IMAT 5232

Computational Intelligence Optimisation

Interim Assessment Two

Authored by

Adam Leonard Hubble

P17175774

P17175774@my365.dmu.ac.uk

Table of Contents

Introduction 2

Benchmark Method Implementation 3

De Jong *3*

Schwefel *4*

Rastrigin *6*

Michalewicz *8*

Miscellaneous Method Implementation 10

Random Solution Generation *10*

Solution Correction *12*

Single-solution Optimiser Implementation 15

Short Distance Exploration *15*

Evaluation 20

De Jong *20*

Schwefel *22*

Rastrigin *24*

Michalewicz *26*

Conclusion 29

References 30

Appendices 32

Introduction

Described as the purpose and criteria of this interim assessment, it was expected that four established testbed problems were to be implemented for addressing algorithmic optimisation; the testbed problems are recognised as De Jong (sphere) [1], Schwefel [2], Rastrigin [3] and Michalewicz [4], which collectively and alongside the already provided Ackley [5], Alpine [6] and Rosenbrock [7] testbed problems, form the benchmark used to test and evaluate a single-solution optimisation algorithms performance. In addition to the testbed problems mentioned and to accompany the anticipated implementation of an elected single-solution optimiser, a series of miscellaneous, overloaded methods [8] were expected to be implemented also; said methods were functionally projected to enable an optimisation solution to be randomly generated (initial guess) within the bounds of a focused decision space and to be corrected to the space where required. In which, solution correction (toroidal) was instructed to maintain a solutions presence within a defined set of boundaries (the search space), by counteracting displacements that can be caused when an optimiser performs series of variable perturbation (optimisation) cycles.

Aside from each of the functional implementations listed, upon testing the nominated single-solution optimiser over the abovementioned benchmark, a series of numerical results and statistic were expected to be gathered, presented, and interpreted, to identify the chosen optimisers performance and thus capability to be a suitable optimisation candidate for the problems entertained by the assessment instructor. Given this nature and although not explicitly stated, an optimisation performance comparison between the elected optimiser and provided Intelligent Single Particle

Optimisation (ISPO) [9] and Covariance Matrix Adaptation Evolutionary Strategy (CMEAS) [10] optimisers was anticipated as well. All of such is explored in the proceeding passages.

Benchmark Method Implementation

De Jong

Featured as one of the simplest problems in the benchmark derived, De Jong's sphere function [1] can be characterised as a continuous, "unimodal and convex" [11] problem, in which there exists only one local minimum, also recognised as the global minimum $f(x) = 0$, when $x = (0, \dots, 0)$ for $D \subseteq \mathbb{R}^n$. The function can be "defined on any input domain" [12], however it is often evaluated on the hypercube that is:

$$x_i \in [-5.12, 5.12] \text{ for } i = 1 \dots n \text{ or } -5.12 \leq x_i \leq 5.12 \text{ for } i = 1 \dots n$$

Hence the boundaries configured for the function are '-5.12', representing the lower bound of the problems search space and '5.12', representing the upper bound of the problems search space. Furthermore, the function is undoubtedly differentiable and separable, which determines that "a sequence of n independent optimization processes can be performed" [13], this is given by the functions support for variable independence in n dimensions; the general implication of separable functions is that convergence can be achieved sooner when compared to inseparable functions, as they present to be "relatively easy to solve", given that solutions can be "obtained independently of all the other parameters" involved.

De Jong (Sphere) Problem Method Implementation

```
. . .
public Sphere(int dimension){ super(dimension, new double[] {-5.12, 5.12}); }
. . .
public double f(double[] x)
{
        . . .
        else
        {
                for (int i = 0; i < n; i++)
                        y += Math.pow(x[i], 2);
        }
        . . .
}
. . .
```

For the implementation of De Jong's sphere method, the fitness value of a passed solution can be resolved by incrementally summing each of its design variable values, squared, for the dimensionality of the problem domain; this is orchestrated using a 'for-loop', which enables each design variable in the solution (a one-dimensional array) to be iterated through for performing the arithmetic described. This operation can be formally notated as:

$$f(x) = \sum_{i=1}^{n} x_i^2, \qquad -5.12 \leq x_i \leq 5.12$$

Notably, this function block is only executed if the dimensionality of the solution x, is equivalent to the dimensionality of the problem domain n. Otherwise, the fitness value $f(x)$ returned by the method is equivalent to null (∅), due to the incompatibility in variable and problem dimensionality; this is programmatically communicated via console output and is structured by an 'if-else' statement.

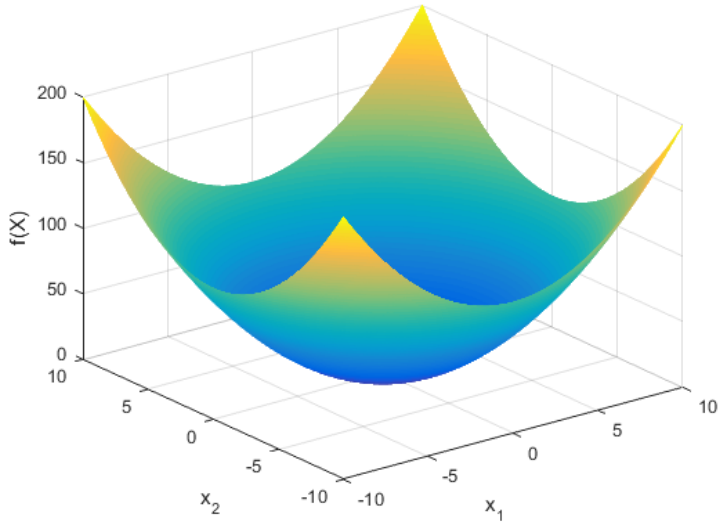

Figure 1: Plot visualization of De Jong's sphere benchmark problem [12].

Schwefel

Submitted as the secondary problem in the benchmark derived, Schwefel's 2.26 function [2] can similarly be characterised as a continuous problem, however, it is also "non-convex" and "multimodal" [13], for which the functions fitness landscape exhibits high multi-modality. This is satisfied by the "number of ambiguous peaks in the function landscape", implying that algorithms relying upon gradient-based information for determining their search direction could easily converge to a suboptimal basin of attraction in said landscape. If an "algorithm encounters these peaks during a search process, there is a tendency that the algorithm may be trapped in one of such peaks" that are formerly recognised as local optima, which mostly provide an unsatisfactory performance with respect to the function's fitness value $f(x)$ potential. Inevitably, the problem presents a "negative impact on the search process" for establishing the global optimum (minimum) of the function, through directing "the search away from the true optimal solution".

The global minimum of the problem $f(x) = 0$ is feasible when $x = (420.9687, \dots, 420.9687)$ for $D \subseteq \mathbb{R}^n$, similarly to De Jong's sphere function, the function can be "defined on any input domain" [15] but is typically evaluated on the hypercube that is:

$$x_i \in [-500, 500] \text{ for } i = 1 \dots n \text{ or } -500 \leq x_i \leq 500 \text{ for } i = 1 \dots n$$

Thus, the boundaries configured for the function are '-500' and '500', representing the lower bound and upper bounds of the problems search space (domain). Moreover, also like De Jong's sphere function, Schwefel's 2.26 function is differentiable and separable which implies that convergence can be achieved relatively quickly as previously explored; this is further heightened by the high multi-

modality of the function's fitness landscape, given that an algorithm may become trapped in one of many local minima, especially if said algorithm identifies as a single-solution deterministic metaheuristic that employs deterministic local search (search in the surrounding areas of a solution).

Schwefel 2.26 Problem Method Implementation

```
. . .
public Schwefel(int dimension) { super(dimension, new double[] {-500, 500}); }
. . .
public double f(double[] x)
{
	final int n = x.length;
	double sum = 0;
	double y = 0;

	if (this.getDimension() != n)
	{
		y = Double.NaN;
		System.out.println("WARNING: the design variable does not match the dimensionality of the problem!");
	}
	else
	{
		for (int i = 0; i < n; i++)
			sum += x[i] * Math.sin(Math.sqrt(Math.abs(x[i])));
	}

	y = 418.9829 * n - sum;
	. . .
}
. . .
```

For the implementation of Schwefel's 2.26 method, the fitness value of a passed solution can initially be gathered by incrementally summing each of its design variable values, that are multiplied by the sine of the square root of each design variables absolute value; this arithmetic is possible by utilising a 'for-loop' that iteratively accesses each design variables value in the solution passed, via indexing, for the dimensionality of the problem domain. Proceeding from this summation, a constant value is multiplied by the dimensionality of the problem, before being deducted by the summed total acquired in the iterative procedure, which resultingly calculates the fitness value of the solution. This operation can be formally notated as:

$$f(x) = 418.9829n - \sum_{i=1}^{n} x_i sin\left(\sqrt{|x_i|}\right), \qquad -500 \leq x_i \leq 500$$

Like De Jong's sphere function implementation, this function block is only executed if the dimensionality of the solution x, is equivalent to the dimensionality of the problem domain n. Otherwise, the fitness value $f(x)$ returned by the method is equivalent to null (∅), due to the incompatibility in variable and problem dimensionality; this too, is programmatically communicated via console output and is structured by an 'if-else' statement.

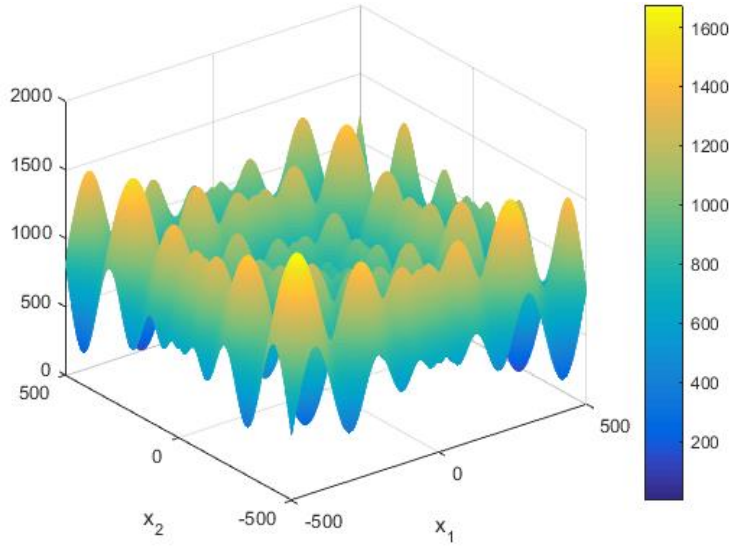

Figure 2: Plot visualisation of Schwefel's 2.26 benchmark problem [14].

Rastrigin

Included as the tertiary problem in the benchmark derived, Rastrigin's function [3] can also be characterised as a continuous and "highly multimodal" [11] problem, that also shares the property of convexity [15] alike De Jong's sphere function. Given its high multi-modality which extends the modality of Schwefel's 2.26 functions fitness landscape, it is increasingly likely that algorithms relying upon gradient-based information for determining their search direction would converge to a suboptimal basin of attraction in the landscape (a local minimum); thus, providing an unsatisfactory performance with respect to the function's fitness value $f(x)$ potential. As previously acknowledged, Rastrigin's problem also presents a negative impact on the search process for establishing the global optimum of the function, through directing the search away from the true optimal solution [13], similarly to Schwefel's 2.26 function but even more so.

The global minimum of the problem $f(x) = 0$ is feasible when $x = (0, \dots, 0)$ for $D \subseteq \mathbb{R}^n$, like both problems explored prior, the function can be "defined on any input domain" [15] but is typically evaluated on the hypercube that is:

$$x_i \in [-5.12, 5.12] \text{ for } i = 1 \dots n \text{ or } -5.12 \leq x_i \leq 5.12 \text{ for } i = 1 \dots n$$

Therefore, the boundaries configured for the function are '-5.12', representing the lower bound of the problems search space and '5.12', representing the upper bound of the problems search space. Like both previously explored problems, Rastrigin's function is differentiable and separable too, which indicates that convergence can be achieved relatively quickly; given the high multi-modality of the function like that of Schwefel's 2.26 function, a hastened convergence is further promoted as an algorithm may become trapped in one of the functions many local minima. This is increasingly likely if the algorithm identifies as a local search function, whether its movement or perturbation operator (search) is stochastically or deterministically adjusted.

Rastrigin Problem Method Implementation

. . .

```
public Rastrigin(int dimension) { super(dimension, new double[] {-5.12, 5.12}); }

public Rastrigin(int dimension, double[] bounds) { super(dimension, bounds); }

public Rastrigin(int dimension, double[][] bounds) { super(dimension, bounds); }

public double f(double[] x)
{
	final int n = x.length;
	double sum = 0;
	double y = 0;

	if (this.getDimension() != n)
	{
		y = Double.NaN;
		System.out.println("WARNING: the design variable does not match the dimensionality of the problem!");
	}
	else
	{
		for (int i = 0; i < n; i++)
			sum += Math.pow(x[i], 2) - (10 * Math.cos(2 * Math.PI * x[i]));

		y = 10 * n + sum;
	}

	return y;
}
. . .
```

For the implementation of Rastrigin's method, the fitness value of a passed solution can initially be acquired by incrementally summing each of its design variables values, squared, before being subtracted by a multiplicative sequence comprising a constant '10', which is multiplied by the cosine of another multiplicative sequence that contains a constant '2', multiplied by π (pi) and the value of the currently iterated design variables value. Once again, the arithmetic of the problem is also assisted by the implementation of a 'for-loop', which is used to iteratively access each design variables value contained in the solution passed; this is achieved via array indexing, which recurs for the dimensionality of the problem domain, $D \subseteq \mathbb{R}^n$. Advancing from the arithmetic described, the same constant value '10' is then multiplied by the dimensionality of the problem, before being totalled with the summation calculated in the iterative procedure prior; this resultingly provides the fitness value of the solution and can be formally notated as:

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10cos(2\pi x_i)\right], \qquad -5.12 \leq x_i \leq 5.12$$

Like the previous problem implementations, this function block is only executed if the dimensionality of the solution x, is equivalent to the dimensionality of the problem domain n. Otherwise, the fitness value $f(x)$ returned by the method is equivalent to null (∅), due to the incompatibility in variable and problem dimensionality; this is programmatically communicated via console output and is structured by an 'if-else' statement.

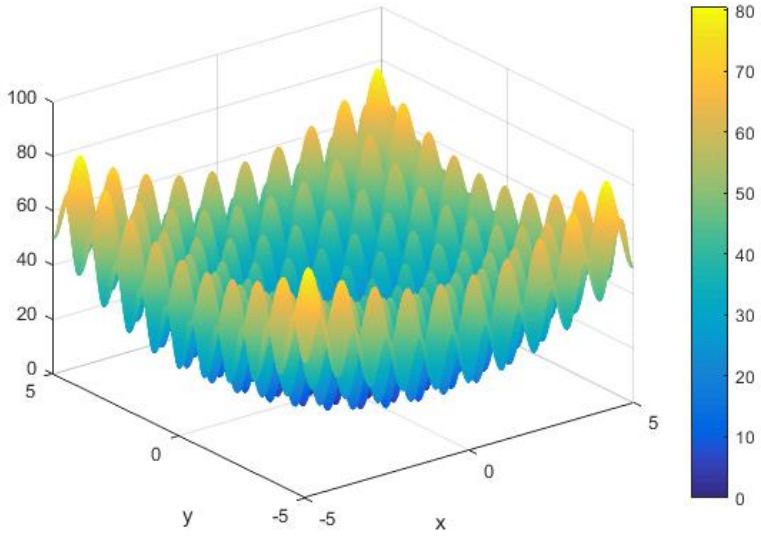

Figure 3: Plot visualisation of Rastrigin's benchmark problem [16].

Michalewicz

Featured as the quaternary problem in the benchmark derived, Michalewicz's function [4] can be characterised as a continuous, non-convex and "multimodal" [16] problem, in which the functions fitness landscape exhibits low multi-modality that presents a series of "valleys and ridges" as opposed to "ambiguous peaks" [13], which proportionately generates n local minima within the decision space. Given the nature of the valleys and ridges comprising the functions modality, it is likely that algorithms relying upon gradient-based information for determining their search direction would converge to a suboptimal basin of attraction in the landscape (a local minimum), given that the plateau regions (ridges) enhance the difficulty of the search by returning no improvement in function fitness value $f(x)$, as well, "an algorithm may be slowed down considerably on the floor of the valley", due to regions of steep descent. Whereby, given the steepness m (parametric) of said valleys and ridges, "a larger m leads to a more difficult search" [16], as the regions of steep descent increase the likelihood of local minima entrapment and slowed search progression; thus, it would be expected for an unsatisfactory performance with respect to the function's fitness value $f(x)$ potential, to be returned in this relation. Alike the Schwefel 2.26 and Rastrigin functions, it is inevitable that the Michalewicz's problem also presents a negative impact on the search process, for establishing the global optimum of the function, through which the search is directed away from the true optimal solution [13].

Uniquely, alongside Michalewicz's function presenting n local minima within the decision space, the global minimum of the problem for n dimensions dynamically depreciates as the dimensionality of the problem increases. In which, the global minimum of the problem $f(x) = -1.8013$ is feasible when $x = (2.20319, 1.57049)$ for $D \subseteq \mathbb{R}^2$, $f(x) = -4.687658$ is feasible for $D \subseteq \mathbb{R}^5$ and $f(x) = -9.66015$ is feasible for $D \subseteq \mathbb{R}^{10}$ [11][17]. The function is typically evaluated on the hypercube that is:

$$x_i \in [0, \pi] \text{ for } i = 1 \dots n \text{ or } -0 \leq x_i \leq \pi \text{ for } i = 1 \dots n$$

Hence the boundaries configured for the function are '0', representing the lower bound of the problems search space and 'π', representing the upper bound of the problems search space. Like all

problems explored, Michalewicz's function is differentiable and separable also, which implies that convergence can be achieved relatively quickly; given the functions regions of steep descent, a fastened convergence is further promoted as an algorithm may become trapped in one of the functions local minima. This is ever more likely if the algorithm identifies as a local search function.

Michalewicz Problem Method Implementation

```
. . .
public Michalewicz(int dimension) { super(dimension, new double[] {0, Math.PI}); }

public Michalewicz(int dimension, double[] bounds) { super(dimension, bounds); }

public Michalewicz(int dimension, double[][] bounds) { super(dimension, bounds); }

public double f(double[] x)
{
	final int n = x.length;
	int m = 10;
	double sum = 0;
	double y = 0;

	if (this.getDimension() != n)
	{
		y = Double.NaN;
		System.out.println("WARNING: the design variable does not match the dimensionality of the problem!");
	}
	else
	{
		for (int i = 0; i < n; i++)
			sum += Math.sin(x[i]) * Math.pow(Math.sin((i + 1) * Math.pow(x[i], 2) / Math.PI), 2 * m);

		y = -sum;
	}

	return y;
}
. . .
```

For the implementation of Michalewicz's function, the fitness value of a passed solution can initially be calculated by incrementally summing the sine of each of its design variables values, multiplied by the sine of a divisive sequence containing a constant 'i + 1' (zero indexed), representing the index or dimension of the solution, which is multiplied by the value of the design variable at the corresponding index, squared, before being divided by π; the sine product of this divisive operation is then raised to the power of the product between a constant '2', multiplied by the steepness m of the functions valleys and ridges. Alike all other problem implementations, the arithmetic of Michalewicz's problem is also assisted by the insertion of a 'for-loop', which is used to iteratively access each design variables value contained in the solution passed, for the dimensionality of the problem domain $D \subseteq \mathbb{R}^n$. Progressing from the arithmetic described, the summation obtained from the iterative procedure is then negated to provide the fitness value of the solution; this problems implementation can also be understood formally, with reference to the following notation:

$$f(x) = -\sum_{i=1}^{n} \sin(x_i) \cdot \left[\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right]^{2 \cdot m}, \qquad -0 \leq x_i \leq \pi$$

$$Where\ m = 10$$

Identical to all other problem implementations, this function block is only executed if the dimensionality of the solution x, is equivalent to the dimensionality of the problem domain n. Otherwise, the fitness value $f(x)$ returned by the method is equivalent to null ($\emptyset$), due to the incompatibility in variable and problem dimensionality; this is programmatically communicated via console output and is structured by an 'if-else' statement.

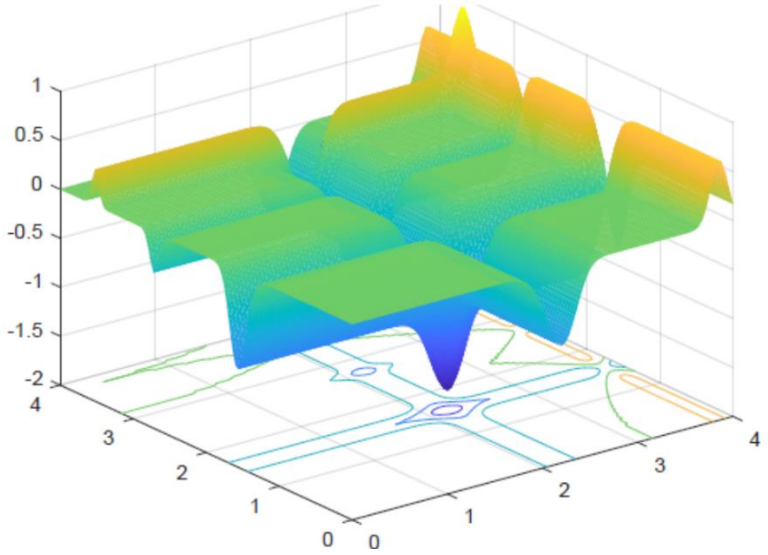

Figure 4: Plot visualisation of Michalewicz's benchmark problem, two-dimensional [17].

Miscellaneous Method Implementation

Random Solution Generation

Featured as a miscellaneous utility method within the 'Misc' class of the Stochastic Optimisation Software (SOS) platform [18], the random solution generation methods given by name, are purposed for arbitrarily producing an initial guess (solution) of n dimensionality for a given problem domain, that the elected algorithm would then optimise the values of (design variables), for establishing the global optimum (point) within the problem's decision space; a separate solution is randomly generated per testbed problem implemented. Being the starting point of the elected algorithms search, the methods ensure that every randomly generated value assigned to a design variable in the solution, is located within the boundaries of each testbed problems search space; this is necessary for respecting the problem domain that a solution is being optimised for.

generateRandomSolution Method Implementation (Unique Search Space Boundaries)

```
. . .
public static double[] generateRandomSolution(double[][] bounds, int n)
{
	double[] r = new double[n];

	for (int i = 0; i < n; i++)
		r[i] = bounds[i][0] + (bounds[i][1] - bounds[i][0]) * RandUtils.random();

	return r;
}
```

```
. . .
```

For addressing both unique (generic) and shared decision spaces for the design variables of a solution, two overloaded methods were implemented, for which a unique boundary method exists to enable every design variable to operate within a distinct search space, and a shared boundary method exists to conversely confine every design variable to share the same search space. This is programmatically acknowledged by each method's parametric requirements, given that the unique boundary method requires a two-dimensional array (boundaries) and integer variable (problem dimensionality), whereas the shared boundary method requires a one-dimensional array and integer variable; the parameters required by each of these methods are passed-by-value [19].

generateRandomSolution Method Implementation (Shared Search Space Boundaries)

```
. . .
public static double[] generateRandomSolution(double[] bounds, int n)
{
        double[] r = new double[n];

        for (int i = 0; i < n; i++)
                r[i] = bounds[0] + (bounds[1] - bounds[0]) * RandUtils.random();

        return r;
}
. . .
```

For either methods implementation, an empty one-dimensional double array is initialised for n dimensionality of a given problem, which exists to compactly facilitate the value of each design variable (a point) comprising the solution. Assisted by a 'for-loop', each index of the solution array is incrementally updated with a random number generated within the bounds of the given problem's decision space(s), this operation can be understood with reference to the following notation:

$$x^L + (x^U - x^L) \cdot \mathbb{R} \in [0, 1]$$

Where x^L is the lower bound(s) of the problems search space, x^U is the upper bound(s) of the problems search space and $\mathbb{R} \in [0, 1]$ is a randomly generated number (real), existing between the bounds of zero and one. Programmatically, the 'bounds' array containing the upper and lower boundaries of a problem's decision space can be indexed accordingly, to access either boundary value for performing the arithmetic shown, for which the first index '0' returns the value of the lower bound and the second index '1' returns the value of the upper bound; this operation is compatible for either dimensionality of array, however, the two-dimensional variant must be iteratively indexed to access the boundaries designated to each design variable in the solution. For randomly generating a value between the bounds of zero and one, the 'random' method provided in the 'RandUtils' class is invocated, per its description, the method returns a random, uniformly distributed double value between zero and one. Thus, to populate the value of each design variable comprising a problem solution, the lower bound value of the problem dimension is summed with the difference between the upper and lower boundary values of the problem dimension, before being multiplied by a randomly generated value; this arithmetic iteratively generates a solution within the boundaries of a problem domain, which is returned by either method for algorithmic optimisation.

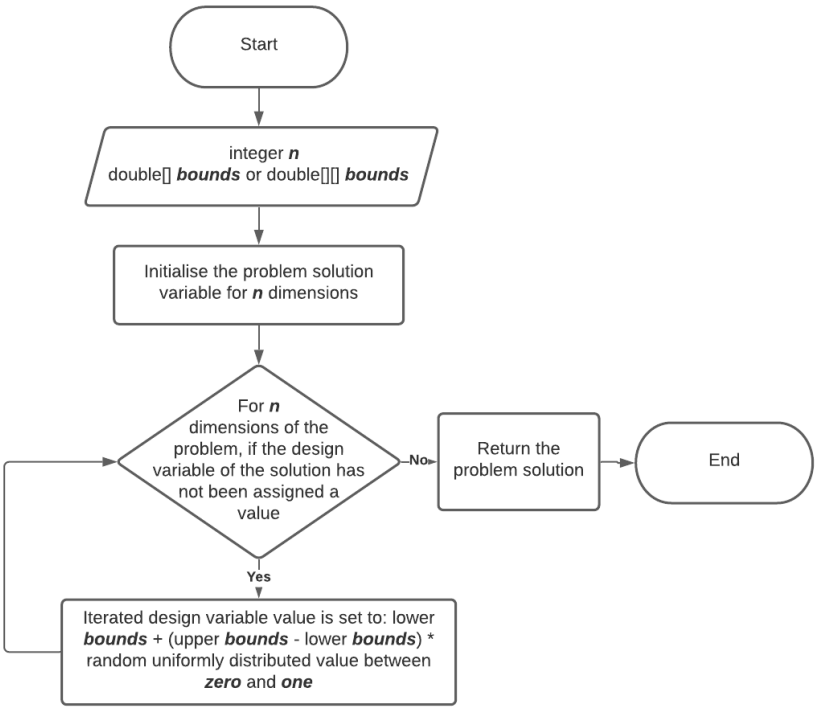

Figure 5: Flow chart visualisation, random solution generation method(s) process.

Solution Correction

Implemented as a miscellaneous utility method within the 'Misc' class of the SOS platform, solution boundary correction of type toroidal specifically, is purposed for correcting a solution to the boundaries of the decision space of a given optimisation problem, upon being displaced out of said space in effect of design variable perturbation. Given the toroidal nature of solution correction, if a solution is displaced outside of a problem's domain, the solution is corrected to a position in the decision space that is relative to its displacement outside the space, alike a torus, "the edge on one side can be thought of as being wrapped around to the opposite edge" [20]. Thereby, points which pose to be beyond the lower bound of a given problem's decision space, are corrected to the upper bound of the decision space and conversely, points which pose to be beyond the upper bound of the problem's decision space, are corrected to lower bound of the decision space. The following example exercises this correction mechanism:

Toroidal solution correction			
x^L	x^U	x	x^C
1	5	7	3
0	3	-1	2

Table 1: Toroidal solution correction mechanism exemplification.

Where x^L represents the lower bound value of the problems search space, x^U represents the upper bound value of the problems search space, x represents the solution and x^C also represents the solution, but when toroidally corrected.

toro (Toroidal) Method Implementation (Shared Search Space Boundaries)

```
. . .
public static double[] toro(double[] x, double[] bounds)
{
	int n = x.length;
	double[] x_tor = new double[n];

	for (int i = 0; i < n; i++)
	{
		x_tor[i] = (x[i] - bounds[0]) / (bounds[1] - bounds[0]);

		if (x_tor[i] > 1)
		{
			x_tor[i] = x_tor[i] - fix(x_tor[i]);
		}
		else if (x_tor[i] < 0)
		{
			x_tor[i] = 1 - Math.abs(x_tor[i] - fix(x_tor[i]));
		}

		x_tor[i] = x_tor[i] * (bounds[1] - bounds[0]) + bounds[0];
	}

	return x_tor;
}
. . .
```

Alike when addressing the random solution generation functionality, solution correction is also addressed by both unique (generic) and shared decision spaces for the design variables of a solution, for which two overloaded methods were implemented to cater for; this allows a unique boundary method to exist for enabling every design variable of a solution to be corrected to a distinct search space, and a shared boundary method to exist for conversely correcting every design variable of a solution to the same search space. This is programmatically acknowledged by each method's parametric requirements, given that the unique boundary method requires a two-dimensional array (boundaries) and integer variable (problem dimensionality), whereas the shared boundary method requires a one-dimensional array and integer variable; as previously acknowledged, the parameters required by each of these methods are passed-by-value.

toro (Toroidal) Method Implementation (Unique Search Space Boundaries)

```
. . .
public static double[] toro(double[] x, double[][] bounds)
{
	int n = x.length;
	double[] x_tor = new double[n];

	for (int i = 0; i < n; i++)
	{
		x_tor[i] = (x[i] - bounds[i][0]) / (bounds[i][1] - bounds[i][0]);

		if (x_tor[i] > 1)
		{
			x_tor[i] = x_tor[i] - fix(x_tor[i]);
		}
		else if (x_tor[i] < 0)
		{
			x_tor[i] = 1 - Math.abs(x_tor[i] - fix(x_tor[i]));
		}
```

```
                x_tor[i] = x_tor[i] * (bounds[i][1] - bounds[i][0]) + bounds[i][0];
        }

        return x_tor;
}
. . .
```

For either methods implementation, an empty one-dimensional double array is initialised for n dimensionality of a given problem, which exists to compactly contain the value of each design variable (a point) comprising the solution. Assisted by a 'for-loop', each point comprising the solution array is corrected to the bounds of the given problem's decision space(s), initially each point is normalised to exist relative to the boundaries of the problem domain, where a point can then be determined inside or outside of the domain of the problem; this is achieved by deducting the problem domains lower bound value from the currently iterated points value, before being divided by the difference between the boundary values of the problem domain, which returns a value relative to the range of zero (lower bound) and one (upper bound), representing the length of the domain. In determination of the arithmetic approached to correct the solution to the domain of the problem, an 'if-else' statement is utilised, this orchestrates the operation performed on the design variables value subject to conditioning; if no correction is required, given that the point already exists within the corresponding search space, the original solution is simply returned as the original one-dimensional double array.

However, if the normalised points value is greater than the value of one and thereby exists outside of the upper bound of the relevant decision space, the design variables value is deducted by itself, but rounded down to the nearest integer value; this is functionally addressed by invocating the 'fix' method that coexists in the 'Misc' class of the SOS platform, which rounds values down (floors) if they are larger or equal to zero and rounds values up (ceils) if they are smaller than zero. Oppositely, if the normalised points value is smaller than the value of zero and thereby exists outside of the lower bound of the relevant decision space, the absolute value of the design variables value deducted by itself, that is rounded up to the nearest integer value, is subtracted from the value of one to return the normalised distance from the domain's upper boundary. Proceeding from either operational block, the resultant value of the point is rescaled by the length of the domain, via denormalization, to restore the points relativity to the problem's decision (coordinate) space $D \subseteq \mathbb{R}$. Upon all the design variables being iterated through in the solution, the corrected solution is then returned by the methods as a one-dimensional double array, for which all its variables reside within the decision space(s) of the problem given.

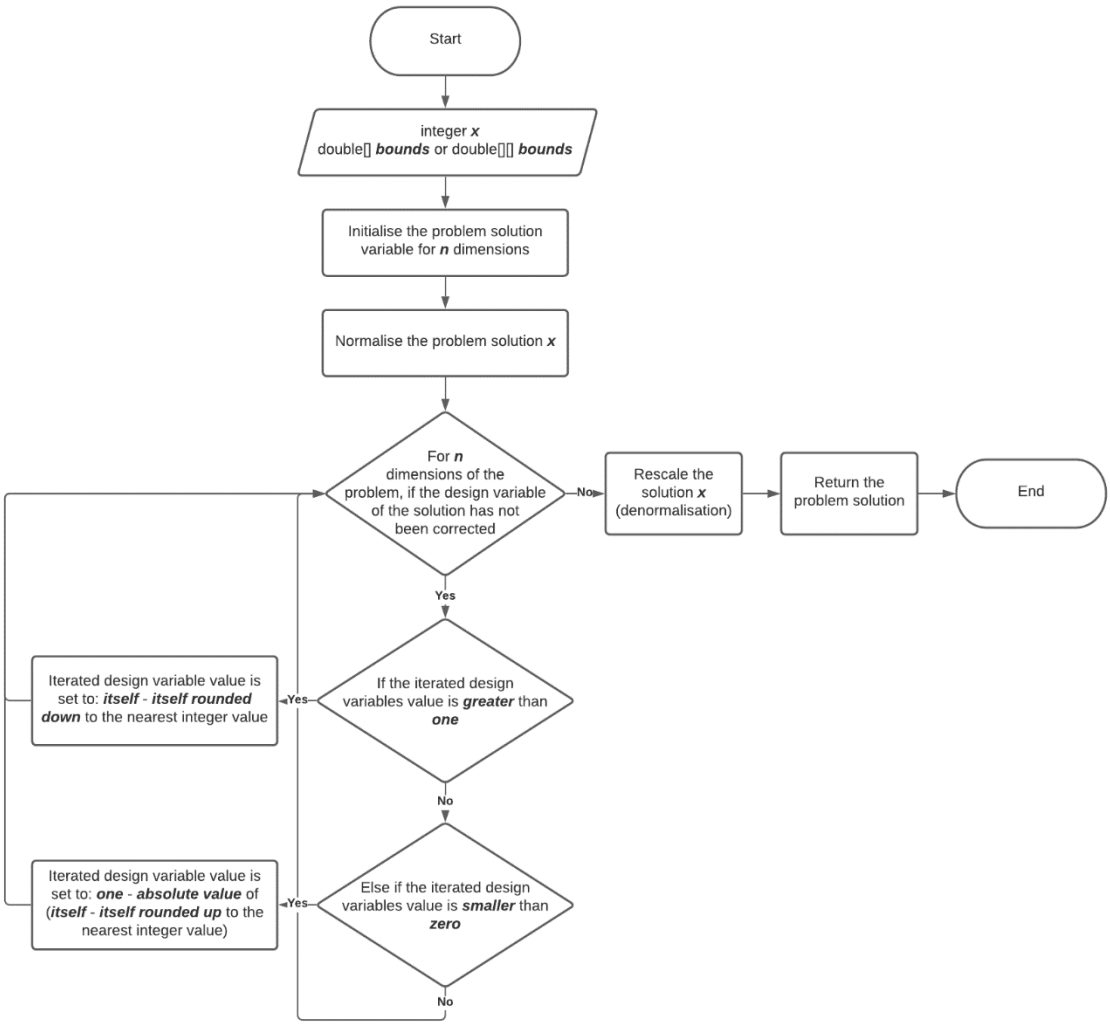

Figure 6: Flow chart visualisation, toroidal solution correction method(s) process.

Single-solution Optimiser Implementation

Short Distance Exploration

Short Distance Exploration (S) is a single-solution deterministic metaheuristic algorithm, which is recognised as a "greedy-descent method" [21] that employs a "perturbation logic derived from Hooke-Jeeves" Method proposed in 1961 [22], to perturb and resultingly optimise the values of each design variable comprising a solution of n dimensionality, for a given problem domain $D \subseteq \mathbb{R}^n$; at which the algorithm converges to the most convenient solution, at the end of the greedy, exploratory search in the domain. Given that S behaves as a "simple steepest descent deterministic local search algorithm" [23], which operates with one generated trial solution to produce one candidate solution without any form of randomisation logic, S can produce the same candidate (elite) solution for the same input, trial solution, through performing a predictable operational sequence of design variable perturbations. Furthermore, as a metaheuristic, convergence to the global optimum of a problem function is not guaranteed, for which S as a local search algorithm, is best recognised for operating on separable problems given that it "perturbs the variables of the elite one by one", which enables it to be "very efficient at handling separable functions" [24] but "detrimental with non-separable problems".

Short Distance Exploration (S) Optimiser Implementation

```
. . .
public FTrend execute(Problem problem, int maxEvaluations) throws Exception
{
	double alpha = getParameter("p0");

	FTrend FT = new FTrend();
	int problemDimension = problem.getDimension();
	double[][] bounds = problem.getBounds();
	double[] xBest = new double[problemDimension];
	double fBest = Double.NaN;
	int k = 0;

	if (initialSolution != null)
	{
		xBest = initialSolution;
		fBest = initialFitness
	}
	else
	{
		xBest = generateRandomSolution(bounds, problemDimension);
		fBest = problem.f(xBest);
		FT.add(k, fBest);
		k++;
		//FT.add(k, fBest);
	}

	double fShort = fBest;

	double[] xShort = xBest;
	double[] exploratoryRadius = new double[problemDimension];

	for (int i = 0; i < problemDimension; i++)
	{
		exploratoryRadius[i] = alpha * (bounds[i][1] - bounds[i][0]);
	}

	while (k < maxEvaluations)
	{
		boolean improved = false;

		for (int i = 0; i < problemDimension && k < maxEvaluations; i++)
		{
			xShort[i] = xBest[i] - exploratoryRadius[i];
			xShort = toro(xShort, bounds);

			fShort = problem.f(xShort);
			k++;

			if (k % problemDimension == 0)
			{
				FT.add(k, fBest);
			}

			if (fShort <= fBest)
			{
				xBest[i] = xShort[i];

				fBest = fShort;
				//FT.add(k, fBest);
```

```
                    improved = true;
                }
                else if (k < maxEvaluations)
                {
                    xShort[i] = xBest[i] + (exploratoryRadius[i] / 2);
                    xShort = toro(xShort, bounds);

                    fShort = problem.f(xShort);
                    k++;

                    if (k % problemDimension == 0)
                    {
                        FT.add(k, fBest);
                    }

                    if (fShort <= fBest)
                    {
                        xBest[i] = xShort[i];

                        fBest = fShort;
                        //FT.add(k, fBest);

                        improved = true;
                    }
                    else
                    {
                        xShort[i] = xBest[i];
                    }
                }
            }

            if (improved == false)
            {
                for (int i = 0; i < exploratoryRadius.length; i++)
                {
                    exploratoryRadius[i] = exploratoryRadius[i] / 2;
                }
            }
        }

        finalBest = xBest;
        FT.add(k, fBest);

        return FT;
}
. . .
```

Summarily, the S algorithm "executes an asymmetric sequence" [21] of design variable perturbations along all function axes, for n dimensionality of a given problem; this "exploration move attempts to fully exploit promising search directions" [25] in the domain, to seek the basin of attraction that is "globally optimal". For each dimension, the algorithm perturbs the variable orthogonally along the relevant axis for a full step of its corresponding exploratory radius $\delta[i]$, if an improvement in the solutions fitness value is not registered from the perturbation, a "half step is performed on the opposite direction" [21] of the same axis. If no improvement is registered by any design variable after perturbation (iteration), the variables return to their values (point) pre-perturbation and the exploratory radius that each variable is associated with is then halved $\delta = \frac{\delta}{2}$ at the end of the iteration, for reducing the step size and the resultant exploration potential of the search, to enable the algorithm to exploit more promising areas of the search space (locally), in following perturbation

cycles (iteration); this continues until a condition on the procedures computational budget is met, which may pose constraints on the solution that the algorithm converges too, if insignificant in value.

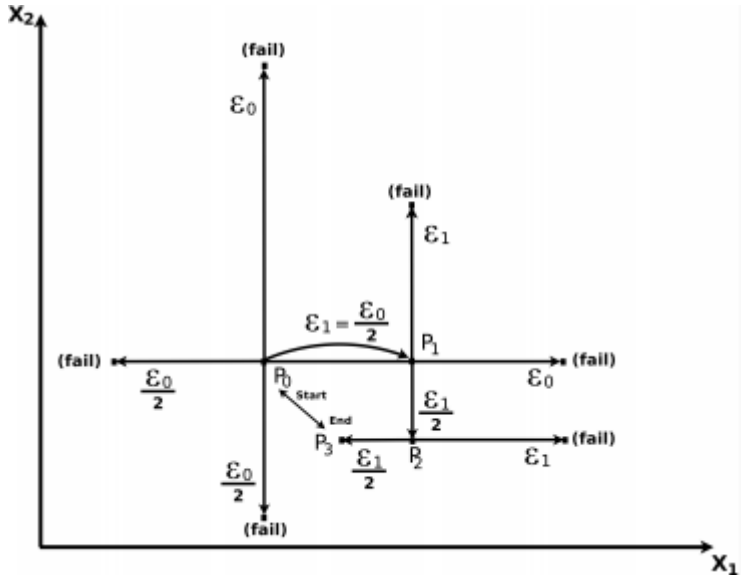

Figure 7: Graph-based visualisation of the Short Distance Exploration (S) algorithms perturbation cycling for a two-dimensional problem domain [21], and a given solution when the initial exploratory radius ε, is set to ε_0. Where P_0 represents the initial guess (start), P_3 represents the solution converged (end), P_1 represents the first successful perturbation operation (improvement), and $(fail)$ represents an unsuccessful perturbation operation (no improvement).

For the implementation of the S algorithm, the initial guess or elite solution is obtained either by another optimiser or is alternatively sampled from the problem domain by invocating the random solution generation method detailed prior; this is initialised as a one-dimensional double array of n dimensions, where the values assigned to each design variable comprising the solution, are subjected to the decision space boundaries passed to the method. A trial solution variable is also instantiated for the same specification, where it is initialised to the initial guess prior to any operation being performed. Additionally, the parametric value of the algorithm α, is simply initialised as a double variable that is assigned the value of '0.4', as recommended [26], which represents the exploratory radius δ or the alpha-cut of the problem domain's length, for the algorithms search procedure; the exploratory radius variable is initialised as a one-dimensional double array of n dimensions also, and with the support of a 'for-loop', each exploratory radii can be iteratively calculated as forty-percent of the length of each axis in the problem domain, in accordance with the 'bounds' array values that are accessed via indexing.

Proceeding from parameter initialisation, a 'while-loop' is approached for addressing the algorithms computational budget, which maintains the pre-configured value of '5000 * n' iterations for governing the algorithms perturbation logic. Therein, features multiple 'for-loop' declarations, where one of which exists to iterate through the solution and perturb each design variable along its corresponding axis in the problem domain given, whereas the other is purposed for iteratively halving the exploratory radii value of each design variable in the trial solution, upon the perturbation cycle registering no improvement on the solutions fitness value. For addressing the algorithms perturbation logic, upon a design variable being perturbed, the variable must be subjected to solution correction by invocating the appropriate toroidal correction method, for ensuring that the solutions optimisation remains relative to the problem domain; this is precautionary to potential displacements outside of the search space. Advancing from all perturbation operations, the trial solutions fitness value is then revaluated,

for which is used to identify an improvement in the company of 'if-else' statements, where given the scenario that an improvement is registered, the elite solution can then be updated to the design variable values of the trial solution; otherwise, when no improvement is registered, the relevant design variable is either re-perturbed along its corresponding axis but in the opposite direction at half the step, or it is restored to its value pre-perturbation, if no improvement persists to be registered. An improvement in the solutions optimality for a given problem is registered by a Boolean flag variable, which by instantiation is assigned the value 'false' and can be assigned the value 'true' upon the trial solution improving upon the elite; this variable is restored to the value of 'false' per n dimensions iterated through the trial solution, to enable the algorithm to balance exploration and exploitation appropriately, given that the variable is the determinant in whether the procedures search radius is minimised or maintained.

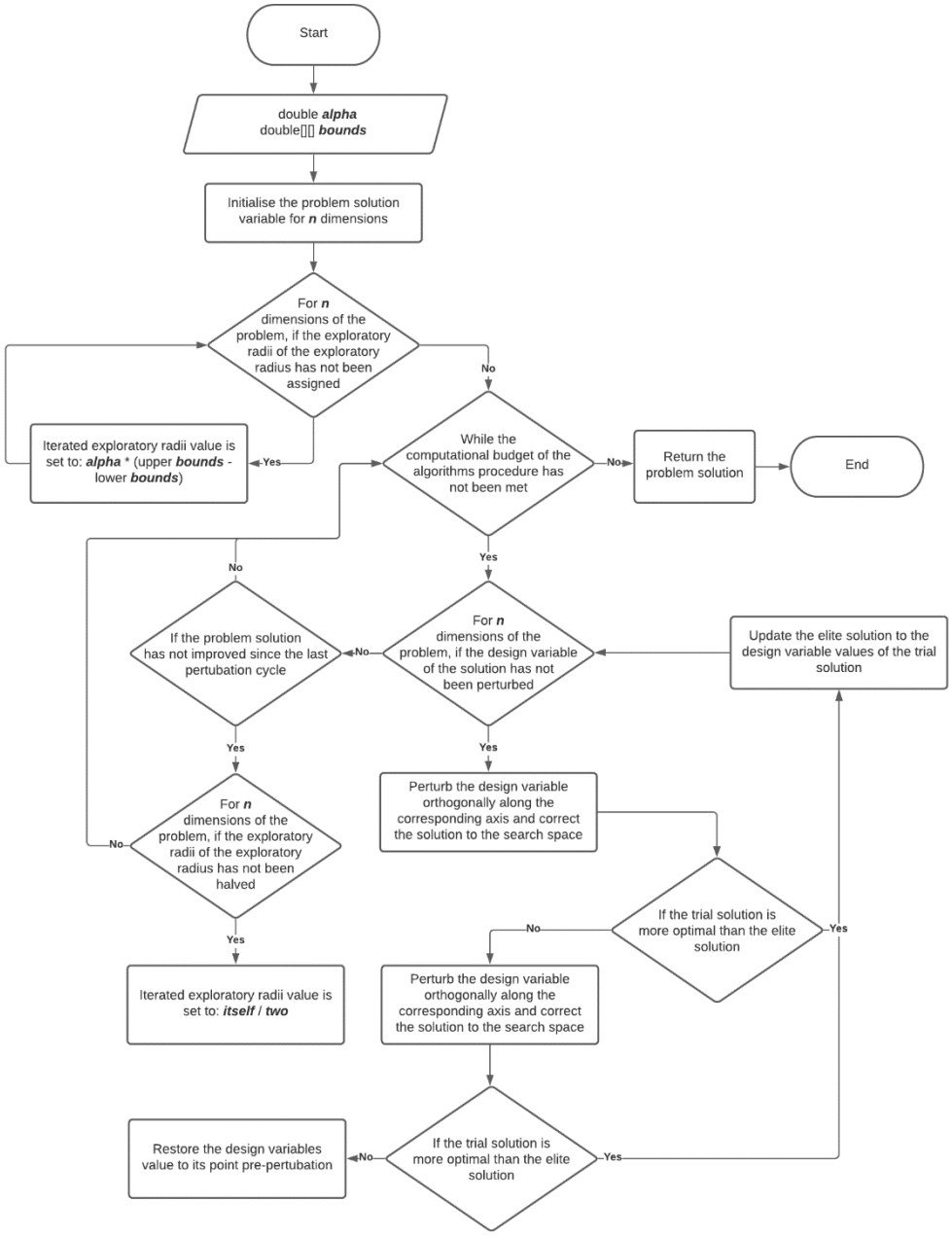

Figure 8: Flow chart visualisation, Short Distance Exploration (S) algorithm procedure.

Provided that many variants of the S algorithm exist, like that of the Three Stage Optimal Memetic Exploration (3Some) algorithm [24][25][27], where S features as one of “three operators” [24] representing the exploratory stages of the algorithm, the implementation of the very simple S algorithm presented, is presumed to be operationally inferior with respect to converging to the global minimum of a given problem domain, as it assumes an increased exploitative focus compared to its counterparts. This is exemplified within the 3Some variant alone, where “exploration is repeated for all the design variables and stopped when a prefixed budget (equal to 150 iterations) is exceeded” [25], as ‘150’ iterations is recognised to be “sufficient to reach a good precision” [21], which enables the algorithms exploratory radius to be reinitialised multiple times throughout the procedure, relative to the computational budget posed. Through reinitialising the exploratory radius of the procedure, the likelihood that the algorithm does not get trapped within a region of step-descent or local minima increases, as its explorative potential expands, thus it can be presumed more efficient than the very simple variant implemented.

Evaluation

De Jong

Upon the S algorithm being benchmarked over De Jong’s testbed problem for the array of problem dimensionality, that ranges from ‘5’ to ’50’ dimensions and for ‘10’ runs of each experiment, S with reference to the figure featured below, proves to reach the global optimum of the problem domain, for all dimensions listed; this is figured as for every dimension, the average fitness value $f(x)$ returned is ‘0’, where $f(x) = 0$ is recognised as the problems global minimum when $x = (0, \dots, 0)$ for $D \subseteq \mathbb{R}^n$. In relation to this problem function specifically, S is observed to be effective, given its consistency to establish the global minimum of the problem function in every dimension it is evaluated within; this behaviour is expected however, given the unimodality of the function’s fitness landscape, meaning that the algorithm cannot be trapped in a local minimum through search diversion, as its only minima is the global optimum.

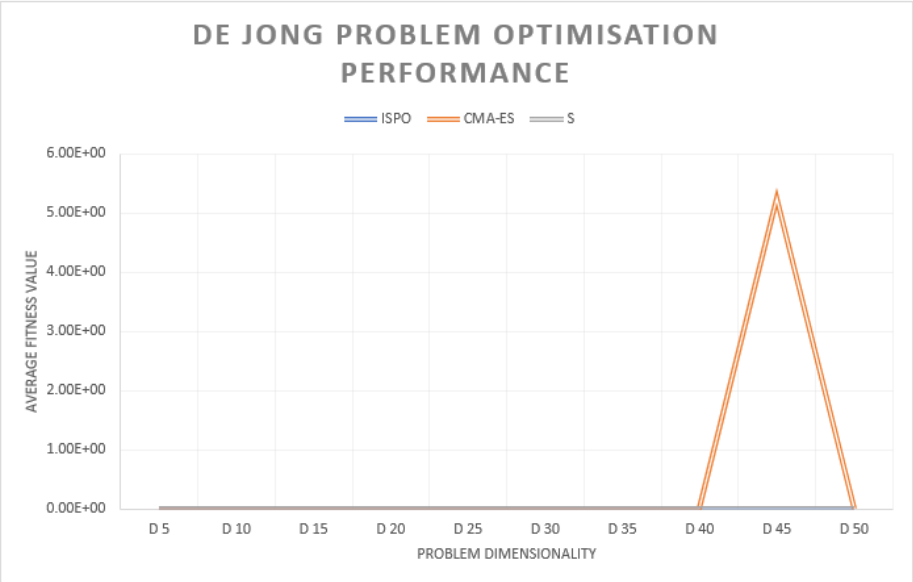

Figure 9: Graph-based visualisation of the optimisation performance of the featured optimiser algorithms, when benchmarked over the De Jong problem, for '10' runs per dimension tested. The dimension at which the average fitness

value of the solution is obtained for, is indicated along the 'x' axis of the above illustration, where the problem is evaluated for 5D, 10D, 15D, 20D, 25D, 30D, 35D, 40D, 45D and 50D.

When compared to both ISPO and CMA-ES algorithms, S is undoubtedly more effective, given that ISPO with reference to the tabularised results below, never converges with a mean fitness value of '0', regardless of the problem's dimensionality. Whereas CMA-ES sometimes does not converge to the global optimum either (outlying data) but is mostly capable; given the simplicity of the functions landscape, this observation is presumed to be a limitation of the allocated computational budget, for which the perturbation logic of S enables it to perform better within by comparison. Supporting the performance ranking described the Wilcoxon Rank-Sum test [21] identifies ISPO to be least optimal algorithm for the problem domain, in all but '45' dimensions where it shares a similar performance with CMA-ES; through means of comparison and statistical analysis, S can be argued the most optimal, given that it converges to the global optimum for all dimensions of the problem and for all runs, as can be determined by the mean fitness value and spread or deviation in fitness values (null) obtained across all '10' runs of the experiments.

Problem Dimensionality	ISPO	CMA-ES	W	S	W
5D	1.489e-35 ± 9.555e-36	6.747e-311 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
10D	5.378e-35 ± 2.360e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
15D	6.516e-35 ± 2.250e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
20D	1.120e-34 ± 4.523e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
25D	1.145e-34 ± 3.002e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
30D	1.727e-34 ± 6.947e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
35D	1.892e-34 ± 3.331e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
40D	2.442e-34 ± 8.327e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
45D	2.134e-34 ± 3.403e-35	5.243e+00 ± 1.049e+01	=	0.000e+00 ± 0.000e+00	-
50D	2.369e-34 ± 4.518e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-

Table 2: Results table for the three featured optimisers benchmarked over the De Jong problem, for 5D, 10D, 15D, 20D, 25D, 30D, 40D, 45D and 50D; this experiment was executed for '10' runs.

Although S presents variation within the number of iterations required to converge to the global optimum of the problems function, it is inevitably efficient, where variation within the algorithms convergence rate is subject to the initial guess (starting point) in the functions search space, as well as its perturbation logic, which increases the likelihood of S continually overstepping the global optimum at the functions promising basin of attraction. As can be noticed from the graphical representation of the algorithms convergence rate below, S on average, is able to converge to the global optimum of the problem function per '1187' iterations or perturbation cycles surpassed.

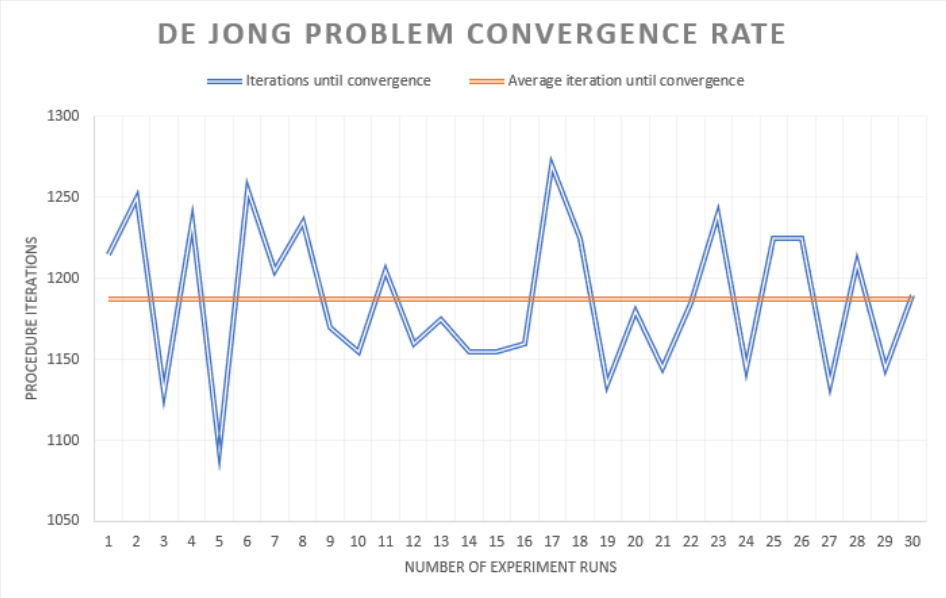

Figure 10: Graph-based visualisation of the convergence rate of the Short Distance Exploration (S) algorithm, when benchmarked over the De Jong problem, for a total of '30' runs when evaluated in 5D.

Schwefel

Upon the S algorithm being benchmarked over Schwefel's testbed problem for the same array of problem dimensionality, that ranges from '5' to '50' dimensions and for '10' runs of each experiment, S with reference to the figure featured below, proves to never reach the global optimum of the problem domain, for all dimensions listed; this is figured as for every dimension, the average fitness value $f(x)$ returned is greater than '0', where parallel with the dimensionality of the problem increasing, the fitness value returned worsens as it becomes greater and thus the solution becomes more distant from the global optimum of the function. Given this relation, a depreciating fitness value for an increasing problem dimensionality can be acknowledged as the result of the multimodality and therefore complexity of the functions fitness landscape, which infers that the algorithm becomes trapped in one of the many regions of steep descent that the fitness landscape facilitates; this is plausible as S being a local search algorithm, adapts to being more exploitative overtime upon its exploratory radius shrinking from no improvements in the solution being registered. Becoming increasingly exploitative theorizes S to search for more promising basins of attraction in the search space in case they are located nearby the current solution (neighbourhood), but as the exploratory radius is never reset nor enlarged, S cannot retarget its initial, explorative nature of search to escape the region; this results in local entrapment.

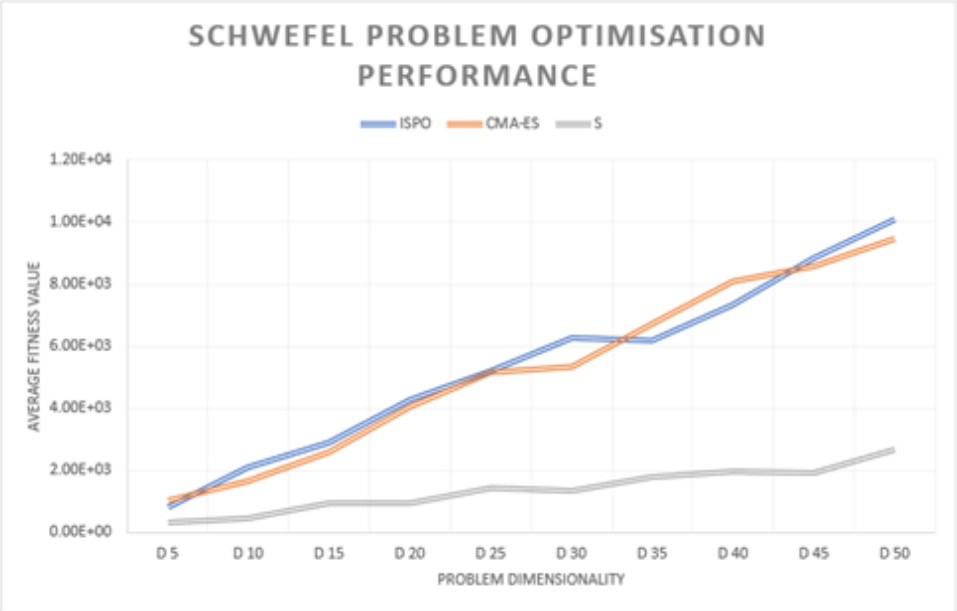

Figure 11: Graph-based visualisation of the optimisation performance of the featured optimiser algorithms, when benchmarked over the Schwefel problem, for '10' runs per dimension tested. The dimension at which the average fitness value of the solution is obtained for, is indicated along the 'x' axis of the above illustration, where the problem is evaluated for 5D, 10D, 15D, 20D, 25D, 30D, 35D, 40D, 45D and 50D.

When compared to both ISPO and CMA-ES algorithms, S is evidently more effective in navigating to an optimal basin of attraction in the functions fitness landscape, given that it always converges to a point or solution in the problem domain associated with a more optimal fitness value, whilst also demonstrating consistency within its search; this is realised by the possession of the least significant deviation in the mean fitness value that it attains, across all '10' runs of the experiments, for most dimensions of the problem. With reference to the tabularised results below, the Wilcoxon Rank-Sum test supports this observation, where in all but a '5' dimension problem space, S outperforms ISPO, and given that CMA-ES and ISPO share a similar range of optimality for all dimensions of the problem, the rank-sum test also infers that S outperforms CMA-ES; this is visually verified above by the graphical visualisation of each algorithms performance during solution optimisation, where both ISPO and CMA-ES almost project a linear relationship between the degradation in mean fitness value, as the dimensionality of the problem increases.

Problem Dimensionality	ISPO	CMA-ES	W	S	W
5D	8.262e+02 ± 4.250e+02	1.037e+03 ± 1.954e+02	=	3.079e+02 ± 1.208e+02	=
10D	2.086e+03 ± 1.754e+02	1.640e+03 ± 4.501e+02	=	4.501e+02 ± 8.863e+01	-
15D	2.900e+03 ± 4.544e+02	2.596e+03 ± 4.034e+02	=	9.633e+02 ± 1.663e+02	-
20D	4.262e+03 ± 2.293e+02	4.058e+03 ± 7.542e+02	=	9.712e+02 ± 2.742e+02	-
25D	5.208e+03 ± 4.789e+02	5.161e+03 ± 3.144e+02	=	1.429e+03 ± 6.284e+02	-
30D	6.267e+03 ± 4.342e+02	5.349e+03 ± 8.680e+02	=	1.350e+03 ± 2.959e+02	-
35D	6.199e+03 ± 9.485e+02	6.713e+03 ± 1.940e+02	=	1.788e+03 ± 1.850e+02	-
40D	7.345e+03 ± 8.102e+02	8.101e+03 ± 6.420e+02	=	1.990e+03 ± 7.897e+02	-
45D	8.857e+03 ± 4.544e+02	8.560e+03 ± 7.487e+02	=	1.942e+03 ± 3.400e+02	-
50D	1.009e+04 ± 6.912e+02	9.467e+03 ± 6.452e+02	=	2.665e+03 ± 1.498e+02	-

Table 3: Results table for the three featured optimisers benchmarked over the Schwefel problem, for 5D, 10D, 15D, 20D, 25D, 30D, 40D, 45D and 50D; this experiment was executed for '10' runs.

Although S presents a sizeable variation within the number of iterations required to converge to a near-optimal solution for the problems function, it is undeniably the most efficient of the three optimisers, for which we assume the deviation in convergence rate for ISPO and CMA-ES to be vaster, in compliance with the statistical analysis featured in the tabularised results above. As previously described, variation within the algorithms convergence rate is subject to the initial guess in the functions search space, as well as its perturbation logic, which increases the likelihood of S continually overstepping the global optimum at the functions promising basin of attraction, or alternatively becoming trapped within a sub-optimal basin of attraction. As can be understood from the graphical representation of the algorithms convergence rate below, S on average, is able to converge to a near-optimal solution for the problem function per '663.67' iterations or perturbation cycles surpassed, which is significantly faster yet less optimal in comparison to the results acquired for De Jong's problem; this too suggests the apparency of local entrapment in the search domain, as a fast rate of convergence in a complex fitness landscape is recognised for.

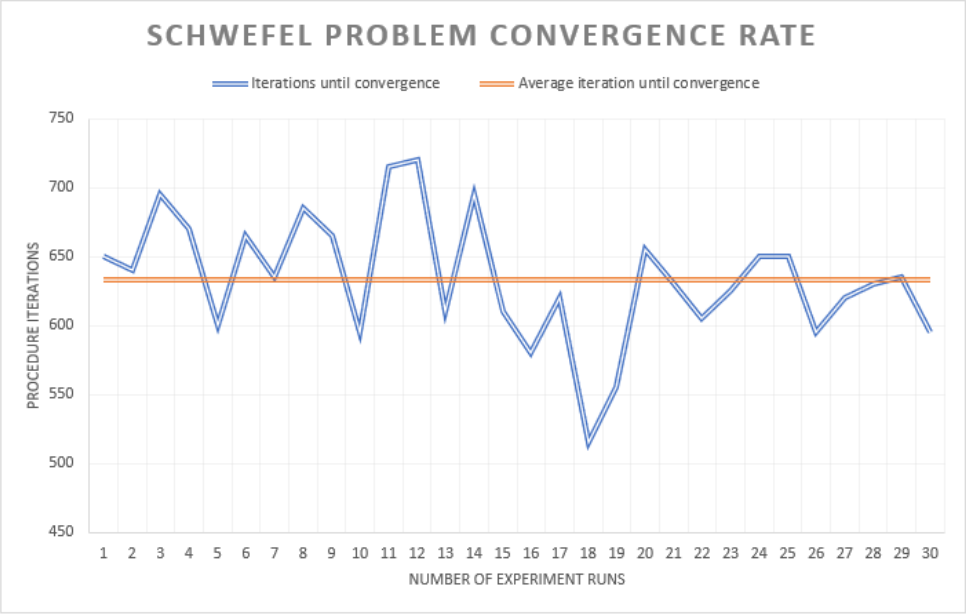

Figure 12: Graph-based visualisation of the convergence rate of the Short Distance Exploration (S) algorithm, when benchmarked over the Schwefel problem, for a total of '30' runs when evaluated in 5D.

Rastrigin

Upon the S algorithm being benchmarked over Rastrigin's testbed problem for the same array of problem dimensionality, that ranges from '5' to '50' dimensions and for '10' runs of each experiment, S with reference to the figure featured below, similarly proves to not reach the global optimum of the problem domain, for the range of dimensions bespoken; this is gathered as for every dimension, the average fitness value $f(x)$ returned is also greater than '0', where parallel with the dimensionality of the problem increasing, the fitness value returned generally worsens as it becomes greater and thus the solution becomes more distant from the global optimum of the function. Alike Schwefel's 2.26 function, a depreciating fitness value for an increasing problem dimensionality can be acknowledged as the result of the high multimodality of the functions fitness landscape, which given that its fitness landscape is significantly more multi-modal than Schwefel's function, the likelihood of the algorithm becoming trapped in one of the many regions of steep descent is

comparatively higher, as the algorithms search is also more likely to be diverted from the global optimum, via exploitation. Whereas previously mentioned, S adapts to becoming more exploitative overtime upon its exploratory radius shrinking to search for more promising basins of attraction in the search space, which is relative to no improvement being registered in the solutions fitness value; as the search radius is never restored nor enlarged, the algorithm cannot escape regions of steep descent and thereby cannot explore surrounding areas of the fitness landscape to seek the actual global optimum, thus it becomes trapped.

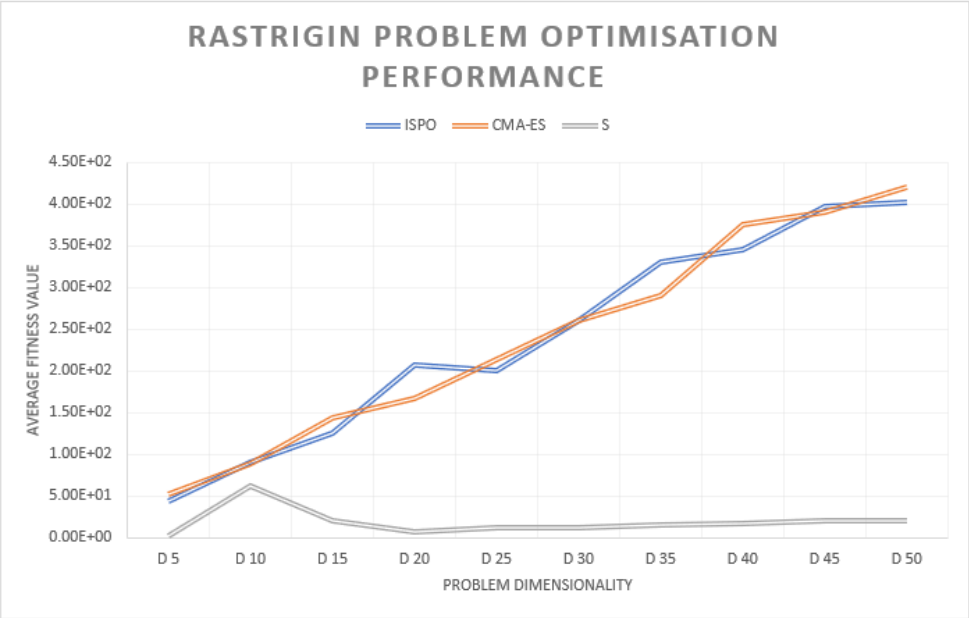

Figure 13: Graph-based visualisation of the optimisation performance of the featured optimiser algorithms, when benchmarked over the Rastrigin problem, for '10' runs per dimension tested. The dimension at which the average fitness value of the solution is obtained for, is indicated along the 'x' axis of the above illustration, where the problem is evaluated for 5D, 10D, 15D, 20D, 25D, 30D, 35D, 40D, 45D and 50D.

When comparing S to both ISPO and CMA-ES algorithms, it is once more evidential that S is more effective in navigating to an optimal basin of attraction in the functions fitness landscape, given that it always converges to a point or solution in the problem domain associated with a more optimal fitness value, whilst also demonstrating consistency within its search; this can be acknowledged by the algorithm proving to have the least significant deviation in the mean fitness value that it attains, across all '10' runs of the experiments, for most dimensions of the problem. With reference to the tabularised results below, the Wilcoxon Rank-Sum test supports this remark, where in all but a '10', dimension problem space, S outperforms ISPO, and given that CMA-ES and ISPO share a similar range of optimality for all but '20' dimensions of the problem domain, the rank-sum test also supposes that S outperforms CMA-ES; this is visually verified above by the graphical visualisation of each algorithms performance during solution optimisation, where both ISPO and CMA-ES almost project a linear relationship between the degradation in mean fitness value, as the dimensionality of the problem increases. Inevitably, each optimisers performance draws close similarities to their performances displayed when being benchmarked over Schwefel's 2.26 problem, which is presumed to be the cause of each function's fitness landscape being multi-modal and similar in geometrical structure.

Problem Dimensionality	ISPO	CMA-ES	W	S	W
5D	4.378e+01 ± 1.893e+01	5.313e+01 ± 2.640e+01	=	1.990e+00 ± 8.899e-01	-

10D	9.034e+01 ± 2.455e+01	8.835e+01 ± 9.675e+00	=	6.190e+01 ± 6.944e+01	=
15D	1.256e+02 ± 1.786e+01	1.433e+02 ± 2.219e+01	=	2.070e+01 ± 2.997e+01	-
20D	2.064e+02 ± 1.822e+01	1.680e+02 ± 2.321e+01	-	7.960e+00 ± 2.180e+00	-
25D	2.000e+02 ± 3.438e+01	2.132e+02 ± 2.815e+01	=	1.154e+01 ± 1.015e+00	-
30D	2.613e+02 ± 3.969e+01	2.600e+02 ± 5.055e+01	=	1.194e+01 ± 1.665e+00	-
35D	3.307e+02 ± 4.253e+01	2.908e+02 ± 4.501e+01	=	1.532e+01 ± 2.404e+00	-
40D	3.447e+02 ± 2.865e+01	3.749e+02 ± 5.091e+01	=	1.691e+01 ± 1.407e+00	-
45D	3.964e+02 ± 8.596e+00	3.897e+02 ± 4.464e+01	=	2.149e+01 ± 1.617e+00	-
50D	4.020e+02 ± 2.664e+01	4.196e+02 ± 6.007e+01	=	2.089e+01 ± 3.615e+00	-

Table 4: Results table for the three featured optimisers benchmarked over the Rastrigin problem, for 5D, 10D, 15D, 20D, 25D, 30D, 40D, 45D and 50D; this experiment was executed for '10' runs.

S once more presents a reasonable variation within the number of iterations required to converge to a near-optimal solution for the problems function but is yet again the most efficient of the three optimisers, for which we assume the deviation in convergence rate for ISPO and CMA-ES to be vaster, in accordance with the statistical analysis featured in the tabularised results above. As can be interpreted from the graphical representation of the algorithms convergence rate below, S on average, is able to converge to a near-optimal solution for the problem function per '697.33' iterations or perturbation cycles surpassed, which is significantly faster yet less optimal in comparison to the results acquired for De Jong's problem but is very similar in convergence rate and to the optimality of the results gathered for Schwefel's problem. Where alike Schwefel's problem, a faster (lower) rate of convergence indicates the apparency of local entrapment in the search domain, for which is more noticeable when comparing to the convergence rate of S, when it is benchmarked over De Jong's problem.

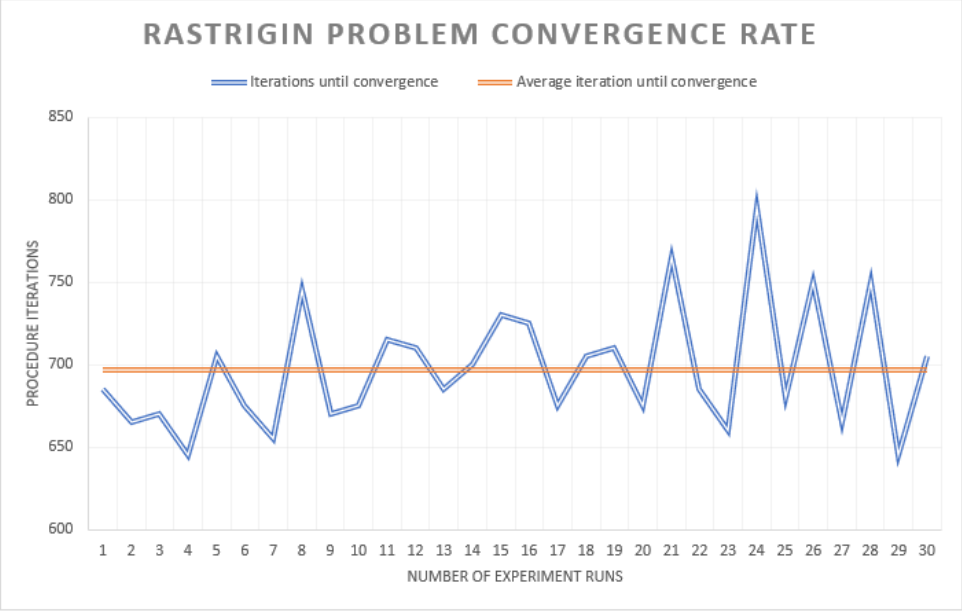

Figure 14: Graph-based visualisation of the convergence rate of the Short Distance Exploration (S) algorithm, when benchmarked over the Rastrigin problem, for a total of '30' runs when evaluated in 5D.

Upon the S algorithm being benchmarked over Michalewicz's testbed problem for the same array of problem dimensionality, that ranges from '5' to '50' dimensions and for '10' runs of each experiment, S with reference to the figure featured below, similarly proves to not reach the global optimum of the problem domain, like both Schwefel and Rastrigin problems for the range of dimensions stated. This observation is supported by the average fitness value $f(x)$ returned, which as similarly seen for the testbed problems evaluated prior, is greater than the global minimum for all dimensions, that varies in value per dimension of the problem domain and is negative by value too. Aligned with the increasing dimensionality of the problem domain, the returned fitness value of the solution worsens, as it becomes less progressive overtime and so the solution becomes more distant from the global optimum of the function. Given this relation, a depreciating fitness value for an increasing problem dimensionality can be acknowledged as the consequence of the multimodality of the function's fitness landscape, which features a series of valleys and ridges that increase in popularity with every increment in problem dimensionality; given the perturbation logic of the algorithm, the ridges of the functions fitness landscape can present uncertainty in search direction for the algorithm, whilst exhausting the computational budget of its procedure which is relied upon to establish the global optimum in the domain. Moreover, as the valleys of the function present to be regions of steep descent, the algorithm is very likely to converge to a sub-optimal basin of attraction, given its exploitative behaviours mentioned prior and due to an improvement being registered as any point in the search space with a non-null derivative; as the algorithm is incapable of escaping such region, it can become trapped within one and therefore converge to a local minimum.

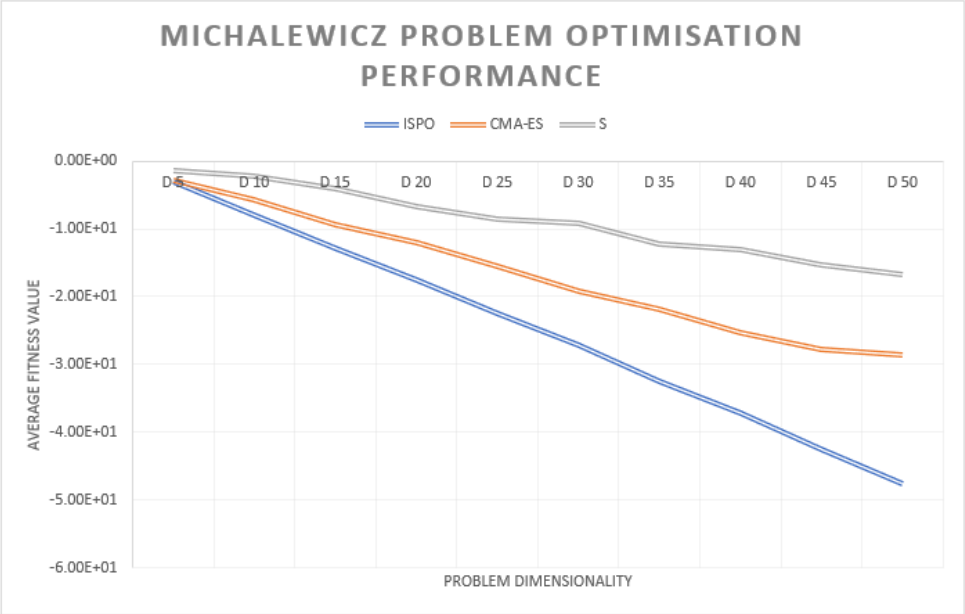

Figure 15: Graph-based visualisation of the optimisation performance of the featured optimiser algorithms, when benchmarked over the Michalewicz problem, for '10' runs per dimension tested. The dimension at which the average fitness value of the solution is obtained for, is indicated along the 'x' axis of the above illustration, where the problem is evaluated for 5D, 10D, 15D, 20D, 25D, 30D, 35D, 40D, 45D and 50D.

When comparing S to both ISPO and CMA-ES algorithms, it is noticeable that S is unusually the least effective algorithm for navigating to an optimal basin of attraction in the functions fitness landscape, given that it always converges to a point or solution in the problem domain associated with a less optimal fitness value, whilst also demonstrating a degrading capability within its search; this can be acknowledged by the algorithm proving to have a greater mean fitness value and a more significant

deviation in the mean fitness value that it attains, across all '10' runs of the experiments, for most dimensions of the problem, when compared to ISPO. With reference to the tabularised results below, the Wilcoxon Rank-Sum test supports this observation, where in all dimensions of the problem space, both ISPO and CMA-ES outperform S; this is visually confirmed above by the graphical visualisation of each algorithms performance during solution optimisation, where the search of both ISPO and CMA-ES is seen to be more progressive given that their returned fitness values reside closer to the global minimum of the problem space, as the dimensionality of the problem increases.

Problem Dimensionality	ISPO	CMA-ES	W	S	W
5D	-3.099e+00 ± 4.140e-01	-2.854e+00 ± 9.115e-01	=	-1.346e+00 ± 4.066e-01	+
10D	-7.857e+00 ± 3.736e-01	-5.708e+00 ± 9.902e-01	+	-2.275e+00 ± 7.869e-01	+
15D	-1.281e+01 ± 4.253e-01	-9.332e+00 ± 1.340e+00	+	-4.120e+00 ± 1.356e+00	+
20D	-1.762e+01 ± 3.163e-01	-1.205e+01 ± 7.956e-01	+	-6.678e+00 ± 6.113e-01	+
25D	-2.245e+01 ± 3.254e-01	-1.554e+01 ± 1.373e+00	+	-8.486e+00 ± 1.363e+00	+
30D	-2.725e+01 ± 4.256e-01	-1.927e+01 ± 8.352e-01	+	-9.236e+00 ± 2.087e+00	+
35D	-3.253e+01 ± 4.738e-01	-2.177e+01 ± 2.605e+00	+	-1.218e+01 ± 1.395e+00	+
40D	-3.727e+01 ± 2.477e-01	-2.525e+01 ± 2.282e+00	+	-1.305e+01 ± 1.404e+00	+
45D	-4.250e+01 ± 3.977e-01	-2.773e+01 ± 1.975e+00	+	-1.535e+01 ± 1.670e+00	+
50D	-4.765e+01 ± 4.571e-01	-2.865e+01 ± 4.048e+00	+	-1.675e+01 ± 1.807e+00	+

Table 5: Results table for the three featured optimisers benchmarked over the Michalewicz problem, for 5D, 10D, 15D, 20D, 25D, 30D, 40D, 45D and 50D; this experiment was executed for '10' runs.

Although S presents a similar variation within the number of iterations required to converge to a near-optimal solution for the problems function, as for the problem functions evaluated prior, it inevitably remains to be the least efficient optimiser of the three featured. For which we assume that the deviation in convergence rate for S is vaster than ISPO and CMA-ES, in compliance with the statistical analysis featured in the tabularised results above, as S struggles to search the domain and thereby exhausts its computational budget by doing so. As referred to throughout this section of the document, variation within the algorithms convergence rate is subject to the initial guess in the functions search space, as well as its perturbation logic, which increases the likelihood of S continually overstepping the global optimum at the functions promising basin of attraction, or alternatively becoming trapped within a sub-optimal basin of attraction. As one can acknowledge from the graphical representation of the algorithms convergence rate below, S on average, can converge to a near-optimal solution for the problem function per '652' iterations or perturbation cycles surpassed, which is significantly faster in comparison to the results acquired for De Jong's problem, whilst also being less optimal in comparison to the results acquired for all testbed problems implemented. Thus, the convergence rate of the algorithm suggests that S was subjected to many regions of steep descent in the functions fitness landscape, as a fast rate of convergence in a complex fitness landscape is recognised as. To see the entirety of the results for the testing regime conducted, see ***Appendix A***.

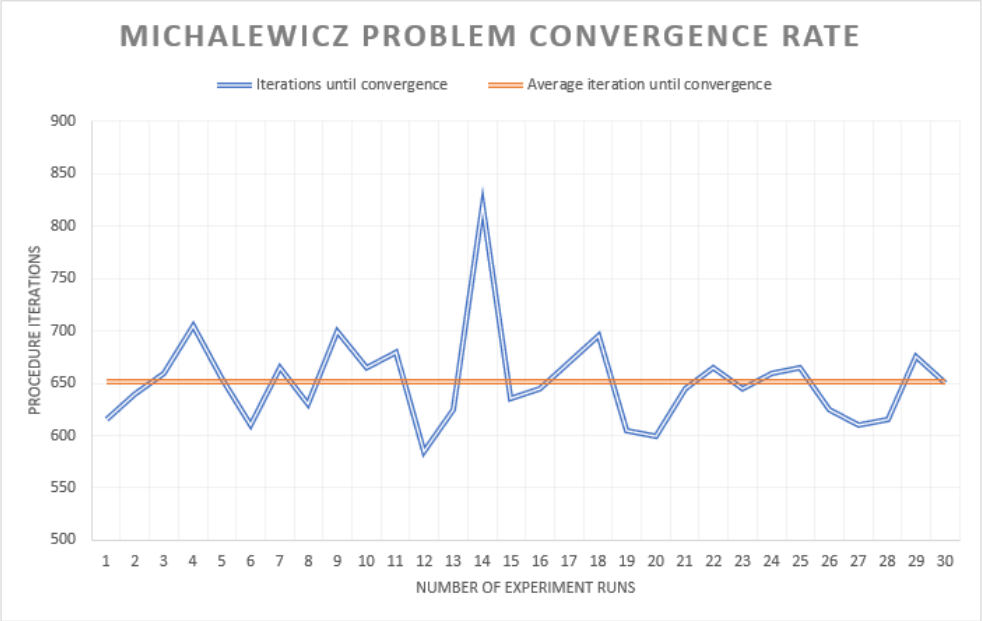

Figure 16: Graph-based visualisation of the convergence rate of the Short Distance Exploration (S) algorithm, when benchmarked over the Michalewicz problem, for a total of '30' runs when evaluated in 5D.

Conclusion

Summarily, the results acquired from the experiments led for benchmarking the featured and implemented single-solution optimisers, have demonstrated that the Short Distance Exploration algorithm is not only operational, but is reasonably capable in establishing the global optimum of a separable testbed problem, that identifies as having a low multi-modal or unimodal fitness landscape, and in establishing a near-optimal solution for multi-modal problems in low-dimensional decision spaces. Given this discovery, it has been evidenced that S performs best when benchmarked over the simplest testbed problem, De Jong (sphere), and the worst when benchmarked over the arguably most complex testbed problem, Michalewicz; for which, it is has been realised that S can outperform ISPO and CMA-ES across all testbed problems other than Michalewicz's problem, with regards to search consistency and the optimality of the solutions obtained. The algorithms difficulty in optimising a solution subjected to Michalewicz's problem, is presumed to be in cause of the functions increasing complexity, per increase in the problem domains dimensionality, which dampens the algorithms search capability in the space given, as its exploratory radius and resultant perturbation operators are nullified overtime, from adapting to exploitative-driven behaviours. In such instance, the 3Some variant of the algorithm that pledges an exploratory radii restoration after '150' iterations, would seem more sensible for navigating the function in attempt of acquiring an improved fitness value, over the simplest variant of S presented; its evaluation would have been considered, had more time been available to the project, as its implementation exists already, within the SOS executable package submitted.

Furthermore, in continued mention of the availability of time to complete this interim assessment, if more time were available, the exploratory radius of the algorithm's perturbation logic would be trialled with random alpha-cut values, applied to the length of the problem domain; this would purpose to expand the algorithms initial exploration potential, in attempt to reduce the likelihood of the algorithm converging to a sub-optimal basin of attraction, prematurely. As well, would

experiments in higher dimensional spaces be conducted, that would also be subjected to more runs for accuracy purposes and in hope of populating a broader variation within performance trends.

References

[1] Nagham, A. A-Madi. (2014) De Jong's Sphere Model Test for a Human Community Based Genetic Algorithm Model (HCBGA). *International Journal of Advanced Computer Science and Applications*. [Online] 5 (1). Available from: https://www.researchgate.net/publication/270553679_De_Jong%27s_Sphere_Model_Test_for_a_Human_Community_Based_Genetic_Algorithm_Model_HCBGA [Accessed: 24/03/21].

[2] Dizangian, B. and Hooshyari, A. (2017) Comparing the particle swarm, whale, water cycle, and cuckoo search algorithms in optimization of unconstrained problems. In: *13th International Conference on Industrial Engineering*. Berlin: ResearchGate.

[3] Bajpai, P. and Kumar, M. (2010) Genetic Algorithm - an Approach to Solve Global Optimization Problems. *Indian Journal of Computer Science and Engineering.* [Online] 1 (3). Available from: https://www.researchgate.net/publication/283361244_Genetic_Algorithm_-_an_Approach_to_Solve_Global_Optimization_Problems [Accessed: 24/03/21].

[4] Vanaret, C. and Gotteland, J-Baptiste. and Durand, N. and Alliot, J-Marc. (2014) Certified Global Minima for a Benchmark of Difficult Optimization Problems. *Hybridization of interval methods and evolutionary algorithms for solving difficult optimization problems*. [Online]. Available from: https://www.researchgate.net/publication/278769271_Certified_Global_Minima_for_a_Benchmark_of_Difficult_Optimization_Problems [Accessed: 24/03/21].

[5] Dieterich, J. M. and Hartke, B. (2012) Empirical Review of Standard Benchmark Functions Using Evolutionary Global Optimization. *Applied Mathematics*. [Online] 3 (10). Available from: https://www.researchgate.net/publication/229157888_Empirical_Review_of_Standard_Benchmark_Functions_Using_Evolutionary_Global_Optimization [Accessed: 24/03/21].

[6] Clerc, M. (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). Washington, DC, USA, July 1999*. New York: IEEE, pp. 1951-1957.

[7] Ma, J. and Li, H. (2019) Research on Rosenbrock Function Optimization Problem Based on Improved Differential Evolution Algorithm. *Journal of Computer and Communications*. [Online] 7 (11). Available from: https://www.researchgate.net/publication/337600944_Research_on_Rosenbrock_Function_Optimization_Problem_Based_on_Improved_Differential_Evolution_Algorithm [Accessed: 24/03/21].

[8] w3schools.com (2021) *Java Method Overloading*. [Online] w3schools.com. Available from: https://www.w3schools.com/java/java_methods_overloading.asp [Accessed: 24/03/21].

[9] Iacca, G. and Caraffini, F. and Neri, F. and Mininno, E. (2013) Single Particle Algorithms for Continuous Optimization. In: *2013 IEEE Congress on Evolutionary Computation, CEC 2013*. *Cancun, Mexico, June 2013*. Berlin: ResearchGate, pp. 1610-1617.

[10] Szynkiewicz, P. (2019) A Comparative Study of PSO and CMA-ES Algorithms on Black-box Optimization Benchmarks. *Journal of Telecommunications and Information Technology*. [Online] 9. Available from:

https://www.researchgate.net/publication/329971785_A_Comparative_Study_of_PSO_and_CMA-ES_Algorithms_on_Black-box_Optimization_Benchmarks [Accessed: 24/03/21].

[11] Yang, X-She. (2010) Test Problems in Optimization. [Online]. Available from: https://www.researchgate.net/publication/45932888_Test_Problems_in_Optimization [Accessed: 24/03/21].

[12] BenchmarkFcns (2021) *Sphere Function*. [Online] BenchmarkFcns. Available from: http://benchmarkfcns.xyz/benchmarkfcns/spherefcn.html [Accessed: 24/03/21].

[13] Jamil, M. and Yang, X-She. (2013) A Literature Survey of Benchmark Functions For Global Optimization Problems. *International Journal of Mathematical Modelling and Numerical Optimisation*. [Online] 4 (2). Available from: https://www.semanticscholar.org/paper/A-literature-survey-of-benchmark-functions-for-Jamil-Yang/ae3ebe6c69fdb19e12d3218a5127788fae269c10 [Accessed: 24/03/21].

[14] BenchmarkFcns (2021) *Schwefel Function*. [Online] BenchmarkFcns. Available from: http://benchmarkfcns.xyz/benchmarkfcns/schwefelfcn.html [Accessed: 24/03/21].

[15] BenchmarkFcns (2021) *Rastrigin Function*. [Online] BenchmarkFcns. Available from: http://benchmarkfcns.xyz/benchmarkfcns/rastriginfcn.html [Accessed: 24/03/21].

[16] Simon Fraser University (2013) *Virtual Library of Simulation Experiments: Test Functions and Datasets*. [Online] Simon Fraser University. Available from: https://www.sfu.ca/~ssurjano/michal.html [Accessed: 24/03/21].

[17] Li, Q. and Liu, S-Yang. and Yang, X-She. (2020) Influence of Initialization on the Performance of Metaheuristic Optimizers. Available from: https://www.researchgate.net/publication/339814374_Influence_of_Initialization_on_the_Performance_of_Metaheuristic_Optimizers [Accessed: 24/03/21].

[18] Zenodo (2020) *The Stochastic Optimisation Software (SOS) platform*. [Online] Zenodo. Available from: https://zenodo.org/record/3237024#.YFc4LtynyUn [Accessed: 24/03/21].

[19] Nero, R. D. (2020) Does Java pass by reference or pass by value? [Online] InfoWorld. Available from: https://www.infoworld.com/article/3512039/does-java-pass-by-reference-or-pass-by-value.html [Accessed: 24/03/21].

[20] Lee, J. S. C. and Kulperger, R. and Yu, Hao. (2013) parspatstat: An R Package for Large-Scale Spatial Analysis with Parallel Computing. [Online]. Available from: https://www.semanticscholar.org/paper/parspatstat%3A-An-R-Package-for-Large-Scale-Spatial-Lee-Kulperger/3db2124dbff8e6a9d0d9d054adbbcf4e73f8d136#citing-papers [Accessed: 24/03/21].

[21] Caraffini, F. (2014) *Novel Memetic Computing Structures for Continuous Optimisation*. [Online] De Montfort University. Available from: https://dora.dmu.ac.uk/handle/2086/10629 [Accessed: 24/03/21].

[22] Moser, I. (2009) Hooke-Jeeves Revisited. In: *2009 IEEE Congress on Evolutionary Computation, CEC 2009*. *Trondheim, Norway, May 2009*. IEEE: New York, pp. 2670-2676.

[23] Caraffini, F. and Iacca, G. and Neri, F. and Mininno, E. (2012) Meta-Lamarckian Learning in Three Stage Optimal Memetic Exploration. In: *2012 12th UK Workshop on Computational Intelligence (UKCI)*. Edinburgh, UK, October 2012. IEEE: New York, pp. 1-8.

[24] Neri, F. and Weber, M. and Caraffini, F. and Poikolainen, I. (2012) Three variants of three Stage Optimal Memetic Exploration for handling non-separable fitness landscapes. In: *2012 12th UK Workshop on Computational Intelligence (UKCI)*. Edinburgh, UK, October 2012. IEEE: New York, pp. 1-8.

[25] Iacca, G. and Neri, F. and Mininno, E. and Ong Y-Soon. and Lim, M-Hiot (2012) Ockham's Razor in memetic computing: Three stage optimal memetic exploration. *Information Sciences*. [Online] 188. Available from: https://www.sciencedirect.com/science/article/pii/S0020025511006104 [Accessed: 24/03/21].

[26] Tseng, L-Yu. and Chen, C. (2008) Multiple trajectory search for Large Scale Global Optimization. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. Hong Kong, China, June 2008. Available from: https://ieeexplore.ieee.org/document/4631210 [Accessed: 24/03/21].

[27] Poikolainen, I. and Iacca, G. and Neri, F. and Mininno, E. and Weber, M. (2012) Shrinking Three Stage Optimal Memetic Exploration. In: *5th International Conference on Bioinspired Optimization Methods and their Applications*. Berlin: ResearchGate, pp. 1-12.

Appendices

Appendix A:

		Dimensionality 5D			
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	1.489e-35 ± 9.555e-36	6.747e-311 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	8.262e+02 ± 4.250e+02	1.037e+03 ± 1.954e+02	=	3.079e+02 ± 1.208e+02	=
f_3	4.378e+01 ± 1.893e+01	5.313e+01 ± 2.640e+01	=	1.990e+00 ± 8.899e-01	-
f_4	-3.099e+00 ± 4.140e-01	-2.854e+00 ± 9.115e-01	=	-1.346e+00 ± 4.066e-01	+

Table 6: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 5D; this experiment was executed for '10' runs.

		Dimensionality 10D			
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	5.378e-35 ± 2.360e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	2.086e+03 ± 1.754e+02	1.640e+03 ± 4.501e+02	=	4.501e+02 ± 8.863e+01	-
f_3	9.034e+01 ± 2.455e+01	8.835e+01 ± 9.675e+00	=	6.190e+01 ± 6.944e+01	=
f_4	-7.857e+00 ± 3.736e-01	-5.708e+00 ± 9.902e-01	+	-2.275e+00 ± 7.869e-01	+

Table 7: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 10D; this experiment was executed for '10' runs.

		Dimensionality 15D			
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	6.516e-35 ± 2.250e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	2.900e+03 ± 4.544e+02	2.596e+03 ± 4.034e+02	=	9.633e+02 ± 1.663e+02	-
f_3	1.256e+02 ± 1.786e+01	1.433e+02 ± 2.219e+01	=	2.070e+01 ± 2.997e+01	-

f_4	-1.281e+01 ± 4.253e-01	-9.332e+00 ± 1.340e+00	+	-4.120e+00 ± 1.356e+00	+

Table 8: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 15D; this experiment was executed for '10' runs.

Dimensionality 20D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	1.120e-34 ± 4.523e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	4.262e+03 ± 2.293e+02	4.058e+03 ± 7.542e+02	=	9.712e+02 ± 2.742e+02	-
f_3	2.064e+02 ± 1.822e+01	1.680e+02 ± 2.321e+01	-	7.960e+00 ± 2.180e+00	-
f_4	-1.762e+01 ± 3.163e-01	-1.205e+01 ± 7.956e-01	+	-6.678e+00 ± 6.113e-01	+

Table 9: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 20D; this experiment was executed for '10' runs.

Dimensionality 25D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	1.145e-34 ± 3.002e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	5.208e+03 ± 4.789e+02	5.161e+03 ± 3.144e+02	=	1.429e+03 ± 6.284e+02	-
f_3	2.000e+02 ± 3.438e+01	2.132e+02 ± 2.815e+01	=	1.154e+01 ± 1.015e+00	-
f_4	-2.245e+01 ± 3.254e-01	-1.554e+01 ± 1.373e+00	+	-8.486e+00 ± 1.363e+00	+

Table 10: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 25D; this experiment was executed for '10' runs.

Dimensionality 30D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	1.727e-34 ± 6.947e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	6.267e+03 ± 4.342e+02	5.349e+03 ± 8.680e+02	=	1.350e+03 ± 2.959e+02	-
f_3	2.613e+02 ± 3.969e+01	2.600e+02 ± 5.055e+01	=	1.194e+01 ± 1.665e+00	-
f_4	-2.725e+01 ± 4.256e-01	-1.927e+01 ± 8.352e-01	+	-9.236e+00 ± 2.087e+00	+

Table 11: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 30D; this experiment was executed for '10' runs.

Dimensionality 35D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	1.892e-34 ± 3.331e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	6.199e+03 ± 9.485e+02	6.713e+03 ± 1.940e+02	=	1.788e+03 ± 1.850e+02	-
f_3	3.307e+02 ± 4.253e+01	2.908e+02 ± 4.501e+01	=	1.532e+01 ± 2.404e+00	-
f_4	-3.253e+01 ± 4.738e-01	-2.177e+01 ± 2.605e+00	+	-1.218e+01 ± 1.395e+00	+

Table 12: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 35D; this experiment was executed for '10' runs.

Dimensionality 40D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	2.442e-34 ± 8.327e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	7.345e+03 ± 8.102e+02	8.101e+03 ± 6.420e+02	=	1.990e+03 ± 7.897e+02	-
f_3	3.447e+02 ± 2.865e+01	3.749e+02 ± 5.091e+01	=	1.691e+01 ± 1.407e+00	-
f_4	-3.727e+01 ± 2.477e-01	-2.525e+01 ± 2.282e+00	+	-1.305e+01 ± 1.404e+00	+

Table 13: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 40D; this experiment was executed for '10' runs.

Dimensionality 45D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	2.134e-34 ± 3.403e-35	5.243e+00 ± 1.049e+01	=	0.000e+00 ± 0.000e+00	-
f_2	8.857e+03 ± 4.544e+02	8.560e+03 ± 7.487e+02	=	1.942e+03 ± 3.400e+02	-
f_3	3.964e+02 ± 8.596e+00	3.897e+02 ± 4.464e+01	=	2.149e+01 ± 1.617e+00	-
f_4	-4.250e+01 ± 3.977e-01	-2.773e+01 ± 1.975e+00	+	-1.535e+01 ± 1.670e+00	+

Table 14: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 45D; this experiment was executed for '10' runs.

Dimensionality 50D					
Problem Function	**ISPO**	**CMA-ES**	**W**	**S**	**W**
f_1	2.369e-34 ± 4.518e-35	0.000e+00 ± 0.000e+00	-	0.000e+00 ± 0.000e+00	-
f_2	1.009e+04 ± 6.912e+02	9.467e+03 ± 6.452e+02	=	2.665e+03 ± 1.498e+02	-
f_3	4.020e+02 ± 2.664e+01	4.196e+02 ± 6.007e+01	=	2.089e+01 ± 3.615e+00	-
f_4	-4.765e+01 ± 4.571e-01	-2.865e+01 ± 4.048e+00	+	-1.675e+01 ± 1.807e+00	+

Table 15: Results table for an experiment featuring the three optimisers and the four implemented testbed problems, for the problem dimensionality of 50D; this experiment was executed for '10' runs.