Empirical analysis of grey wolf optimiser

JW du Toit
Computer Science Department
Stellenbosch University
Stellenbosch, South Africa
22808892@sun.ac.za
22808892

***Abstract*—The grey wolf optimiser (GWO) is meta-heuristic inspired by the behaviour of grey wolf packs. The GWO algorithm simulates the leadership hierarchy and hunting techniques of grey wolf packs to solve optimisation problems. This report performs a comparative study of the GWO algorithm and two other meta-heuristics, particle swarm optimisation (PSO) and differential evolution (DE). These algorithms are compared over six benchmark single-objective, boundary constrained optimisation problems with regard to three standard performance measures. It is observed that the PSO algorithm is the least computationally expensive algorithm, followed by GWO, with DE being significantly less efficient compared to the other two algorithms. Furthermore, it is observed that the GWO algorithm achieves competitive results for several optimisation problems, but performs significantly worse than the other two algorithms for other problems regarding the quality of the best solution and stability. Thus, the PSO algorithm is considered to be the best optimisation technique among the three algorithms analysed in this report.**

I. Introduction

Meta-heuristic optimisation techniques have become a popular method for solving optimisation problems in many different fields. Meta-heuristics are algorithms that provide a sufficiently good solution, but are not guaranteed to provide optimal solutions, to an optimisation problem. Meta-heuristics such as particle swarm optimisation (PSO) [6], ant colony optimisation (ACO) [2], genetic algorithms (GA) [4] and differential evolution (DE) [14] have been applied to efficiently solve a range of optimisation problems. There are many other less-known nature inspired meta-heuristics such as the grey wolf optimiser (GWO) [9] which is the focus of this report.

The GWO is a meta-heuristic based on the behaviour of grey wolf packs. The GWO algorithm simulates the leadership hierarchy and hunting techniques of grey wolf packs to solve optimisation problems. The hierarchy of the wolf pack is simulated through the use of four types of wolves where a wolf represents a solution to the optimisation problem. The algorithm simulates the three main steps of hunting, searching for, encircling and attacking prey, to find a solution to an optimisation problem.

The objective of this report is to compare the performance of the GWO algorithm with two other meta-heuristics and see whether or not it can produce competitive results. This report performs a comparative study of the GWO algorithm with two other popular meta-heuristics, PSO and DE. These algorithms are compared over six benchmark single-objective, boundary constrained optimisation problems regarding computational effort, quality of the solutions obtained and stability.

It is observed that the PSO algorithm is the least computationally expensive algorithm, followed by GWO, with DE being significantly more expensive compared to the other two algorithms. Furthermore, the GWO algorithm achieves competitive results for several problems, but performs significantly worse for other problems regarding the quality of the best solution and stability. Thus, the PSO algorithm is considered to be the best optimisation technique among the three algorithms analysed in this report.

The rest of the report proceeds as follows: Section II provides a discussion on meta-heuristics and the GWO algorithm. Section III provides descriptions of the PSO and DE algorithms. Section IV introduces the procedure followed to obtain empirical results. Section V discusses the results obtained from the experiments and Section VI draws a conclusion for the report.

II. Background

This section provides a discussion on meta-heuristics and the many different meta-heuristics that have been developed. Furthermore, the GWO algorithm is described.

A. Meta-heuristics

Meta-heuristics are algorithms that provide a sufficiently good solution, but are not guaranteed to provide optimal solutions, to an optimisation problem. Meta-heuristics attempt to efficiently find good quality solutions to an optimisation problem through exploring and exploiting the search space without getting stuck in specific zones such as local optima. Meta-heuristics can be classified into two categories, trajectory-based and population-based [10].

Trajectory-based meta-heuristics use only one candidate solution which is replaced by another solution found in its neighbourhood at each iteration. Trajectory-based meta-heuristics are usually faster than population-based meta-heuristics because they only work with one solution at a time. Simulated annealing [5] is a trajectory-based meta-heurisitc based on the annealing process of metals which allows the replacement of a solution with one that does not improve the objective function in order to escape local optima. Tabu search [8] is another trajectory-based meta-heuristic that uses a memory structure

to save information about previous solutions and avoids returning to recently visited solutions in order to facilitate more diversity. Iterated local search [13] is a trajectory-based meta-heuristic that uses a hill climbing local search to find local optima and a perturbation strategy to prevent the search from getting stuck in local optima.

In contrast with trajectory-based meta-heuristics, population based meta-heuristics use a set of candidate solutions at each iteration of the algorithm. At each iteration several solutions in the population are replaced by newly generated solutions. Population based meta-heuristics generally provide increased diversity through the use of many candidate solutions. Evolutionary computation is a class of population-based meta-heuristics inspired by the evolutionary process found in nature. Popular evolutionary computation methods include GA [4], genetic programming [7], evolutionary programming [3] and DE [14]. Swarm intelligence is another class of population-based meta-heuristics based on the collective behaviour of self organised systems [10]. Swarm intelligence methods use a set of agents to perform an exploration of the search space while they interact with agents in their neighbourhood. Popular swarm intelligence algorithms include PSO [6], ACO [2] and artificial immune systems [1].

B. Grey wolf optimiser

The GWO [9] is meta-heuristic inspired by the behaviour of grey wolf packs. The GWO algorithm simulates the leadership hierarchy and hunting techniques of grey wolf packs to solve optimisation problems. The three main steps of grey wolf hunting are searching for, encircling and attacking prey.

The hierarchy of the wolf pack is simulated through the use of four types of wolves where each wolf represents a solution to the optimisation problem. The fittest solution is classified as the alpha (α) wolf. The second and third fittest solutions are classified as the beta (β) and delta (δ) wolves respectively. The rest of the candidate solutions are classified as omega (ω) wolves. The optimisation or hunting process is guided through the α, β and δ wolves as the ω wolves are drawn towards these three wolves.

During the hunting procedure, grey wolves encircle their prey. The encircling behaviour is modelled by

$$\mathbf{D} = \mathbf{C} \cdot \mathbf{X}_p(t) - \mathbf{X}(t) \tag{1}$$

$$\mathbf{X}(t+1) = \mathbf{X}_p(t) - \mathbf{A} \cdot \mathbf{D} \tag{2}$$

where $\mathbf{D}$ is a distance vector used to update the position of a wolf, $\mathbf{A}$ and $\mathbf{C}$ are coefficient vectors, $\mathbf{X}_p(t)$ is the position vector of the prey at iteration t, and $\mathbf{X}(t+1)$ is the position vector of a wolf at iteration $t+1$. The coefficient vectors $\mathbf{A}$ and $\mathbf{C}$ are calculated according to

$$\mathbf{A} = 2\mathbf{a} \cdot \mathbf{r}_1 - \mathbf{a} \tag{3}$$

$$\mathbf{C} = 2\mathbf{r}_2 \tag{4}$$

where the components of $\mathbf{a}$ are linearly decreased from 2 to 0 over the course of the iterations of the algorithm and $\mathbf{r}_1$, $\mathbf{r}_2$ are vectors whose components are sampled uniformly from (0, 1).

The first phase of hunting is to search for prey. The search is guided by the three wolves with the best solutions found so far, the α, β and δ wolves. The rest of the ω wolves update their positions according to the positions of these three wolves. The position of an ω wolf is updated according to

$$\begin{aligned}\mathbf{D}_\alpha &= \mid \mathbf{C}_1 \cdot \mathbf{X}_\alpha(t) - \mathbf{X}(t) \mid, \mathbf{D}_\beta = \mid \mathbf{C}_2 \cdot \mathbf{X}_\beta(t) - \mathbf{X}(t) \mid, \\ \mathbf{D}_\delta &= \mid \mathbf{C}_3 \cdot \mathbf{X}_\delta(t) - \mathbf{X}(t) \mid\end{aligned} \tag{5}$$

$$\mathbf{X}_1 = \mathbf{X}_\alpha - \mathbf{A}_1 \cdot (\mathbf{D}_\alpha), \mathbf{X}_2 = \mathbf{X}_\beta - \mathbf{A}_2 \cdot (\mathbf{D}_\beta), \mathbf{X}_3 = \mathbf{X}_\delta - \mathbf{A}_3 \cdot (\mathbf{D}_\delta) \tag{6}$$

$$\mathbf{X}(t+1) = \frac{\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3}{3} \tag{7}$$

where $\mathbf{X}_\alpha(t), \mathbf{X}_\beta(t), \mathbf{X}_\delta(t)$ are the positions of the α, β and δ wolves respectively at iteration t, $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ are coefficient vectors calculated according to Eq. 3 and $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{C}_3$ are coefficient vectors calculated according to Eq. 4. These equations imply that the α, β and δ wolves estimate the position of the prey such that the ω wolves update their positions randomly around the prey.

The final stage of the hunting phase is the attacking of prey when the prey stops moving. This behaviour is modelled by decreasing the value of $\mathbf{a}$. $\mathbf{A}$ is a random vector with components in the interval $[-2a, 2a]$ where a is the value of each component of $\mathbf{a}$. When $|\mathbf{A}| < 1$ the wolves are forced to attack towards the prey. If $|\mathbf{A}| > 1$ the wolves diverge from the prey and search for better prey. Thus, the components of $\mathbf{a}$ are linearly decreased from 2 to 0 over the course of the iterations of the algorithm to emphasise exploration in the initial stages of the algorithm and exploitation in the later stages. The vector $\mathbf{C}$ is also used to improve exploration abilities of the GWO and is not linearly decreased over the iterations in order to facilitate exploration throughout the execution of the algorithm.

The pseudo code for the GWO algorithm is provided by Algorithm 1 where n is the number of wolves in the pack. The pack is initialised by selecting random values for the elements of each wolf from the bounds for the given problem to be solved. Thus, for each wolf, $\mathbf{X}_i(0)$,$\mathrm{X}_{ij}(0) \sim U(x_{min,j}, x_{max,j})$ where $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the lower and upper boundaries of the search space respectively.

III. Implementation

This section describes the two algorithms used for the comparative study, PSO and DE.

A. Particle swarm optimisation

The PSO [6] algorithm is a population-based, stochastic optimisation algorithm inspired by the social behaviour of birds within a flock. PSO algorithms use a swarm of particles, where

Algorithm 1 Grey wolf optimisation algorithm

Initialise the grey wolf population $\mathbf{X}_i$ $(i = 1, 2, ..., n)$;
Initialise $\mathbf{a}, \mathbf{A}, \mathbf{C}$;
Sort the population in decreasing order of fitness;
Let $\mathbf{X}_\alpha$ = the best wolf;
Let $\mathbf{X}_\beta$ = the second best wolf;
Let $\mathbf{X}_\delta$ = the third best wolf;
Let $t = 0$;
while $t <$ maximum number of iterations **do**
  for each wolf i **do**
    Update the position according to Eq. 7;
  end for
  Update $\mathbf{a}, \mathbf{A}, \mathbf{C}$;
  Sort the population in decreasing order of fitness;
  Update $\mathbf{X}_\alpha$, $\mathbf{X}_\beta$ and $\mathbf{X}_\delta$;
  $t = t + 1$;
end while
Return $\mathbf{X}_\alpha$;

Algorithm 2 Global best particle swarm optimisation algorithm

Create and initialise an n_x dimensional swarm, S;
repeat
  for each particle $i = 1, \ldots, S.n_s$ **do**
    if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ **then**
      $S.\mathbf{y}_i = S.\mathbf{x}_i$;
    end if
    if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ **then**
      $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
    end if
  end for
  for each particle $i = 1, \ldots, S.n_s$ **do**
    Update the velocity according to Eq. 9;
    Update the position according to Eq. 8;
  end for
until stopping condition is true

each particle represents a potential solution to the optimisation problem. Each particle moves through the search space and changes its position based on local and global information. This report uses the inertia weight PSO algorithm developed by Shi and Eberhart [15]. According to this algorithm particle positions are determined by

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{8}$$

where $\mathbf{x}_i^{t+1}$ denotes the position of the i-th particle at iteration $t + 1$ and $\mathbf{v}_i^{t+1}$ denotes the velocity of the i-th particle at iteration $t + 1$. Particle velocities are determined by

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + c_1\mathbf{r}_{i_1}^t(\mathbf{y}_i^t - \mathbf{x}_i^t) + c_2\mathbf{r}_{i_2}^t(\hat{\mathbf{y}}_i^t - \mathbf{x}_i^t) \tag{9}$$

where $\mathbf{y}_i^t$ denotes the personal best position of the i-th particle at iteration t, $\hat{\mathbf{y}}_i^t$ denotes the neighbourhood best position of the i-th particle at iteration t, w denotes the inertia weight which determines the effect of the momentum component of the movement of a particle, c_1 denotes the cognitive acceleration coefficient which determines the importance of local information, c_2 denotes the social acceleration coefficient which determines the importance of global information and $\mathbf{r}_{i_1}^t$ and $\mathbf{r}_{i_2}^t$ are vectors whose components are uniformly sampled from $(0, 1)$. For the purpose of this report the global best PSO is used and thus the neighbourhood of a particle is the entire swarm.

The pseudo code for the global best inertia weight PSO algorithm is given by Algorithm 2. The swarm is initialised by selecting random values for the elements of each particle from the bounds for the given problem to be solved. Thus, for each particle, $\mathbf{x}_i^0$,$\mathbf{x}_{ij}^0 \sim U(x_{min,j}, x_{max,j})$ where $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the lower and upper boundaries of the search space respectively.

For a PSO with inertia weight algorithm there exists certain converge conditions, such that when satisfied it is guaranteed to reach an equilibrium state, *i.e.*, each particle will have a velocity of zero and stop moving. The PSO control parameter values, *i.e.*, w, c_1 and c_2, can be chosen such that these convergence conditions are satisfied. Convergence conditions used for this report were derived by Poli [12] and is given by

$$c_1 + c_2 < \frac{24(1 - w^2)}{7 - 5w}, |w| < 1 \tag{10}$$

B. Differential evolution

Differential evolution [14] is a population-based, stochastic search strategy which differs from other evolutionary algorithms (EA) as it makes use of distance and direction information from the current population to guide the search process. Furthermore, as opposed to other EAs, DE applies mutation first to generate a trail vector which is used by the crossover operator to produce offspring. DE algorithms also do not sample mutation step sizes from a probability distribution function, but rather make use of difference vectors to determine mutation step sizes.

The mutation in DE algorithms produces a trial vector, $\mathbf{u}_i(t)$, by mutating a target vector with a weighted difference vector. Each parent, $\mathbf{x}_i(t)$, generates the trail vector, $\mathbf{u}_i(t)$, according to

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{11}$$

where $\mathbf{x}_{i_1}(t)$ is a target vector selected from the population such that $i \neq i_1$, $\mathbf{x}_{i_2}(t)$ and $\mathbf{x}_{i_3}(t)$ are randomly selected individuals from the population such that $i \neq i_1 \neq i_2 \neq i_3$ where i_2, i_3 are values sampled uniformly from (1, n_s) and $\beta \in (0, \infty)$ is the scale factor. This report selects the target vector randomly from the current population and thus i_1 is sampled uniformly from (1, n_s).

The DE crossover operator applies a discrete recombination of the parent vector, $\mathbf{x}_i(t)$, and the trail vector, $\mathbf{u}_i(t)$ to produce offspring, $\mathbf{x}_i'(t)$ and is calculated according to

$$x_{ij}'(t) \begin{cases} u_{ij}(t) & \text{if } j \in J \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{12}$$

where $\mathbf{x}_{ij}(t)$ is the j-th element of the vector $\mathbf{x}_i(t)$ and J is the set crossover points. This report uses the binomial crossover which selects crossover points randomly from the set of possible crossover points, $\{1, 2, \ldots, n_x\}$, where n_x is the dimension of the problem. The pseudo code for the binomial crossover selection of crossover points is provided in Algorithm 3 where p_r is the probability that a considered crossover point will be included in J.

Algorithm 3 Binomial crossover for selecting crossover points

```
j* ~ U(1, n_x)
J ← J ∪ {j*};
for each j ∈ {1, ...n_x} do
  if U(0,1) < p_r and j ≠ j* then
    J ← J ∪ j;
  end if
end for
```

The pseudo code for the DE algorithm is given by Algorithm 4. The population is initialised by selecting random values for the elements of each individual from the bounds for the given problem to be solved. Thus, for each individual, $\mathbf{x}_i(0)$,$\mathrm{x}_{ij}(0) \sim U(x_{min,j}, x_{max,j})$ where $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the lower and upper boundaries of the search space respectively.

Algorithm 4 Differential evolution algorithm

```
Let t = 0;
Initialise the control parameters β and p_r;
Create and initialise the population, C(0), of n_s individuals;
while stopping condition(s) not true do
  for each individual, x_i(t) ∈ C(t) do
    Evaluate the fitness, f(x_i(t));
    Create the trial vector, u_i(t) by applying the mutation
    operator in Eq. 11;
    Create an offspring, x'_i(t), by applying the crossover
    operator in Eq. 12;
    if f(x'_i(t)) is better than f(x_i(t)) then
      Add x'_i(t) to C(t+1);
    else
      Add x_i(t) to C(t+1);
    end if
  end for
  t = t + 1
end while
Return the individual with the best solution;
```

IV. Empirical Process

This section describes the procedure that was followed to obtain the empirical results for this report.

The GWO, PSO and DE algorithms were evaluated over six benchmark single-objective, boundary constrained, static optimisation problems. The optimisation problems included the Rosenbrock, Rastigrin, Schwefel 1.2, Sphere, Qing and Step 3 functions. The properties of these functions are provided in Table I.

TABLE I
Properties of optimisation problems

Name	Bounds	Separability	Modality	f_{min}
Rosenbrock	$[-30, 30]$	Non-separable	Unimodal	0
Rastigrin	$[-5.12, 5.12]$	Separable	Multimodal	0
Schwefel 1.2	$[-100, 100]$	Non-separable	Unimodal	0
Sphere	$[0, 10]$	Separable	Multimodal	0
Qing	$[-500, 500]$	Separable	Multimodal	0
Step 3	$[-100, 100]$	Separable	Unimodal	0

TABLE II
Control parameter combinations for PSO

	w	c_1	c_2
Combination 1	0.7	1.4	1.4
Combination 2	0.9	0.7	0.7
Combination 3	0.8	1.2	1.2
Combination 4	0.6	1.8	1.8

For each benchmark optimisation problem the three algorithms were compared with regard to three performance measures, the time taken for the execution of the algorithm, the quality of the best solution and the stability of the algorithm. The quality of the best solution was taken over time to obtain insight into how the quality of the best solution changes during the execution of the algorithm.

The PSO and DE algorithms were executed with different combinations of parameter values due to the fact that the control parameter values have a significant impact on the performance of these algorithms. These combinations are used to obtain a measure of how well these two algorithms generally perform. The control parameter values for the PSO algorithm were chosen such that they satisfy the convergence conditions in Eq. 10 which guarantees that the particles will reach an equilibrium state. The control parameter values for the DE algorithm were chosen to fall in the range $[0.5, 0.9]$ and $[0.1, 0.9]$ for β and p_r respectively. The combinations of control parameter values used for the PSO and DE algorithms are provided in Table II and Table III respectively. Furthermore, a population and dimension size of 30 was used for all of the experiments executed.

In order to obtain the performance profiles for a specific algorithm with a certain combination of control parameter values, the following procedure was performed:

1. A timer is started.
2. The algorithm is executed for 2000 iterations where at

TABLE III
Control parameter combinations for DE

	β	p_r
Combination 1	0.5	0.7
Combination 2	0.9	0.1
Combination 3	0.9	0.9
Combination 4	0.5	0.3

each iteration the quality of the best solution is collected.
3. Step 2 is executed 20 times and the average and standard deviation of the values over the 20 independent runs is obtained at each iteration.
4. The average at each iteration is plotted on a graph to indicate how the algorithm performed over time.
5. The standard deviation is added to and subtracted from the average at each iteration and these points are plotted to indicate the standard deviation over time.
6. The timer is stopped and the time taken to execute the steps above is noted.

For each of the six benchmark optimisation problems the procedure mentioned above was executed with the three algorithms.

V. Research Results

This section presents and discusses the results obtained through the empirical procedure described in Section IV. First, the performance of the three algorithms was compared with regard to computational effort. Then, the average quality of the best position over the 20 independent runs for each algorithm was compared. Finally, the stability of the algorithms was compared through the standard deviation obtained for the quality of the best position. For the PSO and DE algorithms the combination of control parameter values which yielded the best results regarding a given performance metric were used for the discussion in this section.

A. Computational effort

The computational effort of the three algorithms was compared through the time taken to execute the 20 independent runs of 2000 iterations. This provides a measure of the computational effort needed to execute each of the algorithms where less time implies a more efficient algorithm in terms of computational cost. Table IV shows the time taken to execute the 20 independent runs by the three algorithms for each of the benchmark optimisation problems.

From Table IV it was observed that the differential evolution algorithm is significantly more computationally expensive than the PSO and GWO algorithms. This is seen for the Rosenbrock function where the DE algorithm has an execution time of 446.07 seconds compared to 53.22 and 34.64 seconds for GWO and PSO respectively. This trend was observed throughout all of the benchmark optimisation problems used in this report. The high computational cost of the DE algorithm can be attributed to the expensive procedures of the mutation, crossover and selection operators.

Table IV indicated that the PSO algorithm is less computationally expensive than the GWO algorithm. This is seen for the Rastrigin function with execution times of 48.74 and 30.61 for the GWO and PSO algorithms respectively. This trend was observed for all of the benchmark optimisation problems. The low computational cost of the PSO algorithm can be attributed to the fewer number of calculations needed compared to the other two algorithms.

TABLE IV
Execution time

Problem	GWO	PSO	DE
Rosenbrock	53.22	34.64	446.07
Rastigrin	48.74	30.61	454.93
Schwefel 1.2	73.71	54.41	560.05
Sphere	42.04	23.88	410.88
Qing	49.83	30.75	478.78
Step 3	48.75	29.71	451.58

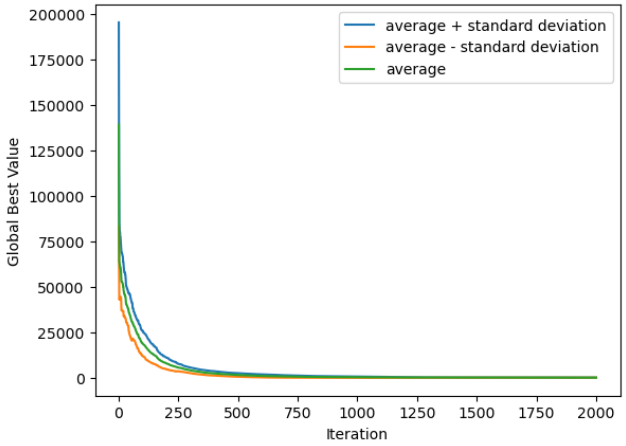

Fig. 1. Global best value over time for Schwefel 1.2 function with PSO

B. Quality of best solution

The quality of the best positions over time for the three algorithms were compared to see how efficiently each algorithm can balance exploration and exploitation throughout its execution to obtain good results.

Fig. 1 shows the quality of the best position at each iteration over the execution of the PSO algorithm for the Schwefel 1.2 function. This figure is representative of the trends shown for the execution of all three algorithms for each of the benchmark optimisation problems used in this report. From Fig. 1 it was observed that the value of the best position decreases towards the global minimum of zero and thus the quality of the best position improves.

Table V shows the average value of the best position found after 2000 iterations for each of the algorithms with regard to all of the benchmark optimisation problems. From Table V it was observed that the DE algorithm outperformed the other two algorithms for the Rosenbrock function. Table V indicated that the PSO algorithm achieved the best results for the Schewel 1.2 and Qing functions. Furthermore, the GWO outperformed the other two algorithms for the Rastigrin and Sphere functions. Finally, the GWO and DE algorithms outperformed the PSO for the Step 3 function.

From these results it is clear that the performance of these algorithms is problem dependent. This is even more evident for the GWO algorithm, as the algorithm achieved very competitive results for several optimisation problems but performed significantly worse than the other two algorithms for problems

TABLE V
AVERAGE BEST POSITION VALUE

Problem	GWO	PSO	DE
Rosenbrock	28.75	40.94	27.93
Rastigrin	0	92.43	1.23E-8
Schwefel 1.2	7757.68	1.14	581.51
Sphere	0	1.05E-8	3.74E-33
Qing	4579.49	0.00003	0.05
Step 3	0	5.45	0

TABLE VI
STANDARD DEVIATION OF BEST POSITION VALUE

Problem	GWO	PSO	DE
Rosenbrock	0.06	29.01	12.25
Rastigrin	0	21.48	8.25E-9
Schwefel 1.2	3893.44	2.68	224.21
Sphere	0	4.55E-8	4.35E-33
Qing	831.69	0.0007	0.22
Step 3	0	7.15	0

such as Schwefel 1.2 and Qing. The bad performance of the GWO algorithm for these problems can be attributed to premature convergence and insufficient exploration of the search space. It should be noted that the control parameter values for the PSO and DE algorithms were not extensively tuned, and only a limited number of control parameter combinations were used. However, even though there was not an extensive tuning of control parameters for these algorithms, they achieved competitive results for most of the optimisation problems considered in this report. With proper tuning of parameters the performance of the algorithms is sure to improve even further. Due to the inconsistent performance of the GWO algorithm across different optimisation problems, the PSO and DE algorithms are considered to be superior algorithms with regard to the quality of the best position.

C. Stability

The standard deviations obtained by the three algorithms for the objective value of the best position were compared to see the performance in terms of stability and consistency. Table VI shows the standard deviation for the value of the best position found after 2000 iterations for each of the algorithms with regard to all of the optimisation problems used in this report.

From Table VI it was observed that the GWO algorithm outperformed the other two algorithms for the Rosenbrock, Rastrigin and Sphere functions. Furthermore, the PSO algorithm achieved the best results for the Schwefel 1.2 and Qing functions. Finally, the GWO and DE algorithms outperformed the PSO for the Step 3 function.

As seen through the analysis of the quality of the best position, the performance of the GWO algorithm is very problem dependent. The GWO algorithm achieved competitive results for most of the optimisation problems with regard to stability but for some problems such as the Schwefel and Qing functions it achieved significantly worse results than the other two algorithms. The PSO and DE algorithms performed more consistently across the different benchmark functions with regard to stability. It should be noted that the control parameter values for the PSO and DE algorithms were not extensively tuned. However, the PSO and DE algorithms achieved competitive results for most of the optimisation problems considered in this report regarding the standard deviation of best position objective values. With proper tuning of parameters the performance of the algorithms is sure to improve even further. Due to the inconsistent performance of the GWO algorithm across different optimisation problems, the PSO and DE algorithms are preferred with regard to stability and consistency over different executions of the algorithm.

VI. CONCLUSION

The objective of this report is to compare the performance of the grey wolf optimiser (GWO) algorithm with two other meta-heuristics and see whether it can produce competitive results. This report performs a comparative study of the GWO algorithm with two other popular meta-heuristics, particle swarm optimisation (PSO) and differential evolution (DE). These algorithms are compared over six benchmark single-objective, boundary constrained optimisation problems regarding computational effort, quality of the solutions obtained and stability.

From the results discussed in Section V it is derived that even though the GWO achieves competitive results compared to the other two algorithms on several problems, it is not a reliable method for solving optimisation problems in general. The PSO and DE algorithms achieve more consistent results across the different optimisation problems in terms of the quality of best solutions and stability compared to the GWO algorithm. Furthermore, the DE algorithm is computationally expensive compared to the other two algorithms and thus the PSO algorithm is considered to be the best optimisation technique among the three algorithms studied in this report.

This report can be extended through additional experiments using more benchmark optimisation problems. Furthermore, another comparative study can be done with an extensive tuning for the control parameters of the PSO and DE algorithms.

REFERENCES

[1] F. Azuaje, "Artificial Immune Systems: A New Computational Intelligence Approach", Neural Networks, vol. 16, no. 8, p. 1229, 2003.

[2] M. Dorigo, "Ant colony optimization", Scholarpedia, vol. 2, no. 3, p. 1461, 2007.

[3] D. Fogel, "Evolutionary programming: an introduction and some current directions", Statistics and Computing, vol. 4, no. 2, 1994.

[4] J. Holland, "Genetic Algorithms", Scientific American, vol. 267, no. 1, pp. 66-72, 1992.

[5] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing", Science, vol. 220, no. 4598, pp. 671-680, 1983.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization", Proceedings of ICNN'95 - International Conference on Neural Networks, 1995, vol. 4, pp. 1942-1948, 1995.

[7] J. Koza, "Genetic programming as a means for programming computers by natural selection", Statistics and Computing, vol. 4, no. 2, 1994.

[8] A. Løkketangen, "Tabu Search – Using the Search Experience to Guide the Search Process. An Introduction with Examples", AI Communications, vol. 8, no. 2, pp. 78-85, 1995.

[9] S. Mirjalili, S. Mirjalili and A. Lewis, "Grey Wolf Optimizer", Advances in Engineering Software, vol. 69, pp. 46-61, 2014.

[10] S. Nesmachnow, ”An overview of metaheuristics: accurate and efficient methods for optimisation”, International Journal of Metaheuristics, vol. 3, no. 4, p. 320, 2014.
[11] E. Oldewage, A. Engelbrecht and C. Cleghorn, “The merits of velocity clamping particle swarm optimisation in high dimensional spaces”, 2017.
[12] R. Poli, ”Mean and Variance of the Sampling Distribution of Particle Swarm Optimizers During Stagnation”, IEEE Transactions on Evolutionary Computation, vol. 13, no. 4, pp. 712-721, 2009.
[13] H. Ramalhinho Dias Lourenço, O. Martin and T. Stutzle, ”Iterated Local Search”, SSRN Electronic Journal, 2001.
[14] R. Storn and K. Price, ”Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, Journal of Global Optimization, vol. 11, no. 4, pp. 341-359, 1997.
[15] Y. Shi and R. Eberhart, “A modified particle swarm optimizer”, In Proceedings of the IEEE International Conference on Evolutionary Computation, pages 69–73, 1998.