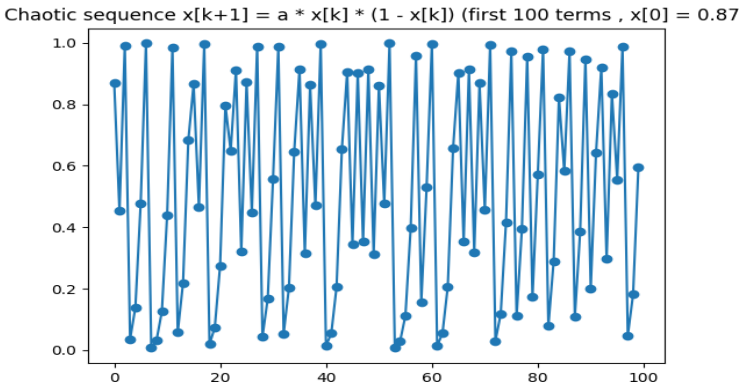

Figure 5: Logistic map (first 100 terms)

Each term of this sequence will be used by another equation to form the new population.

The new population is computed by:

$$V = (1 - \lambda) * X_{best} + \lambda * (lb + y_k * (ub - lb)) \quad (9)$$

Ub and lb represent the upper and lower bound, respectively. X_{best} shows the optimal solution for this specific iteration, t.

Chaotic Sine Cosine is distinguished from the basic SCA due to two reasons. Firstly, as shown on the eighth line of the following image, r1 is updated in a new way (equation (6)). Secondly, as illustrated in lines 22 to 28, a new Chaotic population is created (based on equation (9)). From that population, the program chooses the element with the best fitness and compares it with the optimal individual from SCA's algorithm. If the new individual has a better score than the previous one, it is assigned as the new global optimal. Otherwise, nothing is done. Lambda and y sequence play a critical role in the resulting V population. Regarding the complexity, it is equivalent to SCA's apart from the step that was mentioned.

O (Chaotic SCA) = 3*O(N) + T * O (N * D) + T*2 O (N) +T*O(N*D) +2T(O*N) = (3+4T)*O(N) +2*TO(N*D). The algorithm has a complexity of the same order of magnitude as SCA but a larger one.

Algorithm 2 *Chaotic Sine Cosine Algorithm*

Initialize random values r_1, r_2, r_3 and r_4
for $i = 1$, i++, $i <= n$ **do**
    Initialize individuals randomly $X_i(i = 1, 2, ..., n)$
    Calculate the fitness value for each individual X_i
    Compute the optimal individual P based on the minimum fitness value
end for
while $t_{var} \leq T$ **do**
    Update random values r_1 based on eq.(6)
    for $i = 1$, i++, $i <= n$ **do**
        for $j = 1$, j++, $j <= d$ **do**
            Update random values r_2, r_3 and r_4
            Calculate objective's function value X_{var} eq.(1)
            Update solutions X_i in each dimension
        end for
    end for
    for $i = 1$, i++, $i <= n$ **do**
        Calculate the fitness for each individual X_i
        Update P (Optimal individual)
    end for
    Update λ based on eq.(7)
    Produce chaotic first term based on eq.(8)
    for $i = 1$, i++, $i <= n$ **do**
        for $j = 1$, j++, $j <= d$ **do**
            Calculate V based on eq.(9)
            Update V
        end for
        Update chaotic sequence based on eq.(8)
    end for
    for $i = 1$, i++, $i <= n$ **do**
        Calculate the fitness for each individual V_i
        Compute the optimal individual V_{best}
    end for
    if $fitness(V_{best}) \leq fitness(P)$ **then**
        Update P with V_{best}
    end if
end while
    return optimal individual P

Image 3: Chaotic Sine Cosine Algorithm

2.4 Modified Aquila Optimizer

Since the thesis elaborated thoroughly on Sine Cosine Algorithm, a question was raised on how this process can be improved further. Aquila optimizer gave a definite answer to this problem [2]. To understand it thoroughly, the reader should have knowledge of some of its properties. Firstly, levy distribution plays a critical role in the whole procedure.

Levy number is the quotient of two real numbers, u and v. They both belong in Normal distributions. The first parameter μ is the mean value of the distribution, and the other σ indicates the variance. These numbers equal:

$$levy = \frac{u}{|v|^{\frac{1}{\beta}}}$$

$$u \sim N(0, \sigma_u^2)$$

$$v \sim N(0, \sigma_v^2)$$

$$\sigma_u = 1 \; and \; \beta = \frac{3}{2} and \; \sigma_v \; can \; be \; calculated \; based \; on:$$

$$\sigma_u = \frac{\Gamma(1+\beta) * sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) * \beta * 2^{\frac{(\beta-1)}{2}}}$$

Levy flights are random orbits which have been proven very effective in spotting the areas in the search space which are candidates for optimal global values.

Another vital aspect that the reader should be familiar with is opposite-based learning. Before introducing the aquila algorithm, the math behind this topic will be presented. Consider that there is a population of size n and the range of the domain space having an upper bound u and a lower bound l [3].

$$x = [\, x_1, x_2, \ldots, x_n] \in \mathbb{R}^N \;, \;\; x_1, x_2, \ldots, x_n \in \mathbb{R}$$

The opposite of the original number for the specific range [u, l] can be calculated with the following equation.

$$\bar{x} = u + l - x$$

It should be noted that this thought process can be extended for multiple dimensions where each of them has a specified domain space, as shown below:

$$x_j \in \left[\, l_j, u_j \right] \;\; \bar{x} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n]$$

$$\bar{x}_j = u_j + \; l_j - x, \; j = 1, 2, \ldots, n$$

Opposite numbers have illustrated great effectiveness in improving algorithms because initial numbers may have led the processes to a solution that could not yield desired results. By computing the opposite of the number, the outcome of the procedure could be entirely different as the opposite direction plays a critical role in obtaining optimal global values. However, the research was not limited only to opposite numbers. The paper that was found [2] has sparked great interest among scientists and has provided a radical idea on how these opposite numbers can be utilized.

The procedure that enhances a lot the aforementioned idea is the quasi-OBL (opposite-based learning). It states that for each element of the population and for every dimension of it, the program initially computes the opposite number in that range. The innovation is expressed by the next step. A threshold named D determines the two ways that quasi-OBL products can be assigned. It has to be made clear that this threshold can be used for multiple dimensions since D can be affected by the ub and lb parameters. lb and ub pair can be different for each dimension, but the same pair is used for each one of them in this implementation. This step was integrated into Aquila optimizers since it proved effective in finding better candidates for globally best values or elements with improved fitness values compared to the original OBL. The two mentioned topics put the foundation for simulating the behaviour of aquila eagles in computers. Also, quasi-OBL was applied not only to the initialization step of the population but after each of the update rules in the primary process.

Algorithm 4 *Aquila Algorithm*

Initialize random values r_1, r_2, r_3 and r_4
for $i = 1$, i++, $i <= n$ **do**
    Initialize individuals randomly $X_i (i = 1, 2, ..., n)$
    Compute QOBL quasi - opposites $x_{i,j}^{qo}$ based on Algorithm 3
    Calculate the fitness value for each individual X_i
    Compute the optimal individual P based
    on the minimum fitness value
end for
while $t_{var} \leq T$ **do**
    for $i = 1$, i++, $i <= n$ **do**
        for $j = 1$, j++, $j <= d$ **do**
            if $t \leq \frac{2}{3} * T$ **then**
                if $rand \leq 0.5$ **then**
                    Expand exploration eq.(10)
                    Compute QOBL
                else
                    Narrow exploration eq.(11)
                    Compute QOBL
                end if
            else
                if $rand \leq 0.5$ **then**
                    Expand exploitation eq.(12)
                    Compute QOBL
                else
                    Narrow exploitation eq.(13)
                    Compute QOBL
                end if
            end if
        end for
        if $fitness(x_{i,j}^{qo}) \leq fitness(x_{i,j})$ **then**
            Swap $x_{i,j}$ with $x_{i,j}^{qo}$
        end if
    end for
    Calculate the fitness value for each $x_{i,j}^{qo}$
    Compute the optimal individual P based
    if $fitness(X_{best}) \leq fitness(P)$ **then**
        Update P with X_{best}
    end if
end while
    return optimal individual P

1

Image 4: Modified Aquila Optimizer Algorithm

It must be noted that after every update of 10, 11, 12, 13 equations as well as the quasi-OBL rule, the program attempts to limit the solution to the domain space defined by the user.

Algorithm 3 *quasi - opposite based learning*

1: Initialize $D = (lb + ub)/2$
   //Each dimension has the same bound
2: **for** $i = 1, i{+}{+}, i <= n$ **do**
3: **for** $j = 1, j{+}{+}, j <= d$ **do**
4: $x^o_{i,j} = lb + ub - x_{i,j}$
5: **if** $x_{i,j} \leq D$ **then**
6: $x^{q_o}_{i,j} = D + (x^o_{i,j} - D) * rand()$
7: **else**
8: $x^{q_o}_{i,j} = x^o_{i,j} + (D - x^o_{i,j}) * rand()$
9: **end if**
10: **end for**
11: **end for**
12: Return $x^{q_o}_{i,j}$ //quasi-opposite

Image 5: Quasi-Opposite based learning Algorithm

Quasi-opposites can form a better baseline than the other algorithms because the starting point has a better fitness value than before. This means that there is a higher chance of the optimal solution being found by the program since the starting point is probably closer to that point than before. The same applies after each update step (equations 10-13) because these opposite solutions could potentially be used in later iterations, producing a better outcome at the end. By inserting the randomization step after the opposition rule (line 4 algorithm 3), a greater efficiency was attained, as will be presented in the results section.

The complexity of the algorithm involves three stages:

Initialization, population update and computation of global optima. The initialization stage is almost the same as the previous algorithm except for the quasi-opposites' computation. O(quasi-OBL) = O(N*D).

So, O(initialization) = 3*O(N) + O(N*D)

O(update) = T*(O (N* D * N * N * D), O (global optima computation) = T*O(N). Program needs to loop through all values of the population to compute the fitness of every element and find the minimum one.

The update rule has greater complexity than the previous algorithms. Computation of mean value for its dimension requires O(N) time (in some of the equations 10,11,12,13). The other N*D product is the time to compute the fitness of one element of the population and to assign it to the individual solution if required (D iteration times). The total complexity of the program is: 3*O(N) + O(N*D) + T*(O (N*N*N*D * D)) + T*O(N).

The Aquila algorithm attempts to imitate the behaviour of a kind of eagle named Aquila. After the initialization stage and quasi-opposite computation, the update stage follows. This simulates four different tactics of the hunting bird to catch its prey. Which hunting type will be involved in the process is determined by a random real number inside [0, 1]. If it exceeds a certain number equal to two-thirds of the threshold of iterations, one

specific branch will be chosen by the program. Otherwise, another one. Inside the if then else in line 11 of them algorithm (4), there is another if then else where the Aquila's update is implemented.

The first hunting type is simulated by the equation:

$$x_1^{t+1} = x_{best}^t * \left(1 - \frac{t}{T}\right) + (x_M^t - rand * X_{best}^t) \quad (10)$$

The Aquila is trying to find the best location to hunt and spot potential food resources. It could be interpreted as a broad exploration of the search space.

x_{best}^t, in the t-th iteration, is the best solution in iteration t, which is the current number of iterations. T is the total number of iterations. Rand is a random real value larger or equal to zero and less or equal to 1. Finally, x_M^t, in the t-th iteration, is the mean value of all the elements of the population (N elements) for every dimension. For example, if j=1, then the program computes the sum of all elements in the population for dimension=1.

Then it divides the sum by the size of the population to compute the mean value. This is the explanation for the equation:

$$x_M^t = \frac{1}{N}\sum_{i=1}^{N} x_i(t)\,, \forall\, j = 1, 2, \dots, D$$

This series is computed inside the double for loop, precisely in line 13 of the previous picture (algorithm (4)). The term (1 - t \ T) influences the level of exploration. A low t value provokes broader exploration, whilst a considerable value promotes a less significant one. Consequently, in the earlier stage, the algorithm will explore a larger area. Later, it will investigate a narrower one.

If the random number is more prominent than 0.5, equation (11) will be applied. It proposes an alternative way for the bird to fly, to surround prey. The circling way of flying is simulated by the levy number for the dimension proposed by the problem.

$$x_2^{t+1} = x_{best}^t * levy(D) + x_R^t + (\, y - x\,) * rand \quad (11)$$

x_R^t in the t-th iteration names a random solution from the whole population. Rand's number is the same as in equation (10). $levy(D)$ is the number from the distribution mentioned previously. This number must be multiplied by 0.01 to be scaled and remain accurate to the algorithm.

The circular flights of these creatures are transformed into code by the trigonometric equations of sine and cosine:

$$y = r_1 + U\, D_1 cos\, \left(-\omega\, D_1 + \frac{3\pi}{2}\right)$$

$$x = r_1 + U\, D_1 sin\, \left(-\omega\, D_1 + \frac{3\pi}{2}\right)$$

The Aquila process has adopted a practice from the original SCA, as the previous mathematical types verify it. U, ω values are pre-determined. The value of the former is 0.00565, and of the latter, 0.005. Remember that 3π/2 concerns the third quarter of the trigonometric circle and could be viewed as 270 degrees angle. D1 is a random integer having values from 1 to D (D is the maximum dimension). Finally, r1 is inside the [1, 20] domain.

Although expansion is a primary part of the process, exploitation is also crucial. This process unfolds in this way:

$$x_3^{t+1} = (x_{best}^t - x_M^t) * a - rand + ((ub - lb) * rand + lb) * \delta \quad (12)$$

After Aquila has spotted its prey during the exploration stage, it flies lower and slower to be able to catch it. The value in the next iteration is indicated by (t+1). α and δ are calibration parameters. Upper and lower bound ub, lb were defined before, as well as rand.

Genuine exploitation happens in equation (13). On this occasion, Aquilas are attempting to grab the prey by actions based on randomness. Narrow exploitation happens because the program is focusing on the solution entirely. The formal definition of this situation is presented:

$$x_4^{t+1} = QF + x_{best}^t - \left(G_1 * rand * x_{i,j}^t\right) - G_2 * levy(D) + rand * G_1 \quad (13)$$

In order to produce the updated solution t+1, several terms need to participate. x_{best}^t in the t-th iteration provides the best solution so far. About $x_{i,j}^t$, in t-th iteration, equals the current solution in iteration t of the i-th element of the population in the j-th dimension. G1 and G2 are responsible for describing in which way the Aquila acts. They are specified as such:

$$QF(t) = t^{\frac{2*rand()-1}{(1-T)^2}}$$

$$G_1 = 2 * rand() - 1$$

$$G_2 = 2 * (1 - \frac{t}{T})$$

A smooth switch from the exploration to the exploitation stage is succeeded by the illustrated quality function QF. The exponent of t is based on randomness, and the number of maximum iterations T. t is the current number of iterations. The motion

pattern of the birds, as they move from the source to the destination and capture the prey, is adjusted to code by those mathematical formulas. Aquilas are tracking their food and committing their attack. In a similar fashion, the program tracks the solution and extracts it in a relatively quick and effective manner, as the Aquila.

2.5 General real-life applications of SCA

Sine Cosine Algorithm has many exciting applications which have a significant impact. Scientists have been recorded using SCA to improve their chances of finding new planets in the universe. Moreover, they detected cancer in samples by clustering techniques with a variant of SCA. SCA classified samples better than before. It affected the engineering field too [21]. SCA can solve a lot of mathematical problems. This thesis is one of those examples. By optimizing materials that are utilized by engineers, such as spring, a breakthrough in machine performance can happen.

Another example is electronic and thermal controls (SCA Fuzzy) [10]. Photovoltaic power was improved using SCA because it optimized PID factors used in that system. SCA contributed to the optimal placement of cameras [22], and its efficiency was validated by comparing it to other genetic algorithms, such as PSO. The radical aspect of SCA expanded to feature selection. SCA achieved high classification success in many datasets and minimized the feature's size. In addition, it was applied to image processing. It achieved the image binarization of an Arabic document [23]. Its results were better than some other algorithms existing in the field. SCA was combined with Q-Learning to produce a new algorithm in the field of reinforcement learning [6]. The outcome of this mix involved spotting the best practices searching bottom up at runtime. Computer systems need to consider many parameters, like environments and variables, which render it difficult to test those systems. SCA plays a role in combinatorial testing by providing sets of parameters than can cover all the possible sets of them [24]. Finally, SCA contributed a lot to the optimization of medical diagnosis. Being able to recognize diseases, discover new potential drugs that can save lives, improving our understanding of the relationship between drugs and diseases are only some of the impacts of SCA on this field [25].

3.TENSION COMPRESSION SPRING PROBLEM

3.1 Definition

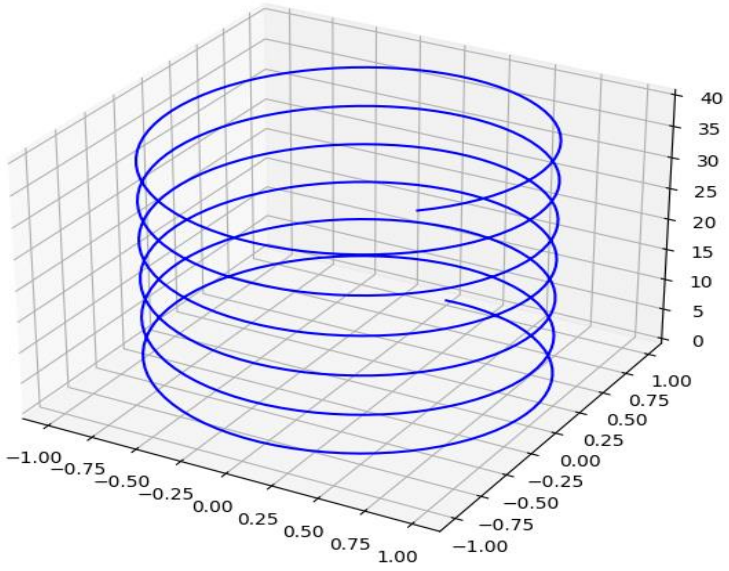

Figure 6:Spring example

$$\min f(x) = (x_3 + 2) * x_2 * x_1^2$$

(14)

Subject to:

$$q_{1(x)} = 1 - \frac{x_2^3 * x_3}{71785 * x_1^4} \leq 0$$

$$q_{2(x)} = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \leq 0$$

$$q_{3(x)} = 1 - \frac{140.45 * x_1}{x_2^2 * x_3} \leq 0$$

$$q_{4(x)} = \frac{(x_2 + x_1)}{1.5} - 1 \leq 0$$

The number of active coils is x_1 (the coils that are not closed), x_2 is the diameter of the spiral, and x_3 is the diameter of the wire. The domain space of those variables is:

$$2 \leq x_1 \leq 15\,,$$

$$0.25 \leq x_2 \leq 1.3\,,$$

$$0.05 \leq x_3 \leq 2$$

For example, on the illustrated image (figure (6)) suppose that the first and last coil is closed, and the other is active. That means that there are five active coils in the spring and two closed ones. The problem is to minimize the volume\weight of the spring, according to equation (14), with respect to the constraints that follow. The spring must carry weight without the material getting destroyed.

The constraints are integrated into the objective function by adding their value to f(x). As a result, f(x) will carry penalty terms which will affect the final output of the function. On those penalties, regularization terms will be applied to scale them properly and have the proper impact on f(x). A larger value of this type will affect the output of the objective f(x) more, while the lower one, less. It needs some experimentation to find the best term for that purpose, and it is dependent on the specific problem. In this work, the regularization term was 10.000. Lower values could be applied as well, around 1000. On another occasion, someone might opt to reduce even more the impact of constraints. However, if the penalty term is insignificant to the magnitude of f(x), the minimization will result in an infeasible solution. If the penalty is balanced, it will lead to a proper solution. To convert the constraints into penalties their value is multiplied by the regularization term and each one is added to the objective value f(x).

The quadratic loss was applied to make the penalty even more strict for the objective function. For example, suppose that there is a constraint:

$$x - 5 \leq 0$$

The value of the penalty is 0 if the constraint is satisfied (x<=5), and it is positive if x-5 >0. This is translated as:

$$penalty(x) = \max(0, x - 5)$$

There is no reason to apply a penalty if the constraint is not broken.

The quadratic loss function refers to the square of this equation.

$$quadratic\_penalty(x) = \max(0, x - 5)^2$$

This allows the penalty to have more impact since the square of a number (outside the (0,1) range) is always greater than the number itself.

The results proved that the quadratic penalty works better than the basic penalty [5].

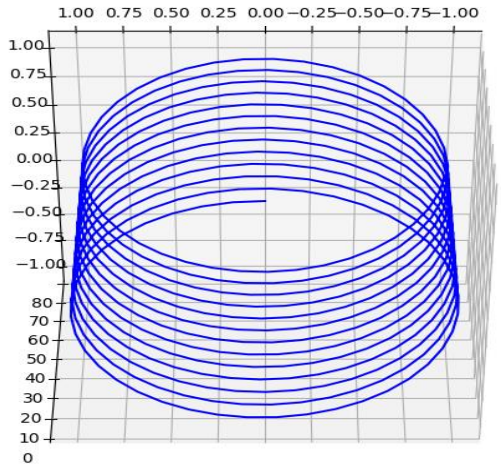

Figure 7: Spring example 2

3.2 Application of SCA on TCSD problem

In SCA's implementation, each element of the population has three variables (x1, x2 and x3, as defined by the problem). Equation (14) calculates the fitness for each element. To adjust SCA for the compression spring problem, the constraints needed to be integrated into the program. Remember that SCA operates as a black box, so the main algorithm remains the same. The only thing that changes is that solutions must stay in range as defined by the problem.

This happens two different times in the program. Firstly, when the whole population is initialized randomly, these values must be limited to the ranges presented. The second time is when SCA's equation (1) is applied. The coordinates that are assigned to each dimension from this equation must stay inside the required scope. For the Chaotic SCA, ranges were used in the same manner.

The individual produced by Chaotic equation (9) will have each of its dimensions in the required range. Careful design of both fitness functions and SCA's took place to produce good results.

3. METHODOLOGY

A robust methodology was used to achieve the objectives. The thesis follows a quantitative methodological approach since quantifiable data, real numbers, were used to measure the performance of the swarm intelligence algorithms. The analysis consists of detailed statistical data to conclude how algorithms behave.

This testing approach enables scientists to ensure that the algorithms can be extended to real-life problems and impact the world. This kind of analysis was wider than the theoretical aspects of the algorithms. Choosing quantitative methodology was easy as most papers about that subject follow this procedure [1], [2], [6]. It is an effective way of testing and perfectly suits the nature of the algorithm, which is manipulating arithmetical data. The reader has reviewed a lot of mathematical equations until this point, something that provides a reason for following this idea. The metrics that are presented in this thesis are in the same spirit. They involve the computation of series to find the function's minimum value. This allows the researchers to compare the results quantitatively and not qualitatively. The results section will verify this thought process.

The quantitative method reviews arithmetical and statistical data to compare the algorithm's effectiveness. This raises the question of why this method is important. The speed and good performance of this approach explain why many scientists adopt this procedure to present their results. When it is adequately undergone, it enables a researcher to expand the implementation to a broader range of problems and with more demanding datasets. Moreover, it is comprehensive and structured since the goals are determined from the beginning, and the purpose of the research is to verify those results or, even better, to improve them. When this method is combined with careful work, it can provide relatable outcomes because its purpose is to test existing hypotheses and provide evidence of why the algorithm behaves in a specific way by proving theories. Remember that all methods have a weak side. It is up to the scientist to have the proper skill to cope with the difficulties that may arise and thoroughly explain the findings. Those outcomes can be generalised in certain situations, which is another positive aspect of it.

Although the quantitative method has many advantages, one should consider its limitations. If the input data is not representative or there are limited sources of them, it can constitute a research complex. Since the input data on this implementation is entirely random real numbers, the thesis overcomes this obstacle. Furthermore, quantitative analysis can be expensive and time-consuming. Considering that this thesis deals with software, all that was needed was computational power which was already existent.

On the other hand, due to extensive analysis, it took much time to produce a detailed report. Thankfully, the results and the acquired knowledge compensated for the efforts. Quantitative analysis requires being familiar with statistics and probabilities. Since a computer scientist wrote this report, much background existed in those areas. Additionally, the presentation of those quantities in diagrams that are descriptive and readable requires expertise. The university provided projects with similar demands, so the prospective graduate was ready to fulfil the tasks.

The dataset that was used in this thesis was based on randomness, the so-called Mersenne Twister. It is an efficient pseudorandom number generator invented by Takuji Nishimura and Makoto Matsumoto in 1997. Its name originates from the Mersenne prime, which is assigned as the period length of this generator. It has passed a lot of statistical tests, which verified its performance. Also, it is cryptographically secure, but

that aspect does not concern this thesis directly. Instead of random data, someone may provide their own solutions, as population individuals with some modifications, and attempt to improve them with SCA and its variants.

The algorithms were implemented with code to improve our understanding of the research field further and build the foundation for even more complex solutions in the future. It is worth noting that C++ language was used in this project because it made it easier to simulate the functionalities of the algorithms and have better segmentation of files with classes. Python language was utilized for the creation of all the images that you will see in this thesis because it offered a friendly environment for that purpose. Latex contributed to the presentation of the results in a formal way, keeping up with the standards of this project.

This work elaborated on the practical side of SCA. Statistics from all the algorithms that were explained previously were obtained. Specifically, mean value, standard deviation, range, median value, and execution time are only some of them. This made it possible to compare quantities and form conclusions that are accurate and relatable to the problem. The quantitative research manner was adopted in the metrics section. The objective function value that is computed by the equations, in theory, is evaluated by universally used test functions. Their behaviour will be explained below, but they use arithmetical data to compute an optimal value.

Some qualitative analysis is also present in this thesis. Someone should pay attention to the diagrams in the theory section. The colours that are shown in the diagrams (blue and orange) show a quality rather than quantity characteristic. They help the reader understand in which direction the solution will direct, according to the results of the mathematical equations. The diagrams that you will see in the results section show some quality too. The thesis reviews the pattern of a given fitness function. Is it linear or not? Do fitness and time increase together, or does one increase as the other decreases? Is the function increasing or decreasing at a constant rate or not? The answers will be shown afterwards. Finally, the bar charts in the metrics section are qualitative attributes because they provide valuable information to the reader about which regions have lower or higher fitness scores.

Finally, this thesis was considered a three-month workload. The student claimed a topic from the assistant professor, Dr Stamatopoulos, in the middle of November 2022. The following month, the student searched through research papers that are shown in the reference section about potential SCA variants that should be presented in this thesis, as well as engineering problems, to verify its performance. After the research was completed, code implementation took place. With cooperation with Dr Stamatopoulos, the code was completed successfully. It should be noted that there was a one and half month pause period due to undergraduate examinations. When the code was completed, the writing of this report began. The work is expected to be finished at the end of April 2023.

4.METRICS

Metrics are a crucial part of assessing algorithms. In this thesis, single objective test functions were used to measure performance. It can definitely be implemented for solving multi-objective optimization too. Both unimodal and multimodal functions were utilized. Remember that unimodal functions concern a single peak, while multimodal functions have multiple peaks. Someone can picture these test functions as simulated landscapes which can enable scientists to learn many essential features of the algorithms, such as precision, convergence rate and any flaws that may be committed during its operation. Some plots are provided to the reader in order to have a better understanding of their behaviour. The metrics section is a stepping stone, as it provides the required information to grasp the results section effectively. Below, each test function that was used in this thesis will be presented, as well as some characteristics about them.

Firstly, Schwefel's 7 test functions:

$$f_1(x) = 418.9829D - \sum_{i=1}^{D} x_i * \sin\left(\sqrt{|x_i|}\right)$$

It is a multimodal continuous nonconvex function defined in the hypercube:

[-500,500] for i =1, 2, …, D. It can be defined in n dimension space and has a negative global minimum value [19].

The searching algorithms are susceptible to missing the optimum global value due to the abrupt surface of the function. There are many local minimum values which can deceive the algorithm into going in the wrong direction. Moreover, the global optimum is quite distant from the second-best value, rendering it difficult for the algorithm to spot it.

The second function is De Jong's1:

$$f_2(x) = \sum_{i=1}^{D} x_i^2$$

It is a continuous unimodal function. It is a convex function, meaning that the curved surface is pointing outwards. Its domain space is:

[-100,100] for i = 1, 2, …, D. It is commonly known as the sphere model. As can be seen in the bar chart, green and blue values have a medium range of fitness. Pink and purple are high-value ones and should be avoided by the process. The desired territory is the yellow and especially the red one since minimum fitness value exists there. Its global minimum is 0.

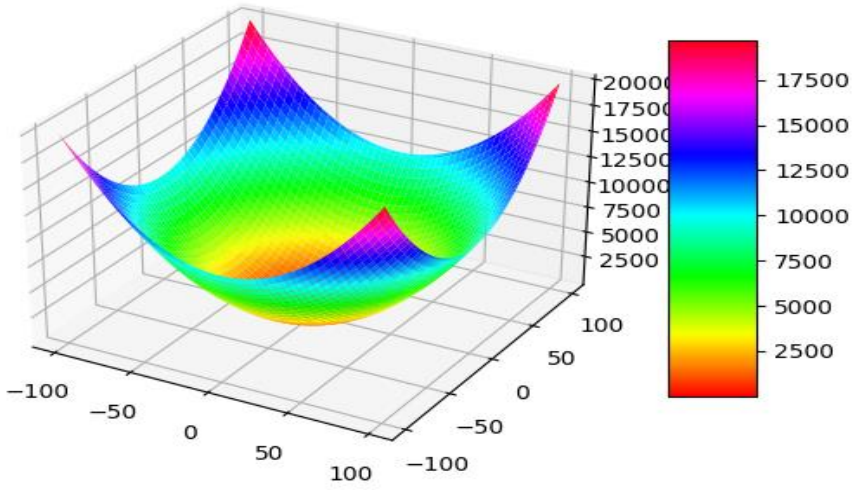

Figure 8: Schwefel's seven 3-dimension graph

The third function is the Rastrigin:

$$f_3(x) = \sum_{i=1}^{D} (x_i^2 - 10 * \cos(2\pi x_i) + 10)$$

It has a similar behaviour as Schwefel's seven but is defined inside [-5.12, 5.12] for i =1, 2, …, D. Its global minimum value is zero. It is continuous, nonconvex, and multimodal, as shown in the graph below. Several local minimum values are allocated around the global minimum, zero. Higher fitness values are found in the corners of the two-dimensional plot.

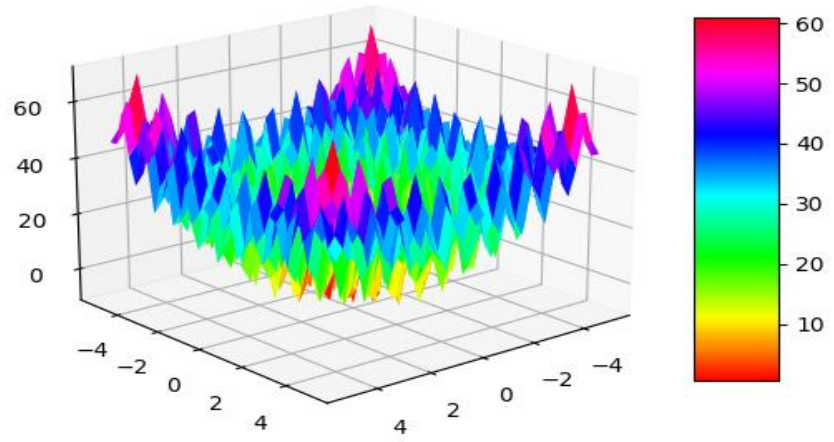

Figure 9: Griewank's 3-dimension graph

The fourth function is named Griewank:

$$f_4(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

It has a vast number of local minimums distributed frequently around the surface. The domain space is the hypercube [-600, 600] for i =1, 2, …, D. The global minimum value of the function is zero. On this occasion, highs and lows are all over the landscape, not only on corners. It is a multimodal function composed of a convex quadratic and a fluctuating(waving) nonconvex one. It differs from the other test functions because it is more difficult to optimize in the initial stages, but then it becomes easier.

Next is the well-known Ackley function:

$$f_5(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi\, x_i\right)\right) + 20 + e$$

David Ackley proposed it in his PhD dissertation in 1987 [26]. The continuous multimodal nonconvex function is evaluated in hypercube [-32, 32] i =1, 2, …, D. It can indeed be limited to smaller spaces. The global minimum value is zero.

The algorithm proposes a challenge for algorithms such as SCA because there are many instances when it gets trapped in a local minimum location. Specifically, Ackley's graph consists of a flat region with a hole at the centre of it. In the former territory, numerous local optima exist, while in the latter, the minimum fitness value is located.

The steepness of the valley where the minimum value resides can be proved difficult for many algorithms to locate.

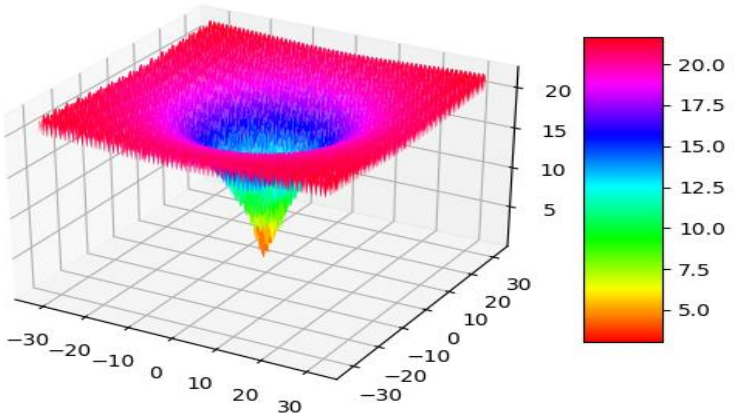

Figure 10:Ackley 3-dimension graph

The following function is called f6-step:

$$f_6(x) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$$

It is the typical step function, but since the steps are multiplied by a square, no negative terms exist in the sum. Due to the floor operation that exists at the base of the power, the series will converge to zero. As a result, the optimum global value in this situation is zero. The nonconvex, not continuous, quadratic function is evaluated in hypercube:

[-100,100] i =1, 2, …, D. It is a unimodal function since it has only one peak.

Proceeding to function 7, which is the Goldstein-Price.

$$f_7(x,y) = [\,1 + (\,x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] * [30 + (2x - 3y)^2(28 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$$

The domain space is fixed: [-2, 2] for i = 2. It has optimal global value 3. It has a very high peak around the point (-2,2), which can be deceiving for many algorithms. Across the surface, there are several valleys which can constitute optimization difficult. The algorithm may fall into a sub-optimal instead of a global optima area, something that should be avoided.

It is a fixed dimensional continuous multimodal and non-convex test function, and the global minimum point is in (0, -1).

The eighth function is the Six-hump camel function:

$$f_8(x,y) = 4x^2 - 2.1x^4 + \frac{x^6}{3} + xy - 4y^2 - 4y^4$$

The scope of this function is the [-5, 5] for i = 2 (two-dimensional). It can be visualized, as the name says, with six humps. It is a landscape that contains valleys and hills. Some valleys are lower than others, especially around the two global minimum points.

It is a continuous multimodal nonconvex function. The global minimum has a value of -1.0316, found at points (0.0898, -0.7126) and (0.0898, 0.7126).

The ninth function is the Rosenbrock.

$$f_9(x) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i+1}) + (x_i - 1)^2]$$

The scope is [-30, 30] for i = 1, 2, …, D and the global minimum value is zero. The function is continuous, nonconvex, and unimodal, and the minimum is in a parabolic valley. Even though it is easy to locate the valley, the convergence to that specific point is complicated. It is a well-known test problem and sometimes is called the valley function.

The tenth function is the Quartic:

$$f_{10}(x) = \sum_{i=1}^{D} (i * x_i^4) + rand(0,1)$$

The scope is [-1.28, 1.28] for i = 1, 2, …, D and the global minimum value is zero. It is a continuous, unimodal test function. It is similar to the DeJong function but with noise.

The eleventh function is the Schwefels2.21.

$$f_{11}(x) = \max\{ |x_i|, 1 \le i \le D \}$$

The range is [-100, 100] for i = 1, 2, …, D and the global minimum value is zero. It has the shape of a reverse pyramid. It is continuous, unimodal, and convex. It gradually descends to the lowest point ((0,0) in two dimensions).

Subsequently, the twelfth function is the Schwefels2.22, defined in the hypercube [-10, 10] for i = 1, 2, …, D and the global minimum value is zero. It is continuous, unimodal, and convex. The mathematical equation of it is as follows:

$$f_{12}(x) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|$$

It looks like a paper folded in four parts due to the absolute values, with the minimum point at the centre. It provides a new challenge for the algorithms since it has different patterns and levels of smoothness.

The thirteenth function is the Schwefels2.12. It is defined in the scope [-100, 100] for i = 1, 2, …, D and global minimum value is zero. It is continuous, unimodal, and convex.

$$f_{13}(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i \right)^2$$

It is commonly known as a double-sum or rotated hyper-ellipsoid test function. It contains a lot of local optima values, proposing a hard task for the algorithms to solve. Along the surface, there is a lot of solution with similar fitness values.

The fourteenth function is the Generalized penalty function two. It is defined in the scope [-50,50] for i = 1, 2, …, D and the global minimum value is zero. Its equation is:

$$f_{14}(x) = 0.1 * \{ 10\sin^2(\pi x_i) + \sum_{i=1}^{D-1}(x_i - 1)^2 [1 + 10\sin^2(3\pi x_i)] + (x_n - 1)^2 \} + \sum_{i=1}^{D} u(x_i, a, k, m)$$

The u function is specified as follows:

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if\ x_i > a \\ 0, & if\ -a \le x_i \le a \\ k(-x_i - a)^m, & if\ x_i < -a. \end{cases}$$

$$a = 5, k = 100, m = 4$$

Finally, the fifteenth function is the Generalized penalty function 1. It is defined in the scope [-50, 50] for i = 1, 2, …, D and the global minimum value is zero. Its equation is:

$$f_{15}(x) = \frac{\pi}{n} * \{ 10\sin^2(\pi y_i) + \sum_{i=1}^{D-1}(y_i - 1)^2 [1 + 10\sin^2(3\pi y_i)] + (y_n - 1)^2 \} + \sum_{i=1}^{D} u(x_i, a, k, m)$$

$$a = 10, k = 100, m = 4$$

$$y_i = 1 + \frac{1}{4} * (x_i + 1)$$

They are both unconstrained functions with a global minimum value of zero. They are continuous, non-convex multimodal test functions. These are the most complex ones in this thesis. Their unique characteristic is that the number of local optima rises exponentially depending on the dimensionality of the solutions. Their graph effectively resembles a penalized spline. Imagine it like multiple waves with lower and higher amplitude [4].

All the above test functions manipulate the corresponding solution from the theory section. Each algorithm is assigned the task of providing the best possible solutions to minimize those functions. The wide variety of behaviours that they present allowed a better understanding of the algorithms' efficiency.

Note: Although the test functions exist in the literature, their order in this thesis is not in the same order as the one typically followed. If you try to compare them, look at which corresponds to the correct number.

4.RESULTS AND DISCUSSION

All the previous sections prepared the reader for a comprehensive understanding of this section. This is the most important part because it verifies that the statements in this thesis are accurate. It will be divided into three particular parts. Firstly, the development of the minimum fitness throughout the execution of each run will be presented. Afterwards, tables will provide information about a variety of experiments regarding the best score that was achieved. An overview of the results will complete this part.

The next three diagrams below illustrate how the minimum fitness changes for each algorithm in test four (Griewank) over the course of the first 2000 iterations and 100 dataset dimensions. As it can be seen, until the first 500 iterations, the minimum fitness stabilizes at a quite high value, about 2500. In contrast, the Aquila algorithm converges to the global minimum of the test function almost instantly (it was recorded at the fifth iteration). Aquila provides a massive improvement compared to the other two algorithms when the dimension of the population is 100. The Chaotic and basic SCA line diagram looks almost similar on this occasion. Even though there is a sizeable difference in the convergence speed to the global minimum among the three algorithms, all of them manage to approach closely to the global minimum. The level of accuracy will be investigated later. For dimension size 100, the clear winner is the Aquila since it spots faster as the globally best value in the Griewank test function (4).

Differentiated results occur when the dimension size increases to 500. SCA's efficiency is essentially the same as when the dimension was 100. It converges partially to the global minimum almost at the same number of iterations. The next 4000 iterations were printed to observe if the convergence of the algorithm would improve. However, that was not the case, as the line remained the same. Remember that 2000 iterations are executed 20 times for each algorithmic experiment, so the first three runs out of the total 20 were recorded. On the other hand, Chaotic SCA improved a lot as the line started from a relatively lower point and converged to the global minimum faster than before. The next 4000 iterations were recorded to verify if the pattern was consistent. Even though the convergence speed looks similar, the line begins at a considerably greater point. Because the algorithms are based on randomness, the initial stages are not always good enough to spot a good solution, so the graph is logical. What is surprising is that whilst the size of the dimension is increasing, Aquila's efficiency does not become affected at all. The algorithm converges to the global minimum at the fifth iteration again.

The Aquila algorithm has the best convergence speed out of both basic and Chaotic SCA. Chaotic SCA proposes a reasonable improvement to basic SCA, especially when the dimension rises significantly to 500. Similar patterns occur in the other test functions too. The thesis manages to enhance the original SCA greatly.

Minimum fitness \ iteration line diagram (dimension=100,500)

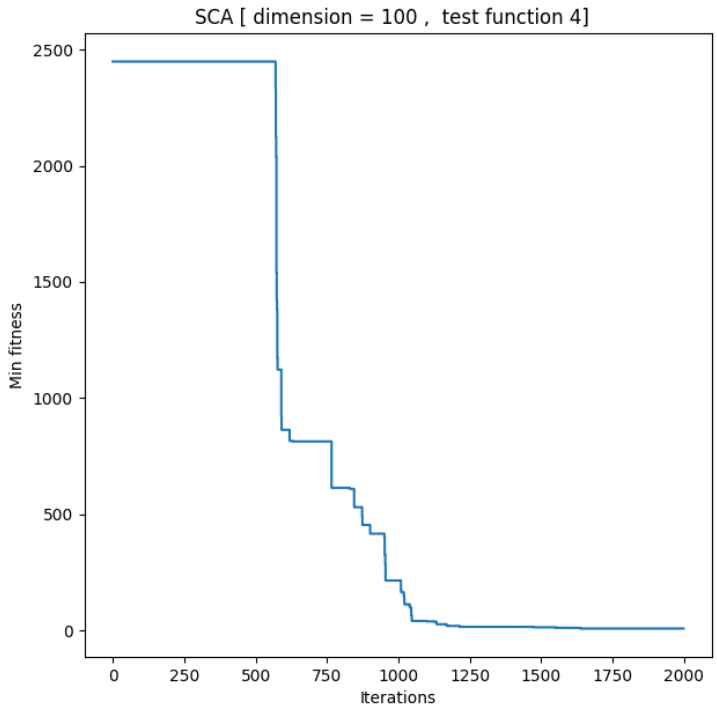

Figure 11: Basic SCA's Min Fitness/iteration diagram (dimension=100)

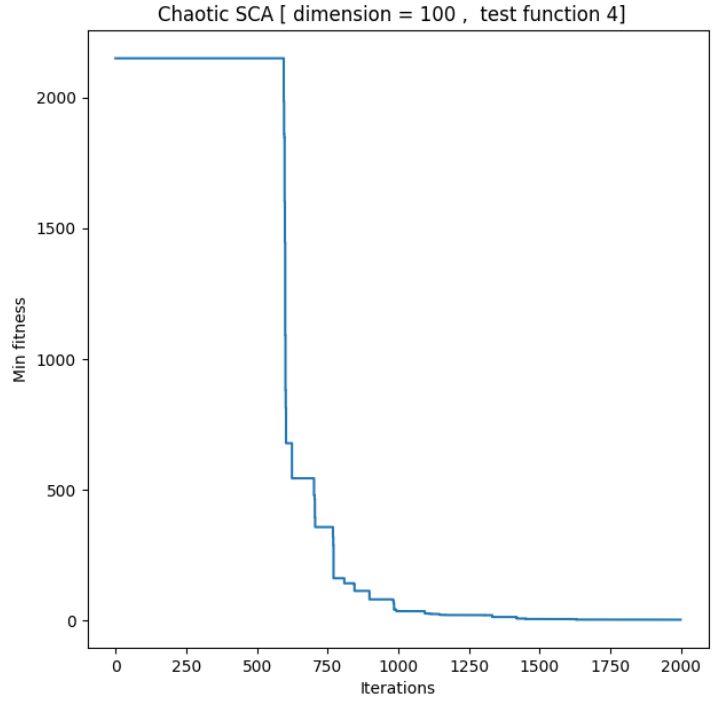

Figure 12: Chaotic SCA's Min Fitness/iteration diagram (dimension=100)

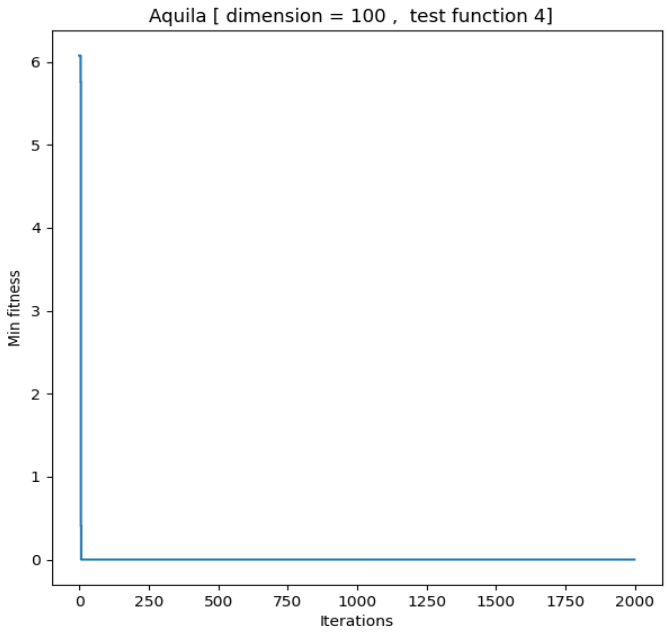

Figure 13: Aquila's Min Fitness/iteration diagram (dimension=100)

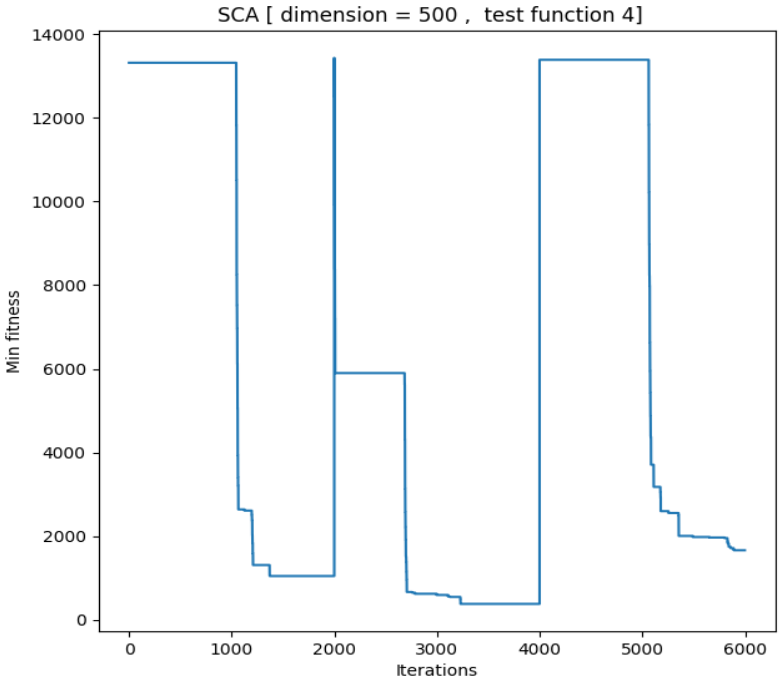

Figure 14: Basic SCA's Min Fitness/iteration diagram (dimension=500)

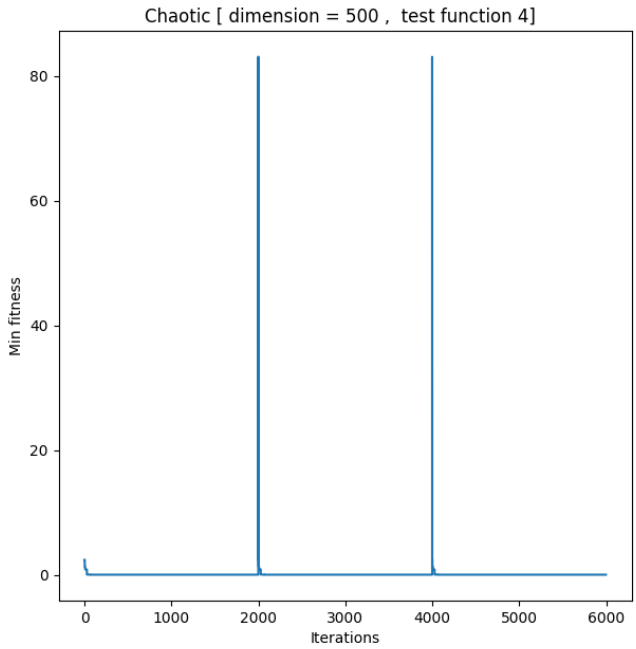

Figure 15: Chaotic SCA's Min Fitness/iteration diagram (dimension=500)

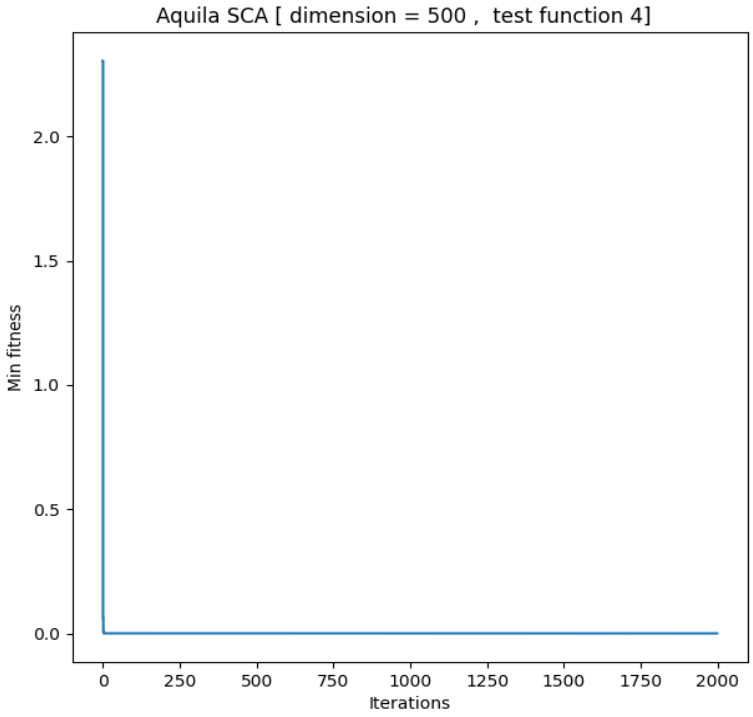

Figure 16: Aquila's Min Fitness/iteration diagram (dimension=500)

The following tables present the parameters that were used throughout the program as well as the scope of each test function in the second column. The third column provides the best fitness value that solutions can produce for the correspondent test function. This, in short, is called the Optimum. Both limited and wider ranges are provided to test

different regions of the test functions which contain the Optimum. It is worth noting that all the algorithms were compared using exactly the same parameters. Population size was chosen according to the existing literature because it is enough to reach a global optimum at a reasonable amount of iterations. This allows the execution time to be satisfying. The “a” value (it is not the same as the Chaotic a) usually equals 2 in research papers on that topic.

Table 1: Parameters of experiment

Parameters		
Test Functions Iterations = 2000 dimension =10, 100, 500 population =100 , a = 2	Scope	Optimum
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	[-500,500]	-418.9829*n
$(f_2)\sum_{i=1}^{D} x^2$	[-100,100]	0
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	[-5.12,5.12]	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600,600]	0
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	[-32,32]	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	[-100,100]	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	[-30,30]	0
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	[-128,128]	0
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	[-100,100]	0
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	[-10,10]	0
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	[-100,100]	0

1

Table 2: Parameters of experiment 2

Sine Cosine Algorithm		
Test Functions Iterations = 2000, dimension =10, 100, 500, population = 100, a = 2 (7th,8th function's dimension = 2)	Scope	Optimum
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2] \times \right.$ $\left.\left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]\right.$	[-2 , 2]	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	[-5 , 5]	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10 \sin^2(\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left[1 + 10 \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	[-50 , 50]	0
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left[1 + 10 \sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	[-50 , 50]	0

Table 3: Basic SCA Results (dimension=10)

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 10	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-20787.7
$(f_2)\sum_{i=1}^{D} x^2$	1.89727E-85
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	5.82291
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00671022
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	1.19658E-29
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	4.85728E-53
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	4.65638E-58

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 10 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10 \sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \left[1 + 10 \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	512.223
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2 \left[1 + 10 \sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	9793.2

The table below provides basic SCA statistics: dimension = 10 , number of runs = 20

Basic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	-16928.9	2505.82	9544.87	-16499.4	13629.5
2	2.58967E-76	9.58097E-76	4.40553E-75	1.91951E-80	12491.4
3	0	0	0	0	13149.2
4	0.0186289	0.0661681	0.298311	0	13031.2
5	3.01981E-15	1.26857E-15	3.55271E-15	3.55271E-15	14067.3
6	0	0	0	0	11968.3
7	3	1.02507E-06	4.37749E-06	3	3318.12
8	-1.03162	4.30707E-06	1.44293E-05	-1.03163	3054.05
9	6.46656	0.357481	1.44768	6.43171	15455.8
10	0.0986462	0.0810359	0.261115	0.0873992	130863
11	2.21125E-24	9.57844E-24	4.39625E-23	1.11764E-27	12077.9
12	2.78093E-47	8.50579E-47	3.82025E-46	2.00955E-50	12175.6
13	1.00513E-43	4.37747E-43	2.00861E-42	1.08809E-51	13972.5
14	3817.71	2915.57	10605.1	2841.69	21375.5
15	52292.2	37381.6	144740	44282.7	19731.6

The results for basic SCA are satisfying for most functions. In most instances, it manages to approach the optimum global value apart from the f9, f14, and f15. It needs to be emphasized that even though it approximates the global optimum, on most occasions, it does not reach it totally.

As can be observed from the statistics, SCA needs about 12 seconds to execute 2000 iterations 20 times for each test function. The mean value is very increased for the fourteenth and fifteen instances. On the other hand, in the seventh function, the algorithm always manages to find the optimal value. This is explained by the fact that the mean value is equal to the minimum fitness that the algorithm produces. The range values are not relatively large apart from functions 14,15 and 1. Significant range values could be interpreted as difficulty in finding the global optima. The median value enables observers to determine the centre of the solutions dataset. When it is close to the mean value, that means that the dataset is distributed in equal amounts from lowest to highest. This is happening most of the time except for f14 and f15. Median helps eliminate outliers(values that are not prevalent in the dataset but they affect the mean value). When the mean is less than the median, the graph is negatively skewed (weighing to the left). On opposite occasions, it is positively skewed (weighing to the right). Standard deviation refers to which place the data is more concentrated on. Low values of it mean that the data are distributed around the mean. In contrast, significant value can be viewed as a sparse graph, where the data are clustered above the mean. The latter happens in f1, f14, and f15.

Table 4: Basic SCA Results (dimension=100)

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 100	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-36429.2
$(f_2)\sum_{i=1}^{D} x^2$	23.4694
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	39.1849
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	1.41739
(f_5)-20 $*$ $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ $-$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0.00094925
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	365768
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	7.39593E+07
$(f_{11})\max_{i=1}\{\lvert x_i \rvert \ , 1 <= i <= D\}$	68.5121
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \Pi_{i=1}^{D} \lvert x_i \rvert$	4.77144E-06
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	61206.2

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 100 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10 \sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \left[1 + 10 \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	1.88837E+07
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2 \left[1 + 10 \sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	9.26712E+07

The table below provides basic SCA statistics: dimension = 100 , number of runs = 20

basic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	-22275.3	9260.31	29430.7	-24882.2	146220
2	1124.8	1524.89	5751.94	612.478	132732
3	147.278	67.8845	271.585	142.41	148808
4	8.38202	9.55845	38.4122	4.43208	145429
5	17.756	6.52376	19.2363	20.5315	133094
6	701.75	1039.47	4754	412	124541
7	3	6.5132e-07	2.22195e-06	3	3211.18
8	-1.03162	2.89182e-06	9.41659e-06	-1.03163	3472.3
9	1.34133e+07	1.0837e+07	3.74738e+07	1.0086e+07	163856
10	1.22366e+09	1.23128e+09	4.51576e+09	5.10412e+08	250285
11	76.9844	4.11544	14.4343	77.9977	126087
12	0.0808108	0.318942	1.46986	0.0023698	133962
13	117265	32807.2	133558	116033	226585
14	8.74646e+07	5.95815e+07	2.31267e+08	8.02501e+07	185728
15	1.66223e+08	4.35515e+07	1.52223e+08	1.64104e+08	204746

A new trend can be observed for the basic SCA when the dimension is increased to 100. All the test function's fitness value rises greatly. f10, f12, f13, and f14 minimum fitness increases significantly. Those test functions were placed in that order because it becomes gradually difficult to find global minimum by increasing order. The value for the fixed dimension function remains equal to the previous table's one. The only function that the fitness improves is the f1. It can be understood that the dimension size plays a significant role in the complexity of the problem. The higher the dimension means a more complicated situation and, thus, probably a greater minimum overall fitness value.

The situation is clear in the statistics table too. The table has lost many low values that were existent in the previous statistics table. This means that the overall range, standard deviation, mean value and median have increased notably. Something that is on par with the worsening picture in the minimum fitness table. Programmers want to keep those values as low as possible to improve their chances of extracting the actual global minima from the test function. The greatest mean value and median are in f15's row. The largest execution time in f10's. The lowest mean value and range are assigned to f1. The lowest value for standard deviation and range columns is inside f7's and f8's row.

Table 5: Basic SCA Results (dimension=500)

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 500	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	6485.33
$(f_2)\sum_{i=1}^{D} x^2$	47133.7
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	192.286
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	227.598
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	9.08693
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	61546
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	5.91844E+08
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	4.24644E+11
$(f_{11})\max_{i=1}\{\lvert x_i \rvert \ , 1 <= i <= D\}$	96.6532
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	7.98993
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	3.32514E+06

basic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 500 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10 \sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \left[1 + 10 \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2.64124E+09
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2 \left[1 + 10 \sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2.82566E+09

The table below provides basic SCA statistics: dimension = 500 number of runs = 20

basic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	56051.3	22529	91515.9	53051.4	726044
2	114699	46196.4	165520	117865	644968
3	917.188	410.329	1730.17	881.651	685884
4	954.873	376.591	1585.92	935.079	721914
5	19.14	3.85266	11.7239	20.7404	641660
6	123008	37489.2	114016	129813	762108
7	3	4.29046E-07	1.87906E-06	3	3244.47
8	-1.03163	3.02719E-06	9.89178E-06	-1.03163	3100.37
9	1.17035E+09	2.43739E+08	9.90849E+08	1.13938E+09	862444
10	8.17642E+11	2.01064E+11	7.63496E+11	7.80781E+11	804187
11	98.0548	0.538488	2.10855	98.1731	631537
12	51.2403	40.2658	168.796	37.2065	655476
13	4.14651E+06	611184	2.35926E+06	4.11725E+06	2.34843E+06
14	5.27836E+09	1.12777E+09	4.74151E+09	5.36496E+09	884038
15	3.96442E+09	5.50714E+08	2.28524E+09	3.87453E+09	1.10284E+06

The tendency in this experiment follows the same pattern as when the dimension was 100. The minimum fitness values are rising even more, missing entirely the desired global minimum point. This renders SCA ineffective in dealing with these higher-dimension problems. The greater minimum fitness exists in test function 10, which is 4.2E+11, while the minimum is zero. The only functions that withstand the complexity at some level are the f5, f11 and f12. The statistics verify that effect. The mean value and the range are almost double the minimum fitness on most occasions. High execution times are observed (around 16 minutes maximum). The median is sometimes lower and other times higher than the mean. When the standard deviation is close to zero, it means that the solutions are concentrated on some value. When it is relatively elevated, as in f14, it shows that the solution varies a lot. The improvement of this algorithm to tackle this adversity becomes imminent.

Table 6: Chaotic SCA Results (dimension=10)

Chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 10	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-80750.8
$(f_2)\sum_{i=1}^{D} x^2$	4.45339E-66
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
(f_5)-20 $*$ $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ $-$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	4.98473E-06
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00284738
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	1.22582E-24
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	3.26137E-41
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	5.14365E-47

chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 10 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10 \sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \left[1 + 10 \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	16.0018
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2 \left[1 + 10 \sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	58.8684

The table below provides Chaotic SCA statistics: dimension = 10 , number of runs = 20

Chaotic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	-80750.8	0.00188297	0.0106568	-80750.8	14773.6
2	4.5419E-61	1.07048E-60	4.47461E-60	2.92542E-63	12303.2
3	0	0	0	0	13612.3
4	7.75889E-07	0	0	7.75889E-07	152748
5	0	0	0	0	18471.3
6	0	0	0	0	13146.9
7	3	9.98346E-07	3.42305E-06	3	4029.26
8	-1.03163	1.69009E-06	5.71581E-06	-1.03163	5661.71
9	4.98473E-06	8.47033E-22	0	4.98473E-06	20014.1
10	0.0993132	0.0958357	0.340183	0.0686477	234241
11	9.10249E-09	3.5257E-08	1.61549E-07	1.08629E-17	15514.6
12	2.63183E-37	7.13119E-37	3.09024E-36	4.29862E-39	14438.1
13	3.93327E-37	1.21898E-36	4.92897E-36	1.90253E-43	17540.2
14	16.0018	3.90931E-11	1.80606E-10	16.0018	24453.2
15	58.8684	2.70671E-11	1.0165E-10	58.8684	24899

Even though dimension size has not escalated a lot, chaos is already very effective. From the first two tables, it can be observed that Chaotic sine cosine has better performance than basic SCA in f1, f9, f10, f14 and f15. Test functions f2, f11, f12 and f13 show slightly better scores for SCA but not by a lot. Test functions f3, f4, f5, f6, f7, and f8 have the same scores for both algorithms.

Regarding the statistics, basic SCA runs 2-3 seconds faster per test function in that dimension. However, Chaotic SCA has a lot better statistics table. For functions f7 and f8, the scores are equivalent. Only a little less standard deviation is produced by Chaotic SCA. f3, f5 and f6 columns are the same too. The figures in other test functions follow the same trend as the minimum fitness tables. The one that scored better minimum fitness has lower values in the statistics columns. It is impressive that at f14 and f15 Chaotic SCA has a standard deviation of almost 0 while basic SCA's are around 3000. That shows that Chaotic SCA is a lot more accurate and does not diverge a lot from the best solution.

Table 7: Chaotic SCA Results (dimension=100)

Chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 100	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-807508
$(f_2)\sum_{i=1}^{D} x^2$	8.23026E-07
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	4.28032E-07
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	7.75889E-07
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0.000116167
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	5.4832E-05
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	4.98473E-06
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00537346
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	9.07208E-05
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	0.000907208
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	0.0773064

Chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 100 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	160.018
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	78.7623

The table below provides chaotic SCA statistics: dimension = 100 runs number = 20

chaotic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	-807508	1.16415E-10	0	-807508	154069
2	8.23026E-07	0	0	8.23026E-07	130476
3	4.28032E-07	0	0	4.28032E-07	143102
4	7.75889E-07	0	0	7.75889E-07	152748
5	0.000116167	0	0	0.000116167	142442
6	0	0	0	0	131112
7	3	5.62055E-07	2.12342E-06	3	4544.83
8	-1.03163	1.69099E-06	5.71581E-06	-1.03163	5661.71
9	5.4832E-05	6.77626E-21	0	5.4832E-05	223667
10	0.0590336	0.0523052	0.159945	0.044928	383584
11	9.07208E-05	2.71051E-20	0	9.07208E-05	128394
12	0.000907208	0	0	0.000907208	132840
13	0.188107	0.0651871	0.253554	0.177522	284954
14	160.018	4.21031E-12	1.69678E-11	160.018	236475
15	78.7623	1.41209E-14	0	78.7623	241131

As expected, when the dimensionality of the problem rises, variants of SCA are able to improve the weaknesses of the original SCA. This time, Chaotic SCA is better on all the test functions apart from f5, f6 and f12. The figures are represented equally in f7 and f8 since the dimension is fixed. A striking feature of the minimum fitness table is that the SCA f10 score stands at 7.3E+07, while Chaotic is around 0.005. Also, the value of SCA's best fitness on f15 accounts for 2.82E+09, whereas Chaotic is almost 78. Chaotic SCA is approximately one hundred million times superior to the original SCA.

Chaotic SCA records vastly better statistics than original SCA in all the test functions. This makes it clear that Chaotic SCA has the potential to provide better results than SCA. Basic SCA outscored Chaotic SCA on some occasions only by luck since the algorithms are based on randomness. Chaotic SCA has a better dynamic, and that is verified by the minimum fitness tables.

Table 8: Chaotic SCA Results (dimension=500)

Chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 500	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-4.03754E+06
$(f_2)\sum_{i=1}^{D} x^2$	4.11513E-06
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	2.14018E-06
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	1.04336E-06
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0.000116167
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	0.000276375
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00264206
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	9.07208E-05
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	0.00453604
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	14.5321

Chaotic Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 500 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	800.09
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	85.6839

The table below provides chaotic SCA statistics: dimension = 500 runs number = 20

chaotic SCA					
Test Functions	mean value	std	range	median	time (ms)
1	-4.03754E+06	4.65661E-10	0	-4.03754E+06	778424
2	4.11513E-06	8.47033E-22	0	4.11513E-06	649488
3	2.14018E-06	0	0	2.14018E-06	695362
4	1.04336E-06	0	0	1.04336E-06	717411
5	0.0579676	0	0	0.0579676	883614
6	0	0	0	0	634575
7	3	4.26963E-07	1.71606E-06	3	4143.57
8	-1.03163	2.31843E-06	8.77307E-06	-1.03163	3783.05
9	0.000276375	1.0842E-19	0	0.000276375	1.0415E+06
10	0.0353398	0.0404854	0.193874	0.0225168	964571
11	9.07208E-05	2.71051E-20	0	9.07208E-05	618668
12	0.00453604	8.67362E-19	0	0.00453604	625175
13	29.6566	9.82724	36.4574	26.5931	4.06906E+06
14	800.09	7.97325E-12	3.10365E-11	800.09	1.18743E+06
15	85.6839	9.81352E-13	3.12639E-12	85.6839	1.30727E+06

It is evident from the table that Chaotic SCA is invariably the most productive out of the two. On all test functions, Chaotic SCA yields minimum fitness to a great extent. Undoubtedly, Chaotic SCA is the viable variant when the program has to deal with high dimensional populations. For instance, in f2, Chaotic manages 4.11E-6 whilst SCA only 47113.7. SCA is unable to provide profound results because it overlooks the global minimum, which is zero. Chaotic SCA is close to reaching it in almost all situations.

Statistics of SCA in dimension 500 maintain the same numbers approximately as when the dimension was 100, apart from execution time. That proves that Chaotic SCA is robust and does not get too affected by dimension surge. However, basic SCA's statistics are a lot worse than Chaotic's, and that renders essential the use of the variants.

Table 9: Modified Aquila Results (dimension=10)

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 10	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-213850
$(f_2)\sum_{i=1}^{D} x^2$	0
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
(f_5)-20 * $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ − $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	8.77507
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00054939
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	0
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	0
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	0

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 10 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3.09608
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.02729
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	61704.4
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2266.13

The table below provides Aquila statistics: dimension = 10 , number of runs = 20

Aquila					
Test Functions	mean value	std	range	median	time (ms)
1	-22288.7	45441.7	217302	-12788.1	15380.5
2	0	0	0	0	13124.6
3	0	0	0	0	41962.9
4	0	0	0	0	43476.5
5	0	0	0	0	15917.4
6	0	0	0	0	13668.3
7	24.3442	32.1472	89.0092	8.51141	4296.28
8	-1.00896	0.011648	0.0310849	-1.0002	3790.88
9	8.90746	0.0330498	0.192236	8.90969	21270.9
10	0.0482245	0.0512732	0.1623	0.0299196	244560
11	0	0	0	0	13347.9
12	0	0	0	0	13243.7
13	0	0	0	0	16107.4
14	410934	133750	513133	436933	22128.4
15	3.33199E+06	2.4067E+06	7.83385E+06	2.53119E+06	24517.9

When the dimension is 10, Aquila outperforms the other algorithms in the majority of the test functions. In particular, it is the leading procedure in nine out of the fifteen test functions. It performs worse than Chaotic in f7, f8, f9, f14 and f15. Moreover, it obtains inferior minimum fitness to basic SCA in f7, f8 and f9.

Regarding the statistics, Aquila SCA is superior to basic SCA in nine out of the fifteen functions. In detail, it does not surpass it in f1, f7, f8, f9, f14 and f15 since its std, range, and median is higher than the basic one. In addition, it requires substantially more execution time per test(around 10-200 sec). In comparison to basic SCA, the outlook is the same as the original SCA. A lot of emphasis needs to be put on the fact that Aquila manages to find not only the best fitness value but also the global minimum point of the test.

Table 10: Modified Aquila Results (dimension=100)

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 100	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-1.43255E+06
$(f_2)\sum_{i=1}^{D} x^2$	0
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
(f_5)-20 $*$ $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ $-$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos\left(2\pi x_i\right)\right) + 20 + e$	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	97.9998
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	0.00493745
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	0
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	0
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	0

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 100 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3.09608
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.02729
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	1.79248E+07
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	1.00379E+07

The table below provides Aquila statistics: dimension = 100 , number of runs = 20

Aquila					
Test Functions	mean value	std	range	median	time (ms)
1	-321214	469031	1.46862E+06	-97392.2	149316
2	0	0	0	0	132086
3	0	0	0	0	132674
4	0	0	0	0	137255
5	0	0	0	0	160411
6	0	0	0	0	132865
7	36.9705	38.4085	90.9866	11.221	4080.99
8	-1.00282	0.00818444	0.0298281	-0.99999	3854.15
9	98.0503	0.0926432	0.403997	98.0116	207973
10	0.0441697	0.0411243	0.140077	0.0238207	395404
11	0	0	0	0	145849
12	0	0	0	0	137933
13	0	0	0	0	290530
14	4.73665E+06	1.08139E+06	3.73265E+06	5.14202E+06	228702
15	7.97804E+07	2.34546E+07	8.63214E+07	8.92682E+07	264472

Further escalation of the dimension leads to a clear difference in performance. On this occasion, Aquila is a better minimizer than SCA in almost all test functions apart from the ones whose dimension is fixed(f7, f8). In comparison with Chaotic's implementation, it is a lot closer. In detail, Chaotic exhibits greater accuracy in f7, f8, f9, f14 and f15, whereas Aquila on all the others. However, both are capable of approaching the global minima on all tests(except Aquila in f14 and f15 when its output is significantly away from the optimal). In the majority of test cases, Aquila shows better accuracy by finding the optimal value. It is evident from the table that those variants are enhancing the original algorithm by a wide margin.

Regarding execution times, Aquila is executed faster than SCA but slightly slower than Chaotic by a few seconds (1-20 seconds per test function). Aquila's statistics ensure better quality than basic SCA apart from f1, f7, f8, f14 and f15. In those tests, Aquila produces significant standard deviation and range. This maybe expresses a liability of the Aquila algorithm. In comparison to Chaotic, it represents better effectiveness in the majority of tests except for f1, f7, f8, f9, f14 and f15. Remember that the desired output in each column of the statistic table is the minimum one.

Table 11: Modified Aquila Results (dimension=500)

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 500	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-8.77512E+06
$(f_2)\sum_{i=1}^{D} x^2$	0
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	0
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0
(f_5)-20 $*$ $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ $-$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	0
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	0
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	493.959
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	3.46672E-05
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	0
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \Pi_{i=1}^{D} \lvert x_i \rvert$	0
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	0

Aquila Algorithm	
Test Functions Iterations = 2000 , dimension = 500 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3.55662
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.01011
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	1.79248E+07
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2.29267E+08

The table below provides Aquila statistics: dimension = 500 , number of runs = 20

Aquila					
Test Functions	mean value	std	range	median	time (ms)
1	-1.68754E+06	2.13198E+06	8.8146E+06	-971242	913682
2	0	0	0	0	727311
3	0	0	0	0	736025
4	0	0	0	0	768704
5	0	0	0	0	855570
6	0	0	0	0	768406
7	48.5805	40.8588	94.2801	38.709	4901.34
8	-1.00027	0.0025996	0.0153118	-0.999998	4517.63
9	494.19	0.219617	0.93231	494.123	1.25016E+06
10	0.0156662	0.0126872	0.0454721	0.0139684	1.13641E+06
11	0	0	0	0	747601
12	0	0	0	0	775909
13	0	0	0	0	4.35703E+06
14	2.67414E+07	3.38714E+06	1.11228E+07	2.88223E+07	1.32204E+06
15	4.12842E+08	7.8288E+07	2.53133E+08	4.56141E+08	1.29483E+06

When the dimension is 500, the most demanding test in this thesis takes place. As previously, Aquila outperforms basic SCA in all tests apart from f7 and f8. Regarding chaotic's implementation, Aquila surpasses it in most tests except for f7, f8, f9, f14 and f15. The most striking feature of these tables is that Aquila clearly beats SCA, as its scores are more than a thousand times better. However, a weakness of Aquila is showcased in f14 and f15, when Chaotic is undoubtedly better by around a million times.

Regarding the statistics, Aquila remains robust. It illustrates greater scores than basic in all tests except for f1, f7 and f8. It is exceptionally effective since a lot of values are near zero, even though the dimension has increased significantly. This is the goal of the statistics experiment.

Execution times are worse than SCA's and Chaotic's by 0-4 minutes per test approximately. Finally, Aquila remains competitive in comparison to Chaotic. Aquila's statistics are better than Chaotic in f2-6 and f11-13, which constitute the majority of the tests.

Table 12: Fuzzy Sine Cosine Algorithm (dimension=500)

Fuzzy Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 100	Minimum Fitness
$(f_1)\sum_{i=1}^{D} -x_i * \sin\left(\sqrt{\lvert x_i \rvert}\right)$	-10598.3
$(f_2)\sum_{i=1}^{D} x^2$	36620.9
$(f_3)\sum_{i=1}^{D} \left(x_i^2 - 10 * \cos\left(2\pi x_i\right) + 10\right)$	387.257
(f_4) $\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	181.12
(f_5)-20 $*$ $\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)$ $-$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + e$	2.13021
$(f_6)\sum_{i=1}^{D} \left(\lfloor x_i + 0,5 \rfloor\right)^2$	45117
$(f_9)\sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$	5.88526E+08
$(f_{10})\sum_{i=1}^{D} \left(i * x_i^4\right) + random[0,1)$	4.59531E+11
$(f_{11})\max_{i=1}\left\{\lvert x_i \rvert \ , 1 <= i <= D\right\}$	96.7786
$(f_{12})\sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	1.68386
$(f_{13})\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	2.70255E+06

Fuzzy Sine Cosine Algorithm	
Test Functions Iterations = 2000 , dimension = 500 (7th,8th function's dimension = 2)	Minimum Fitness
(f_7) $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \times$ $\left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	3
(f_8) $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	-1.03163
(f_{14}) $a = 5, 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = 1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $0,1 * \left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2\left[1 + 10\sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2.78853E+09
(f_{15}) $a = 10, k = 100, m = 4$ $f_{\min}(x^*) = 0$ $x_i^* = -1$ $-50 \leq x_i \leq 50, i = 1, 2, \ldots, n$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $\frac{\pi}{n} * \left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} +$ $+ \sum_{i=1}^{n} u(x_i, a, k, m)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & if x_i > a \\ 0, & if -a \leq x_i \leq a \\ k(-x_i - a)^m, & if x_i < -a \end{cases}$	2.60103E+09

The table below provides fuzzy SCA statistics: dimension = 500 runs number = 20

fuzzy SCA					
Test Functions	mean value	std	range	median	time (ms)
1	46555.7	29150.2	110685	50450.3	888526
2	104019	43564.3	155312	94907.4	652742
3	880.181	341.771	1601.18	837.163	768700
4	818.624	289.955	1087.53	871.164	946404
5	74.9977	77.3816	284.14	41.7938	1.05373E+06
6	104498	42779.5	164484	103508	664039
7	3.00002	2.41609E-05	8.29965E-05	3.00001	7559.02
8	-1.03163	1.71368E-06	5.69039E-06	-1.03163	7510.01
9	9.15069E+08	2.08813E+08	6.85189E+08	8.52199E+08	1.23818E+06
10	6.15227E+11	9.59151E+10	3.2346E+11	6.23879E+11	1.2139E+06
11	98.1016	0.501821	1.95842	98.2775	607507
12	54.7367	40.8363	184.347	45.9353	685010
13	3.51077E+06	575041	1.91579E+06	3.43586E+06	5.54182E+06
14	4.3428E+09	8.97419E+08	3.38315E+09	4.23422E+09	1.3612E+06
15	3.38514E+09	4.76631E+08	1.68064E+09	3.42187E+09	1.56456E+06

The fuzziness concept improves basic SCA results, but it is not on the same level as Aquila and Chaotic.

Table 13: Tension/Compression spring design Results

Tension/Compression spring design problem			
Algorithm	Test 1 optimum	Test 2 optimum	Test 3 optimum
SCA	0.0127671	0.0126999	0.0126956
Chaotic	0.0127077	0.0126931	0.0126972

1st note: test 1 happened with 2000 iterations
and 100 population (20 runs of this scenario)
2nd note: test 2 happened with 5000 iterations
and 500 population(20 runs of this scenario)
3rd note: test 3 happened with 10000 iterations
and 1000 population(20 runs of this scenario)

Tension/Compression spring design problem			
Algorithm SCA	x_1	x_2	x_3
Test 1 optimum	0.05	0.31689	14.1155
Test 2 optimum	0.0512801	0.346484	11.9386
Test 3 optimum	0.0515872	0.353904	11.4799

Tension/Compression spring design problem	
Best Algorithm	Best test optimum
Chaotic	0.0126896

4th : Best result recorded throughout the experiment

Tension/Compression spring design problem			
Algorithm Chaotic	x_1	x_2	x_3
Test 1 optimum	0.0517701	0.358044	11.2427
Test 2 optimum	0.0512801	0.346484	11.9386
Test 3 optimum	0.051292	0.346791	11.9168
Best Test optimum	0.0518398	0.359917	11.1195

The table below provides statistics of spring problem: dimension = 3 runs number = 20

statistics of spring problem					
Algorithm	mean value	std	range	median	time (ms)
SCA	0.0128658	5.38808E-05	0.000210111	0.0128618	4594.72
SCA	0.0128592	6.55115E-05	0.000278647	0.0128399	4663.71
SCA	0.0127303	1.83158E-05	6.4362E-05	0.012729	220683
Chaotic	0.0128418	7.10529E-05	0.000286051	0.0128338	4530.63
Chaotic	0.0127557	2.83529E-05	0.000107146	0.012755	54764.7
Chaotic	0.0127325	1.94796E-05	7.80872E-05	0.0127292	232703
Chaotic	0.0127123	1.24685E-05	4.48599E-05	0.0127111	1.16349E+06

In this section, the original SCA algorithm and its Chaotic variation are implemented to solve the compression design problem. A thorough experimentation takes place since many tests are performed to measure performance. Three different tests involving a varying number of iterations and populations happen. It is evident that the Chaotic variant manages to outscore the original SCA in most situations. It also manages the best overall score throughout the experiment (0.0126896). This thesis attained the optimization of the SCA algorithm on this specific problem since minimum fitness improved further by Chaotic variant. The variables that produced the best score are illustrated on the corresponding tables for each experiment. The fourth experiment is more extensive, with more iterations and a larger population capacity, in order to further

improve performance. The outcome verifies that by proposing more demanding tests, someone can improve results even more.

Regarding the statistics, they correspond to the minimum fitness table. The tests, which produced better scores, usually depict more satisfying values in statistics. For instance, the last row in the statistics table depicts the fourth test with the Chaotic variant. This attempt produced the lowest mean value, standard deviation, range and median. However, it required more execution time in comparison to the other attempts because it was extensive. It should be noted that this time program utilizes the metrics from the theory section about the tension spring problem rather than the metrics from the results sections. Also, it uses only three dimensions since it involves only three variables defined by the problem. It limits those specific variables in the given domains, as shown in theory. The results are satisfying since this thesis not only managed to solve the tension spring design with competitive scores but also achieved to augment SCA scores with the Chaotic variant. Further iterations and larger population of solutions could have led to even better scores. The optimal results regarding the problem were approximately 0.01266, as shown in the literature [11].

7.CONCLUSIONS AND FUTURE WORK

Overall, this analysis led to several positive outcomes. Sine and Cosine Algorithm was scrutinized through a wide range of experiments, and the integration of its variants into the main process proved successful. This thesis achieved almost optimal results even when the dimensions of the solutions were large, verifying that our approach was effective. Our results are equivalent to the benchmarks that are displayed in the literature, constituting this thesis as valid and with great potential for new improvements. Based on this work, a new variant may arise that can solve even more challenging problems. Another area of possible research is the field of test functions [17]. More challenging tests can be proposed to further limit test these very robust algorithms. This research showcased the magnitude of those procedures since their application can be extended to many aspects of real life. A well-known engineering problem was solved, which contributed to familiarising the undergraduate student with the practical sides of those techniques [16]. It is very fascinating to explore the problem-solving capabilities of swarm intelligence algorithms, such as SCA, because the program can mimic a lot of behaviours of the natural world. Everything that can be observed in the environment might be applicable to code, and that itself is impressive. Finally, an extensive results report enabled the student to learn how to elaborate further on the topic and have a better understanding of the behaviour of population-based procedures. SCA proved to be a tool that can be utilized on the majority of occasions and provide a valuable solution in an easy way and at a reasonable time.

TABLE OF TERMINOLOGY

optimisation	The act of improving results at the best possible level.
Population-based algorithms	Algorithms that update multiple solutions instead of a single one to obtain result.
Test functions	Artificial landscapes that are utilized by computer programmers to observe the behavior of optimisation of algorithms.
Global optima	A feasible point of a test function that its objective value is greater than all other points.
Fitness	Evaluates the performance of an algorithm.
Agent	A feasible solution in the search space.
Feasible solution	Solution inside the domain space
Deterministic (process)	A process in which no randomness is involved in the production of future states
Stochastic (process)	A process in which randomness is involved in the production of future states
Metaheuristics	Heuristics or higher lever procedures that provide strategies for developing optimisation algorithms
Swarm Intelligence	Population-based(collective) intelligence
Exploration stage	Explore the whole domain space
Exploitation stage	Search around local solution
Objective function	A function that produces solutions which minimise (or maximise) objective
Black Box	No adjustments of the algorithm according to the input
Mutation stage	A different stage added in the original process
Bifurcation diagram	A visual representation of period-doubling as ‘a‘ increases (Ex. period=2: two numbers, period=4: four numbers)
Γ	Gamma function

ABBREVIATIONS ACRONYMS

SCA	Sine Cosine Algorithm
OBL	Opposite based Learning
TCSD	Tension Compression Spring Design
ΕΚΠΑ	Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών
CEC	IEEE Congress on Evolutionary Computation
PID	Proportional Integral Derivative
PSO	Particle Swarm Optimisation

ANNEX I: HARDWARE SPECS AND TOOLS

Operating System

Windows 10 Home 64-bit

CPU

Intel Core i5 4460 @ 3.20GHz

Haswell 22nm Technology

RAM

16,0GB Dual-Channel DDR3 @ 657MHz (9-9-9-24)

Motherboard

Gigabyte Technology Co. Ltd. Z97X-SLI-CF (SOCKET 0)

Graphics

23MP65 (1920x1080@60Hz)

2047MB NVIDIA GeForce GTX 760 (ASUStek Computer Inc)

Storage

111GB SanDisk SDSSDHII120G (SATA (SSD))

931GB Samsung SSD 860 EVO 1TB (SATA (SSD))

C++ version

g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

Python3 version

Python 3.8.10

Visual studio version

code 1.77.3

REFERENCES

[1] S. Mirjalili, “SCA: A Sine Cosine Algorithm for solving optimization problems,” Knowledge-Based Systems, vol. 96, pp. 120–133, Mar. 2016, doi: https://doi.org/10.1016/j.knosys.2015.12.022.

[2] M. H. Ali, A. T. Salawudeen, S. Kamel, H. B. Salau, M. Habil, and M. Shouran, “Single- and Multi-Objective Modified Aquila Optimizer for Optimal Multiple Renewable Energy Resources in Distribution Network,” Mathematics, vol. 10, no. 12, p. 2129, Jun. 2022, doi: 10.3390/math10122129. [Online]. Available: http://dx.doi.org/10.3390/math10122129.

[3] M. Abd Elaziz, D. Oliva, and S. Xiong, “An improved Opposition-Based Sine Cosine Algorithm for global optimization,” Expert Systems with Applications, vol. 90, pp. 484–500, Dec. 2017, doi: https://doi.org/10.1016/j.eswa.2017.07.043.

[4] “Generalized Penalized Function No.01,” Power Systems and Evolutionary Algorithms. https://al-roomi.org/benchmarks/unconstrained/n-dimensions/172-generalized-penalized-function-no-1.

[5] “Penalty Functions.” Available: https://web.stanford.edu/group/sisl/k12/optimization/MO-unit5-pdfs/5.6penaltyfunctions.pdf.

[6] Y. Ji et al., “An Adaptive Chaotic Sine Cosine Algorithm for Constrained and Unconstrained Optimization,” Complexity, vol. 2020, p. e6084917, Oct. 2020, doi: https://doi.org/10.1155/2020/6084917.

[7] J. Liu, S. Anavatti, M. Garratt, K. C. Tan, and H. A. Abbass, “A survey, taxonomy and progress evaluation of three decades of swarm optimisation,” Artificial Intelligence Review, vol. 55, no. 5, pp. 3607–3725, Nov. 2021, doi: https://doi.org/10.1007/s10462-021-10095-z.

[8] A. B. Gabis, Y. Meraihi, S. Mirjalili, and A. Ramdane-Cherif, “A comprehensive survey of sine cosine algorithm: variants and applications,” Artificial Intelligence Review, vol. 54, no. 7, pp. 5469–5540, Jun. 2021, doi: https://doi.org/10.1007/s10462-021-10026-y.

[9] L. Abualigah and A. Diabat, “Advances in Sine Cosine Algorithm: A comprehensive survey,” Artificial Intelligence Review, Jan. 2021, doi: https://doi.org/10.1007/s10462-020-09909-3.

[10] N. Nayak, S. Mishra, D. Sharma, and Binod Kumar Sahu, “Application of modified sine cosine algorithm to optimally design PID/fuzzy-PID controllers to deal with AGC issues in deregulated power system,” Iet Generation Transmission & Distribution, vol. 13, no. 12, pp. 2474–2487, Jun. 2019, doi: https://doi.org/10.1049/iet-gtd.2018.6489.

[11] Y. Celik, “Solving the Tension/Compression Spring Design Problem by an Improved Firefly Algorithm.” Available: https://ceur-ws.org/Vol-2255/paper2.pdf.

[12] A. Tzanetos and M. Blondin, “A qualitative systematic review of metaheuristics applied to tension/compression spring design problem: Current situation, recommendations, and research direction,” Engineering Applications of Artificial Intelligence, vol. 118, p. 105521, Feb. 2023, doi: https://doi.org/10.1016/j.engappai.2022.105521.

[13] C. Qu, Z. Zeng, J. Dai, Z. Yi, and W. He, “A Modified Sine-Cosine Algorithm Based on Neighborhood Search and Greedy Levy Mutation,” Computational Intelligence and Neuroscience, vol. 2018, p. e4231647, Jul. 2018, doi: https://doi.org/10.1155/2018/4231647.

[14] Wikipedia Contributors, “Logistic map,” Wikipedia, Oct. 11, 2019. https://en.wikipedia.org/wiki/Logistic_map

[15] T. Si and D. Bhattacharya, “Sine Cosine Algorithm with Centroid Opposition-Based Computation,” Algorithms for intelligent systems, pp. 119–129, Jan. 2021, doi: https://doi.org/10.1007/978-981-33-4604-8_9.

[16] M. Bin, M. Tasir, and P. Darul, “OPTIMAL DESIGN OF HELICAL COMPRESSION SPRINGS,” 2016. [Online]. Available: https://utpedia.utp.edu.my/id/eprint/16909/1/DISSERTATION%2016418%20Muhammad%20Hakim%20B%20Mat%20Tasir.pdf.

[17] “Progress Rate Analysis of Evolution Strategies on the Rastrigin Function: First Results,” springerprofessional.de. https://www.springerprofessional.de/en/progress-rate-analysis-of-evolution-strategies-on-the-rastrigin-/23366296.

[18] Y. Huang, J. Li, and P. Wang, “Unusual phenomenon of optimizing the Griewank function with the increase of dimension,” Frontiers of Information Technology & Electronic Engineering, Dec. 2019, doi: https://doi.org/10.1631/fitee.1900155.

[19] M. Jamil and X. S. Yang, “A literature survey of benchmark functions for global optimisation problems,” International Journal of Mathematical Modelling and Numerical Optimisation, vol. 4, no. 2, p. 150, 2013, doi: https://doi.org/10.1504/ijmmno.2013.055204.

[20] “This equation will change how you see the world (the logistic map),” www.youtube.com. https://www.youtube.com/watch?v=ovJcsL7vyrk&t=617s&ab_channel=Veritasium

[21] C. Qu, Z. Zeng, J. Dai, Z. Yi, and W. He, “A Modified Sine-Cosine Algorithm Based on Neighborhood Search and Greedy Levy Mutation,” Computational Intelligence and Neuroscience, vol. 2018, p. e4231647, Jul. 2018, doi: https://doi.org/10.1155/2018/4231647.

[22] A. Fatlawi, A. Vahedian, and N. K. Bachache, “Optimal Camera Placement Using Sine-Cosine Algorithm,” IEEE Xplore, Oct. 01, 2018. https://ieeexplore.ieee.org/document/8566344

[23] Mohamed Abd Elfattah, Sherihan Abuelenin, Aboul Ella Hassanien, and J.-S. Pan, “Handwritten Arabic Manuscript Image Binarization Using Sine Cosine Optimization Algorithm,” Advances in intelligent systems and computing, pp. 273–280, Nov. 2016, doi: https://doi.org/10.1007/978-3-319-48490-7_32.

[24] J. M. Altmemi, R. R. Othman, R. Ahmad, and A. S. Ali, “Implementation of Sine Cosine Algorithm (SCA) for Combinatorial Testing,” IOP Conference Series: Materials Science and Engineering, vol. 767, p. 012009, Mar. 2020, doi: https://doi.org/10.1088/1757-899x/767/1/012009.

[25] X. Ye, Z. Cai, C. Lu, H. Chen, and Z. Pan, “Boosted Sine Cosine Algorithm with Application to Medical Diagnosis,” Computational and Mathematical Methods in Medicine, vol. 2022, p. e6215574, Jun. 2022, doi: https://doi.org/10.1155/2022/6215574.

[26] “Ackley function,” Wikipedia, Oct. 19, 2019. https://en.wikipedia.org/wiki/Ackley_function