

AFRICAN BUFFALO OPTIMIZATION

Julius Beneoluchi Odili and Mohd Nizam Mohmad Kahar

Faculty of Computer Systems & Software Engineering, Universiti Malaysia Pahang,
26300 Gambang, Pahang, Malaysia
Email: pcc14004@stdmail.ump.edu.my
Phone: +60169455363; Fax: +6095492144

ABSTRACT

This is an introductory paper to the newly-designed African Buffalo Optimization (ABO) algorithm for solving combinatorial and other optimization problems. The algorithm is inspired by the behavior of African buffalos, a species of wild cows known for their extensive migrant lifestyle. This paper presents an overview of major metaheuristic algorithms with the aim of providing a basis for the development of the African Buffalo Optimization algorithm which is a nature-inspired, population-based metaheuristic algorithm. Experimental results obtained from applying the novel ABO to solve a number of benchmark global optimization test functions as well as some symmetric and asymmetric Traveling Salesman's Problems when compared to the results obtained from using other popular optimization methods show that the African Buffalo Optimization is a worthy addition to the growing number of swarm intelligence optimization techniques.

Keywords: Graphite; African Buffalo Optimization; Metaheuristics; population-based; global optimization; Traveling Salesman's Problem.

INTRODUCTION

A very common consideration in scientific, business and engineering designs is the issue of cost and serviceability, thus, highlighting the need for optimization. Just as business organizations are concerned with maximizing profit, so engineering-design organizations are concerned with continually maximizing the efficiency of the designed products and scientists are continually researching to obtain better results with less input of time and materials. Efficient algorithms are, therefore, required to help solve these problems. This gave rise to the development of some traditional algorithms like the Newton-Raphson (Ben-Israel, 1966), Linear Programming (LP) (Luenberger, 1973), Dynamic Programming (Bertsekas, Bertsekas, Bertsekas, & Bertsekas, 1995), finite elements (Ghanem & Spanos, 2003), finite volume methods (Said & Wegman, 2009) etc. Similarly, lots of relatively-modern algorithms that draw their inspiration from the harmonious, self-organized elements in nature have also been developed. These modern algorithms are classified as Natural Computing.

Natural Computing is simply the process of using the computer to extract ideas from nature to develop computational systems or using natural materials such as molecules to perform computation. From this definition, it is explicit that natural computing can either be drawing inspiration from nature, otherwise called Nature-Inspired Computing (NIC) or computing with natural materials (CWN). Computing with natural materials is one of the most recent computing paradigms where researchers

are working to replace the use of silicons in the development of new software or hardware computational tools to those that use natural media (Zang, Zhang, & Hapeshi, 2010). Nature-inspired optimization algorithms are getting increasingly popular in the past few decades in scientific and engineering research all over the world. Researchers thrilled by this development have adduced several reasons for this: some of these reasons are that they are developed to mimic the most successful dynamics in biological, chemical and physical processes in nature. This situation throws up the issue of choice of algorithm (since we now have several of them) whenever a researcher has an optimization problem to solve. Generally, there is consensus among researchers, that the choice of the 'best' algorithm to solve a particular problem is based largely on the type of problem one is confronted with. This line of thought is reinforced by the No free-lunch theorems for optimization (Wolpert & Macready, 1997). In fact, for large-scale, nonlinear optimization problems, there is no consensus on the recommended guidelines for making a choice of algorithm (Yang, 2011).

The rest of this paper is organised thus: section 2 deals with the review of related literature; section 3 focusses on the swarm intelligence techniques highlighting their individual strengths and weaknesses; section 4 is concerned with the African Buffalo Optimization detailing the nature of the buffalos, their communication and the basic flow of the algorithm; section 5 examines how the African Buffalo Optimization searches divergent search spaces; section 6 validates the algorithm performance in solving Symmetric Travelling Salesman's Problems; section 7 examines the ABO performance in Asymmetric Travelling Salesman's Problems and section 8 draws conclusion on the study.

REVIEW OF LITERATURE

Traditional Optimization algorithms such as the Simplex Method and Newton-Raphson are usually deterministic in nature and use the gradient-based approach. They are very effective in smooth monomodal problems since they use function values and their derivatives in arriving at solutions. However, in situations where there is a break in the objective function, these methods encounter great difficulties. In such situations, non-gradient (derivatives-free) methods which only use function values such as Hooke-Jeeves pattern search and Nelder-Mead downhill simplex are preferred (Haftka & Gürdal, 2012).

The traditional optimization algorithms are very good in solving problems with large number of decision variables. Similarly, these methods require very little problem-specific tuning of parameters. Moreover, in addition to their being rather difficult optimization techniques, they are very inefficient in multimodal search environments. Again, they encounter serious difficulties in solving discrete optimization problems and are very weak in handling situations with numerical noise (Arora, 2004; Vanderplaats, 2007).

Stochastic algorithms

There are broadly two types of stochastic algorithms, namely, Nature-inspired Computing (NIC) and Computing with Nature (CWN). Both types make elaborate use of heuristics and metaheuristics. Heuristic algorithms simply exploit some information about a particular problem in order to solve it. The use of heuristics enables algorithms to obtain quality solutions to difficult optimization problems at a reasonable time.

However, they are near-exact algorithms. That is to say those heuristic algorithms do not guarantee the exact optimal solutions. Metaheuristics, on the other hand simply means 'beyond heuristics' and are generally expected to perform better than heuristics. In general, metaheuristic algorithms employ some local search in addition to global exploration through the use of randomizations. Randomizations help these algorithms to steer away from being restricted in a local environment to a more global search. For the purpose of this study, we will use heuristic and metaheuristic algorithms interchangeably since both are usually non-exact methods but rather employ trial-by-eliminating-errors approach. The overall aim of any metaheuristic algorithm is to achieve the best possible result by using internal mechanisms to achieve adequate exploration and exploitation of the search space(Blum & Roli, 2003).

Metaheuristics, like we shall see in the next section, is generally classified as either population-based or trajectory-based. Population-based metaheuristics can be traceable to Holland whose works used a combination of theoretical genetics and automata methodology. Since 1962 when Holland published his work, several researchers have been attracted to this method of applying variation and diversification techniques to a population to achieve results within a search space. Some of these methods include Schaffer's Vector-Evaluated Genetic Algorithm (VEGA)(Schaffer, 1985); Farmer, Packard and Perelson's Artificial Immune Systems (Farmer, Packard, & Perelson, 1986); Holland's and Rechenberg's Evolutionary Strategies (Eigen, 1973); Dorigo and Di Caro's Ant Colony Optimization ACO (Di Caro & Dorigo, 1998), Grey Wolf Optimizer (Mirjalili, Mirjalili, & Lewis, 2014), to mention a few.

Most population-based techniques exploit some previous knowledge of the solution space and use this in the initialization phase to move the search agents towards the feasible region. In situations where this information is absent, the decision vectors are distributed uniformly within the search space. Based on the foregoing discussion, most population-based metaheuristics have a format similar to Figure 1.

Characteristics of metaheuristic algorithms

Two important features mark out a good metaheuristic algorithms and these are their abilities to employ global search mechanism (otherwise called exploration or diversification) and local search (exploitation or intensification) mechanism in course of a search. While diversification ensures that the algorithm covers as much as it can of the search space within a reasonable search time, the intensification ensures that the algorithm concentrates a closer search possibly around the areas with promising outcomes in addition to selecting the search candidates (decision vectors) that has the best results/solutions. An 'excellent' metaheuristic algorithm, therefore, is one that is able to achieve the best trade off between exploration and exploitation. When an algorithm embarks on too concentrated exploitation, it may likely be trapped in a local minima and so makes it impossible to trace the global optima. Conversely, embarking on too much exploration with very little exploitation will result in the system finding it hard to converge. Similarly, too detailed exploration and exploitation will lead to so much delay at a great cost of users' time and computer resources. In the same vein, too little exploration and exploitation will result in the degradation of the algorithm's efficiency and effectiveness.

Another key feature of a good algorithm is the ability to identify the best solution in a particular iteration and possibly the best design vector associated with such

best solution. This is generally called ‘The survival of the fittest’ criterion. One way of achieving this is to keep updating the current best found so far (Yang, 2011).

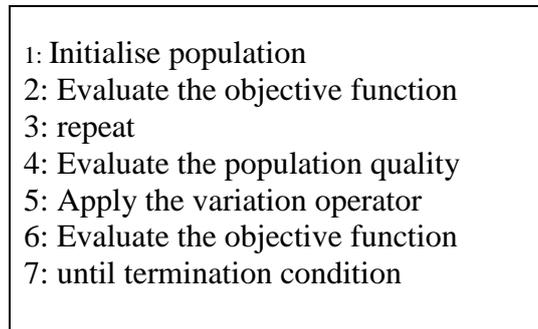


Figure 1. Format of population-based algorithms

Randomization in metaheuristics

Given that the three main concerns of metaheuristic algorithms are exploration, exploitation and identifying the best performer, the mechanism employed by each algorithm to achieve these marks out an algorithm from the others. In general, algorithms achieve these noble goals through randomization in combination with a deterministic procedure. A common mechanism to achieve randomization is to determine the upper and lower boundaries in a uniformly distributed variable between 0 and 1. Algorithms such as Particle Swarm Optimization and Firefly Algorithm employ this method. Other algorithms use the Levy flight. Still some algorithms employ the use of crossover and mutation to achieve the exploration effects. Mutation ensures that new solutions are different from the initial populations (parents) while crossover places a limit on over-exploration (Rani, Jain, Srivastava, & Perumal, 2012). These kinds of algorithms, like the Genetic Algorithm (GA), achieve intensification by generating new solutions around a promising or a superior solution. This could be achieved by employing a random walk as the move in Simulated Annealing (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983) and pitch adjustment in Harmony Search (HS) algorithm (Mahdavi, Fesanghary, & Damangir, 2007):

Aside handling the issue of intensification and diversification, another important characteristic of a good optimization algorithm is identifying and preserving the decision vectors (individuals) with good solutions while at the same time discarding bad performers. This elitism is helpful, especially in multimodal and multi-objective problems. Elitism in GA and P_g in Particle Swarm Optimization are good examples. Discarding of worse solutions is achieved by some form of randomization and probabilistic selection criteria such as mutation in Genetic Algorithms and casting away of an unproductive nest in Cuckoo Search algorithm (Gandomi, Yang, & Alavi, 2013).

Classification and taxonomy of metaheuristic algorithms

Metaheuristic algorithms have been classified in several ways in literature. One of such ways is to classify them as either population-based or trajectory-based. Population-based optimization techniques solve problems using a number of decision vectors at a time. An example of this is the GA which uses a set of strings. So also is Particle Swarm that uses a number of agents or particles (Kennedy, 2010). On the other hand,

Simulated Annealing and Hill Climbing are trajectory-based and use a single agent that moves through the search space in a piecewise fashion (Kennedy, 2010; Mirjalili et al., 2014).

Another way to classify metaheuristic optimization algorithms is to identify them as either Evolutionary or deterministic. (Venter, 2010). A popular general-purpose deterministic algorithm is the DIRECT algorithm which makes use of Lipschitzian technique to identify promising subregions in the search space. (Olafsson, 2006). For the purpose of this paper, however, our interest is the Population-based optimization techniques. At this juncture, let us attempt to a taxonomy of the Nature-inspired algorithms. This attempt is by no means exhaustive but will give a clearer picture of these optimization techniques (Figure 2).

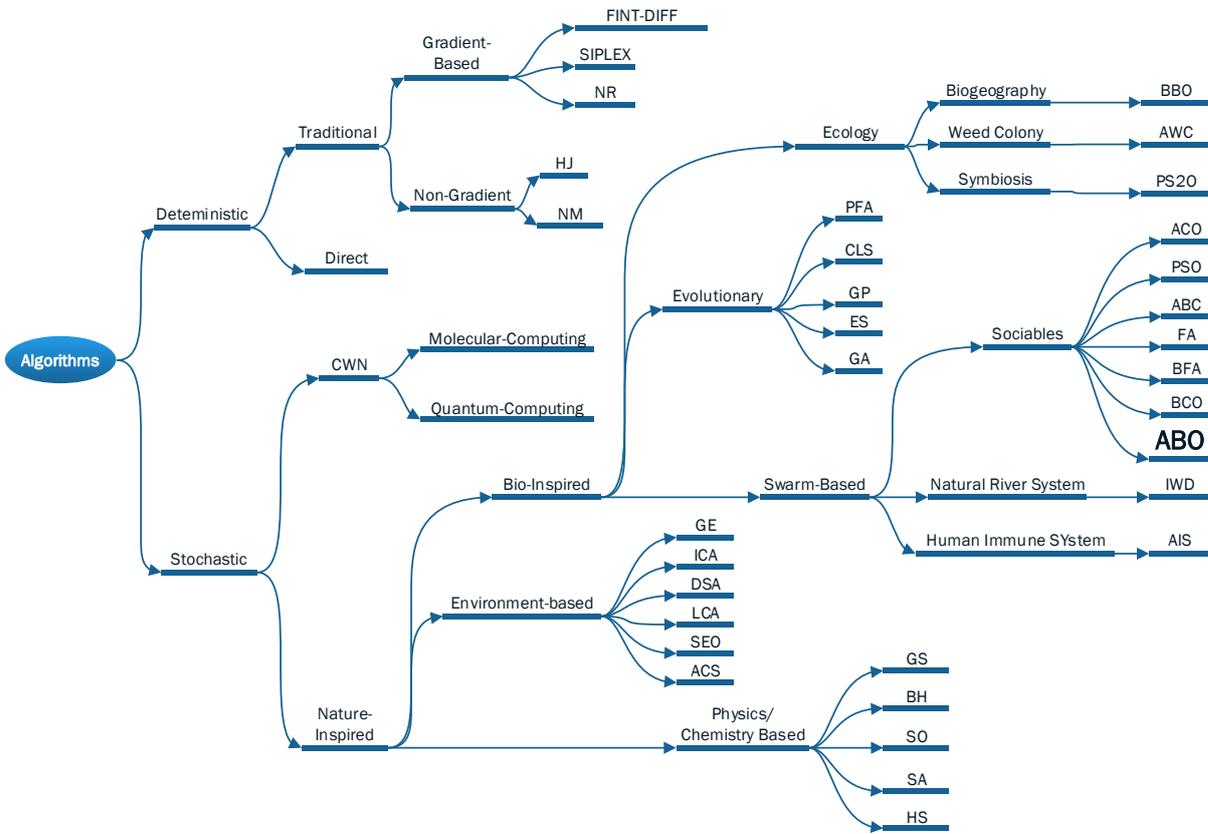


Figure 2. Taxonomy of Algorithms

CWN=Computing with Nature
 FINT. DIFF =Finite Difference
 GA= Genetic Algorithm
 N.R = Newton-Raphson method
 CLS= Classifier Systems
 GP= Genetic Programming
 LCA= League Championship Algorithm
 DSA= Differential Search Algorithm
 ICA= Imperialist Competitive Algorithm
 GE= Grammatical Evolution

HJ= Hooke-Jeeves pattern
 NM= Nelder-Mead simplex
 ES= Evolution Strategies
 BBO= Biogeography-Based Optimization
 FA= Firefly Algorithm
 PFA= Paddy Field Algorithm
 BCO= Bee Colony Optimization
 AWC= Artificial Weed Colony
 BFA=Bacteria Foraging Algorithm

GS= Gravitational Search
 BH= Black Hole
 SO= Spiral Optimization
 SA= Simulated Annealing
 HS= Harmony Search
 ABO= African Buffalo Optimization

IWD= Intelligent Water Drop
 AIS= Artificial Immune System
 ACO= Ant Colony Optimization
 PSO= Particle Swarm Optimization
 ABC= Artificial Bee Colony

Swarm-based techniques

Swarm Intelligence is based on collective social interactions of organisms and involves the implementation of collective intelligence of groups of simple agents such as ants, animals, plants and other elements in our ecosystem based on their behavior in real-life situations. Since the African Buffalo Optimization belongs to the Swarm Intelligence group of algorithms, it is pertinent to do a critical review of some major algorithms in this sub-category of nature-inspired algorithms.

Particle swarm optimization

Particle swarm optimization (PSO), inspired by the flocking of birds and schooling of fishes in search for food, was proposed by Kennedy and Eberhart as a population-based, stochastic, global optimization technique with the aim of being simple, easy to implement yet efficient search technique. Till date, it has proven to be a very successful algorithm enjoying wide applicability in several problem domains. The PSO harnesses the velocity and positions of these simple ants in its search for solutions. Each agent called particles in PSO represents a solution to the problem being solved using five major parameters, namely, the current position, the particles' best position, the best position found by its neighbor, the individual particle's knowledge of the its best position achieved so far as well as the its current velocity. Using a simple rule, a particle updates its velocity and position with each iteration as the algorithm progresses until it reaches termination condition. Moreover, an information repository is maintained that documents the best achieved objective function values for each particle involved in the search process

PSO models the behavior of, for example, a swarm of birds searching for a food source. The entire particles converge on the best solution through the use of the information gathered by each particle, the neighboring particle as well as that obtained from the entire flock. The algorithm starts by initializing the particles in the search space, followed by updating the position and velocity of each particle in each iteration. The updating rule is

$$X_i^{u+1} = X_i^u + V_i^u * \Delta t \quad (1)$$

$$V_i^{u+1} = V_i^u + C1r1 (P^i_k - X^i_k) + C2 r2 (P^g_k - X^i_k) \quad (2)$$

where i represents the i th particle in the swarm; u represents the u th iteration and V_i^u is the velocity of i th particle in u th iteration; Δt is a unit of time. At the beginning, each particle is given random position and velocity that gets updated with each construction step.

In Eq. (10), V_i^u represents the particle's velocity; C_1 and C_2 represent the cognitive and social factors; r_1 and r_2 represents the random numbers; P^i_k is the

individual particle's best position; P_k^g is the best particle in the herd; and X_k^i is the position of particle i . Basically, the action steps in PSO algorithm are as in Figure 3.

1. Initialize the swarm by assigning random position and velocity to each particle.
2. Evaluate the fitness function for each particle.
3. For each particle, compare the particle's fitness value with its personal best so far. If the present fitness value (obtained from step 2 above) is better than the personal best value obtained from previous iterations, then set this present best fitness value as particle's best and the current particle's position, x_i , as the best position so far. Also identify P_i 's best performing neighbor
4. Identify the particle in the entire swarm that has the best fitness value so far as set it as the group best fitness.
5. Update the velocities and positions of all the particles using (2) and (3).
6. Repeat steps 2–5 until you reach the stopping criteria.

Figure 3. PSO pseudocode

Comment: A critical examination of the PSO algorithm reveals that classical PSO algorithm is easy to implement and there are relatively fewer parameters to adjust than in some algorithms like the ACO. It is important to make this assessment based on the classical PSO because there have been several modifications of the initial PSO of late with the introduction of more parameters in a bid to achieve better results (Clerc & Kennedy, 2002).

Moreover, PSO has a more efficient memory capability than algorithms like the GA. Also, PSO achieves better diversity in exploring the search space since all the particles use the information obtained by the best particles in each iteration to improve their locations and speed. This search scheme is diametrically opposed to GA, for instance, where the worse solutions are discarded and the population only revolves around the fittest individuals.

PSO shares some similarities with algorithms like the GP, ES and GA through the initialization of solutions and the update of generations. Though the PSO does not use evolution operators like mutation, crossover, inversion and selection, particles aim at the optimum by following the global best particle. On the whole, PSO has been proven to be effective in searching continuous functions and in multimodal search environments (Bonabeau, Dorigo, & Theraulaz, 1999). However, PSO uses several parameters and this affects the speed and efficiency of the algorithm.

Ant colony optimization (ACO)

This is one of the most popular metaheuristic algorithms in literature. The Ant Colony Optimization algorithm was developed by Marco Dorigo and Di Caro in 1999 after some initial work on the Ant System by Marco Dorigo and Ant Colony System by Dorigo and Gambardella. The Ant Colony Optimization was inspired by the random walk of ants in search of food. Once a food source is located, the ant that discovered the food carries a particle of the food and as it returns to the nest, usually follows a shorter route and continually deposits some amount of pheromones as a way of informing other ants of its breakthrough. The neighboring ants, perceiving the scent of the pheromone will likely join the successful ant to track the food source. Once these ants get to the food source, they, in turn, carry some fragments of the food and on their way back to the nest, drop pheromones as they further optimize the route of the initial ant. This

process increases the pheromone concentration on the favourite ‘shortest’ route and in that process attract other ants. Within a short while, the colony of ants are on the optimized route moving to and from the food source. On the other hand, when an ant arrives at the food and could not find food, may be, because that food is exhausted or the particular ant missed the direction of the food source, on its way back it drops no pheromone and so with time that route loses its attraction due to pheromone evaporation. This situation leads the ants to explore other areas.

The ACO algorithm is designed by a process of simulating the ant foraging behavior, brood sorting, nest building and self-assembling. The ACO has three main functions, namely, ConstructAntSolutions, Pheromone Update and DeamonActions.

(i)*ConstructAntSolutions*: This function performs the construction of the the solution. Here the artificial ants move through adjacent states of a problem according to a transition rule and, in this way, gradually build a solution iteratively.

(ii)*Pheromone Update*: This function updates pheromone trails either through pheromone reinforcement or evaporation. The is done in two ways depending on the variant of the ACO one is working with. The pheromone update function could be done at the end of each iteration or when the ants individually completes a solution.

(iii)*DeamonActions*: This function is rather optional depending on the problem being solved. It involves increasing the pheromone levels to select promising edges (Kumbharana & Pandey, 2013). See Figure 4.

Step 1: Initialise pheromone values τ for all the edges in the graph. For a start, all the edges should have equal amount of pheromone unless there exist some heuristic information favouring some edges that may lead to speedier convergence.
 Step 2: Construct a solution for each ant x . $X = (1, 2, \dots, N)$.
 Step 3: Update the pheromone values for each edge depending on the quality of solution.
 Step 4 Go to Step 2 until stopping criterium is reached.

Figure 4. ACO pseudocode

Comment: The ACO has proven to be a robust algorithm that can be easily hydridized with other algorithms to enhance efficiency. Moreover, it is efficient in search situations where the graph is prone to dynamic changes. Also ACO performs well in distributed computing environments. However, it has the weakness of easily falling into premature convergence because its pheromone update is according to the present best path. Again, it uses several parameters that require proper tuning. Such parameters as pheromone quantity, pheromone update rule, evaporation rate, pheromone reinforcement rate etc [28, 27].

Artificial Bee Colony (ABC)

This is inspired by the behavior of natural honey bee swarm and was proposed by Karaboga and Basturk in 2009 (Karaboga & Akay, 2009). This algorithm categorizes bees into three: scout bees, onlooker bees and employed bees. A scout bee is one that flies around the search place seeking solutions (food source); an onlooker bee is one that is in the nest waiting for the report of the scout bee and an employed bee is one that

after watching the waggle dance of the scout bee opts to join in harvesting the food source. A scout bee transform to an employed bee once it (the same scout bee) is involved in harvesting the food source. In this algorithm, the food source represents a solution to the optimization problem. The volume of nectar in a food source represents the quality (fitness) of the solution.

When the employed bee brings some nectar to the hive, it has three options: return to get more nectar, accompany other dancing bee to a new site or simply stay back at the hive. The bee's decision is informed by a number of factors ranging from the distance of the food source to the hive, the quality and quantity of the available nectar at the food source and the number of employed bees harvesting the nectar at that food source. The global search mechanism of the ABC depends of the random search capacity of the scout bees while the exploitation capacity of the algorithm is based on the activities of the employed and onlooker bees. This bee behavior can be replicated in some real life problems, such as in transportation and business applications (Pham et al., 2011). For the pseudo-code of ABC, please refer to Figure 5.

- (1) Initialize population with random solutions.
- (2) Evaluate fitness of the individual bees in the population.
- (3) While (stopping criterion is not reached), form new population.
- (4) Choose site(s) for the next neighborhood search.
- (5) Select bees for selected locations to be searched ensuring that more bees are allocated to the best locations & evaluate fitness.
- (6) Choose the fittest bee from each patch for the next best harvesting site(s).
- (7) Assign remaining bees to search randomly and evaluate their fitness.
- (8) End While.

Figure 5. ABC pseudo-code

Comment: Previous studies on the ABC has shown that it is very effective in feed-forward artificial neural networks training in addition to the algorithm being very efficient in multidimensional search environments due to its capacity to get out of a local minimum with ease. However, studies on the Artificial Bee Colony are still not widespread [31-33] and there are several parameters to appropriately tune in order to get good results.

Bee Colony Optimization

In this algorithm, a population of artificial bees conducts a search for the optimal solution(s). Each bee generates one solution to the problem. The Bee Colony Optimization algorithm makes use of two alternating phases, namely, forward pass phase and backward pass phase. In the forward pass phase, the bees explore the search space through a number of moves that construct or improve a solution and thereby creating a new solution. After arriving at this partial solution, the bees return to the nest to initiate the backward pass phase where they communicate information about their newly-found solutions. They communicate the distance as well as the quality of the food source (solution) to the nest through a waggle dance. Based on the information collected at this phase, the bees (including the dancer) decide, with a certain probability, to abandon the newly-found solution and remain in the nest or to follow the dancing bee to the food source. This leads to the second forward pass where the bees improve on the

previously found solutions and after which they return to another backward pass phase. These phases are performed repeatedly until a stopping criterion is reached. The stopping criterion could be the maximum number of forward/backward pass phases, the maximum number of forward/backward phases without tangible improvement of the objective function or the arrival at the optimal solution. Basically, the pseudo-code of the Bee Colony Optimization algorithm is as in Figure 6.

1. Initialization: set every bee to an empty solution; (some previous knowledge of the problem is helpful here)
2. For each bee, perform the forward pass:
 - a) Set $k = 1$; k is a counter for constructive moves in the forward pass;
 - b) Evaluate all possible constructive moves;
 - c) According to the results from the evaluation, select one move using the roulette wheel;
 - d) $k = k + 1$; If $k \leq NC$ ($NC = \text{maximum number of counters}$), return to step 2b.
3. Let the bees start the backward pass phase;
4. Based on the objective function value of each bee, sort the bees;
5. Let each bee decide, randomly, whether to continue its own exploration and become a recruiter, or to become a follower of bees with higher objective function value;
6. For every follower bee, choose a new solution from recruiter bees using a the roulette wheel;
7. If the stopping criterion is not reached, return to step 2;
8. Output the best result.

Figure 6. BCO pseudo-code

Comment: The mechanism that informs the decision of a bee to follow a particular dancer is not well established but it is rather vaguely considered that the decision to follow a particular bee is a function of the food source [34]. However, the abandonment phenomenon is of great benefit to the algorithm. That is to say that when the employed bees could not find the optimized solutions after some repetitions, they transform to the scout bees again and move in random paths to start searching for optimized solutions. This way, the solutions which are not optimized are abandoned and further search are made for the global optimized points. This procedure helps the algorithm not to fall into a local optima or minima in a multi-dimensional search environment. As a result, it could be safe to say that the artificial bees use a combination of of local and global searching methods to arrive at solutions.

It should, however, be observed that that the BCO has complicated fitness function. As such, obtaining good results is a fuction of proper setting of parameters such as minimum overshoot, rise time, steady state error and settling time in the state response. Moreover, it has been observed that Bee algorithm is not as adaptive as ACO. Finally, BCO algorithm shows poor performance and remains inefficient in exploring search space because its search equation is significantly influenced by a random quantity which helps in exploration at the cost of exploitation of the search space [35].

AFRICAN BUFFALO OPTIMIZATION (ABO) ALGORITHM

At this juncture, one may be tempted to ask: ‘Since we have several algorithms already, why then do we need to add more?’ A critical review of the algorithms discussed so far, in this study, reveals that the algorithms exhibit exceptional strength in solving different problems. However, they are plagued by several inadequacies ranging from premature convergence, the use of several parameters and inefficient exploration of the search space, complicated fitness function to delay in obtaining optimal or near-optimal solutions [32]. The African Buffalo Optimization is, therefore, an attempt to complement the existing algorithms with the aim of solving some of the perceived weaknesses of the earlier algorithms, especially the problems of delay and inefficiency.

African buffalos

The Africa Buffalo is Africa's only wild cow and is of the same species with the domestic cattle. They are referred to as Buffalo, African Buffalo, Cape buffalo, Forest buffalo, and Savannah buffalo. They are usually very strong, imposing, and fierce-looking. Most times, their colour is usually blackish-grey or dark-brown. Moreover, both males and female have upturned horns, meeting at a central protective shield on their forehead, called boss. These African Buffalos live in herds of up to 1,000 individuals. These large animals are unique in their ability to defend themselves and to help one another in times of danger, especially when under attack by a pride of lions. They are about the only known animal that dare to confront a lion in defence of one of their own in danger. This do by responding to the distress/alert call from a buffalo in danger or other other buffalos observing the incidence. In response to the distress waaa calls, the herd will group together to defend another of their kind that is under attack from a predator.

Their famed cooperative abilities is further highlighted by their voting behaviour in decision making. It is believed that the buffalos consult widely among themselves in taking decisions about the choice of of grazing fields. In a study, it was discovered that their movements are controlled by the majority decisions. This they do by standing up and gazing at a particular direction in turns. When the average direction of gaze is compared with the subsequent movement of the herd, the average deviation is only three degrees, which is well within measurement error. On days in which cows differ sharply in their direction of gaze, the herd tends to split and graze in separate patches for the night (Stueckle & Zinner, 2008).

In addition to this evidence for communal decision making, there is no evidence for individual leadership. For example, no individual cow or bull stays in the vanguard of the herd for more than a few minutes. Prins (Prins, 1996) regards communal decision making in buffalo herds as similar to the famous bee dance, in which individuals also communicate the direction of beneficial resources to each other (Wilson, 1997).

Communication among African buffalos

African buffalos use two main sounds to organise themselves in their constant migrant lifestyle in search of grazing lands. They emit low-pitched, two- to four-second calls intermittently at three- to six-second intervals to signal the herd to move. To signal to the herd to change direction, buffaloss will emit “cracky” waaa sounds. When moving to drinking places, some individuals make long ‘maaa’ calls to summon the herd. When

being aggressive, they make explosive grunts that may last for a couple of seconds or turn into rumbling growl. Cows produce croaking calls when looking for their calves. Calves will make a similar call of a higher pitch when in distress. When threatened by predators, they make a long drawn-out ‘waaa’ calls. Dominant individuals, especially males, make long intimidating ‘maaa’ calls to announce their presence and location. A more intense version of the same call is emitted as a warning to ward off an encroaching inferior.

African buffalo optimization

African Buffalo Optimization (A.B.O) is a simulation of the alert (‘maaa’) and alarm (‘waaa’) calls of African buffalos in their foraging assignments. The waaa calls is used to warn the buffalos of the presence of predators, ward off an approaching inferior, assert dominance or express the lack of pastures in a particular area and therefore urge the herd to move on to safer or more rewarding areas (exploration). Whenever this call is made, the animals are asked to be alert and to seek a safer or better grazing field. The maaa calls is used to encourage the buffalos to be relaxed as there are good grazing fields around, reassure an inferior and to express satisfaction about the amount of pastures cum favourable grazing atmosphere at a particular location (exploitation). With these sounds, the buffalos are able to optimize their search for food source. The ABO is a population-based algorithm in which individual buffalos work together to solve a given problem. Using the waaa (move on) signal or the maaa (hang around) signal, the animals are able to obtain amazing solutions in their exploration and exploitation of the search space.

This study is an attempt to develop a robust, fast, efficient, effective, yet simple-to-implement algorithm that has enormous capacity to explore and exploit the search space by simulating the communicative and cooperative characteristics of the African buffalos in their search for solutions. It solves the problem of premature convergence by regularly monitoring and updating the location of the best buffalo in the herd. In a situation where the best buffalo is not updating in a given number of iteration, the entire herd is re-initialized. This helps to ensure adequate exploration. The problem of slow speed is handled with the African Buffalo Optimization’s use of very few parameters, primarily the learning parameters (lp_1 and lp_2). The issue of adequate exploration and exploitation of the solution space is further enhanced with the democratic equation (Eq.3) where the animals regularly communicate with one another.

The ABO algorithm

In Figure 7, the $w_{.k}$ represents the waaa (move on / explore) signals of the buffalos with particular reference to buffalo k ; $m_{.k}$ is the maaa (stay to exploit); $w_{.k+1}$ is the request for further exploration; $m_{.k+1}$ represents a call for further exploitation; lp_1 and lp_2 are the learning parameters.

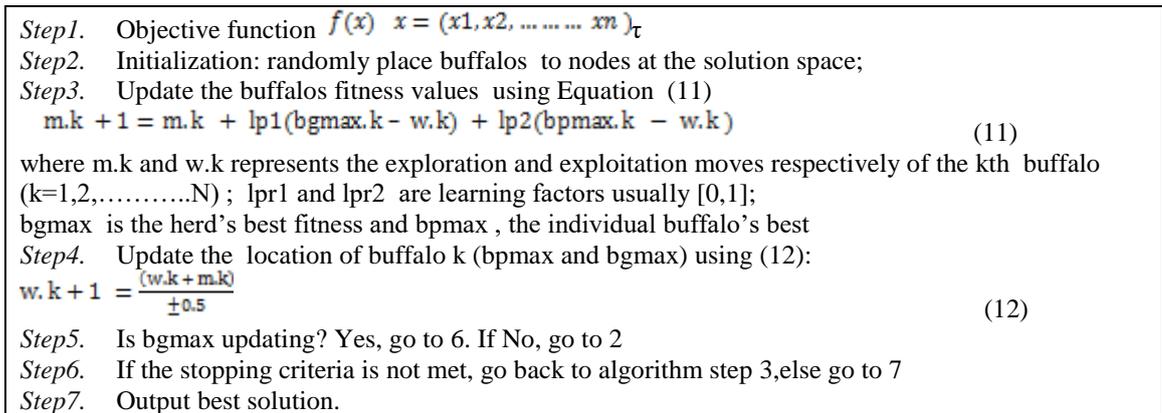


Figure 7. ABO algorithm

Mathematical model

African Buffalo Optimization algorithm, basically, models the three principal aforementioned characteristics of the African Buffalo, namely, excellent memory, regular communication and exceptional intelligence. The *waaa* calls of buffalo k ($k=1, 2, 3\dots n$) which propels the buffalos to move on to explore other grazing areas since where they are presently is unsafe or lacks sufficient pasture is represented by $w.k$ and the *maaa* sound which asks the animals to stay on to exploit the present location because it has enough pasture and is safe is represented by $m.k$. Mathematically, the democratic equations which determines the movement of the buffalos is:

$$m.k + 1 = m.k + lp1(bgmax - w.k) + lp2(bpmax.k - w.k) \quad (11)$$

The democratic Equation (3) has three main parts, namely, the memory part ($m.k + 1$) which is an indication that the animals are aware that they have relocated from their former positions ($m.k$) to a new one. This is an indication of their extensive memory capacity which is a vital tool in their migrant lifestyle. The second part represents the cooperative attributes of the animals $lp1(bgmax - w.k)$. The buffalos are excellent communicators and are able to track the location of the best buffalo in each iteration. The last part of this equation ($lp2(bpmax.k - w.k)$) brings out the exceptional intelligence of these animals. They are able to tell their previous best productive location in comparison to their present position. This enables them take informed decisions in their search for solutions. Eq. 12, basically, propels the buffalos to a new location following the outcome of Eq. 11.

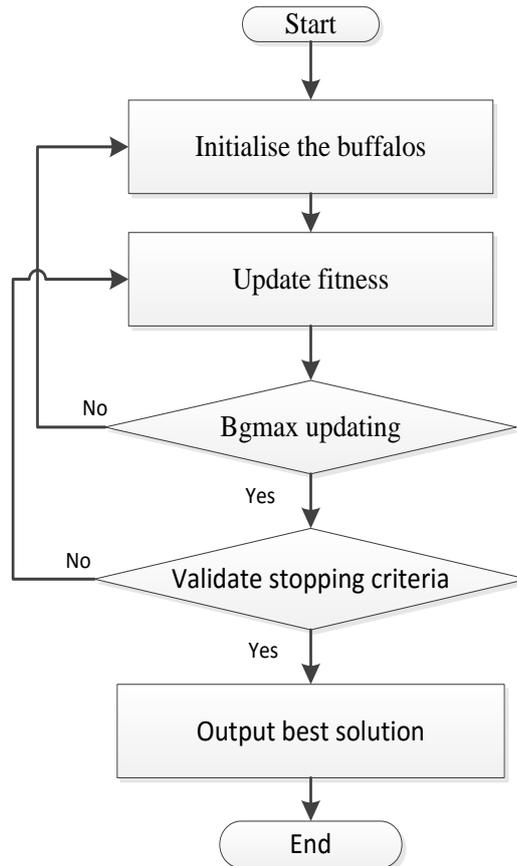


Figure 8. ABO flowchart

The working of the ABO

The algorithm begins by initializing the population of buffalos. It does this by assigning a random location to each buffalo within the search space. Next, it updates each buffalo's fitness in relation to the target goal and in this way ascertain the herd's best animal (bgmax) and each buffalos personal best (bpmax.k). In each step, each animal keeps a memory of its coordinates. If the present fitness is better than the individual's maximum fitness (bpmax), the algorithm saves the location vector for the particular buffalo. If the fitness is better than the herd's overall maximum, it saves it as the herd's maximum (bgmax). Next, the algorithm updates the location of the buffalos using Eq.4. After this, it confirms the improvement or otherwise of the leading buffalo(bgmax).

If there is no improvement in a number of iteration, the algorithm re-initializes the entire herd. If the best buffalo is improving its positions, the algorithm checks to see if the stopping criteria is reached. If our best buffalo fitness (bgmax) meets our exit criteria, it ends the run and provides the location vector as the solution to the given problem. Some of the high points of this new algorithm are its ease of implementation, ability to search both locally and globally at the same time, use of relatively few parameters, flexibility and fast rate of convergence. The Algorithm flowchart is presented in Figure 8.

NUMERICAL TESTS

Numerical investigations are very necessary in testing the capacity of novel algorithms to arrive at solutions. The new algorithms should be investigated for speed, convergence and robustness (Baritomba & Hendrix, 2005). In this section, the test for speed is done by Average number of Function Evaluations (‘AFE’) taken to arrive at the optimal solution; convergence by the relative error between the optimal solution obtained by the algorithm (‘Best’) and the Optimal result of the function (‘Opt’) and robustness by testing different search spaces ranging from Unimodal non-separable, Multi-modal non-separable to Multi-modal separable. A comparative lower relative error, therefore, is indicative of a better algorithm. To do this, six benchmark global optimization functions were selected due to their diversities and difficulties they pose to search algorithms. Two functions are selected in each category: from unimodal and separable, we chose Brown 1 and Brown2; from multi-modal and non-separable functions, we chose Hartman3 and Branin functions and from multimodal and separable, we chose F1 and F3 functions (Ali, Khompatraporn, & Zabinsky, 2005).

The experimental results obtained were compared with those from Improved Genetic Algorithm (IGA) and the Staged Continuous Tabu Search (SCTS) algorithm (Zheng, Ngo, Shum, Tjin, & Binh, 2005). These algorithms chosen to compare with the ABO in this study have posted some of the best results in literature. The detailed comparison is shown in Table 1.

Table 1. Global optimization search results

Func= Function name; Opt= optimal result of the benchmark function under investigation; Xter=Characteristics; Var=number of variables required by each function; Best= Algorithm’s best result; Suc%= Percentage of Successful runs; AFE= average number of Function Evaluation;

Func	Opt	Xter	Var	ABO				IGA				SCTS			
				Best	Rel Err%	AFE	Suc %	Best	Rel. Err %	AFE	Suc %	Best	Rel Err%	AFE	Suc %
Brown1	2	UN	20	2.273	13.6	8039	87	8.552	327.6	128644	0	2.008	0.05	111430	100
Brown3	0	UN	20	0.001	0.1	4000	99	0.674	67.4	106859	5	0.006	0.009	15142	100
Hartm3	-3.863	MN	3	-3.863	0	91	100	-3.861	0.05	1680	100	-3.86	0.018	560	100
Branin	0.3978	MN	2	0.398	0	153	100	0.398	0.25	2040	100	0.398	0.025	492	100
F1	-1.232	MS	1	-1.232	0	44	100	-1.232	0	784	100	-1.23	0.03	134	100
F3	-12.03	MS	1	-12.03	0	5347	100	-12.03	0	744	100	-12	0.001	181	100

In Table 1, The Relative Error is obtained by:

$$Rel. Error = \frac{|Best - Opt|}{Opt R} * 100 \tag{13}$$

In a situation where the optimal result of a function is 0, the formula for obtaining is:

$$Rel. Error = |Best - Opt| * 100 \tag{14}$$

Table 1 shows the capacity of ABO to obtain competitive results in searching diverse search spaces. The novel algorithm obtained optimal results in four of the six test functions; with IGA obtaining the optimal solution in two out of the six test cases and SCTS obtaining none. ABO’s capacity to escape local minima is traceable to her frequent updating of the fitness of each buffalo in relation to the best buffalo in the herd. Another strength of the ABO as can be observed here is its speed in obtaining solutions. This is evaluated using the average number of function evaluations (AFE). In all the test instances, the ABO is faster in five of the six functions than the other algorithms. The cumulative AFE of the ABO is 17,674 to IGA’s 240,771 and SCTS’s 127,939. This shows that ABO is 1362% faster than the IGA and 724% faster than the SCTS (See Figure 9).

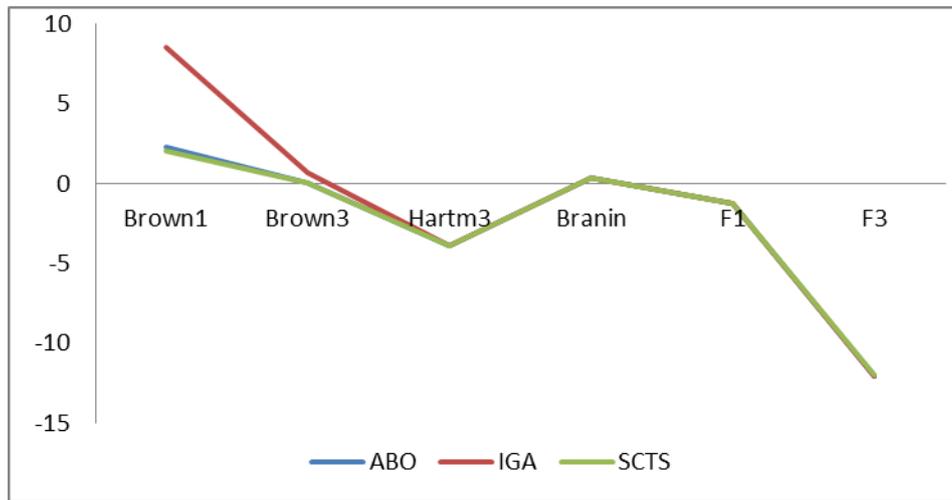


Figure 9. Global optimization line graph

APPLICATION OF THE ABO TO SOLVE THE SYMMETRIC TSP

In applying ABO to solve the TSP, the basic solution steps are:

- (a.) choose, according to some criterion, a start city at which the buffalos are positioned;
- (b.) use the democratic Eq. 3 and heuristic values to probabilistically construct a buffalo tour by adding cities that the animals are yet to visit.
- (c.) calculate buffalo fitness using Eq. 11 & move the animals to the next node with Eq. 12.
- (d) repeat steps b and c until the buffalos complete the tour
- (d.) go back to the initial city.
- (e.) after all buffalos have completed their tour, determine the best route using Eq. 5

$$P_{ab} = \frac{w^{lp_1} m^{lp_2}}{\sum_{i=1}^n w^{lp_1} m^{lp_2}} \quad (15)$$

$$ab = \pm 0.5$$

In Eq. 5, lp_1 and lp_2 are learning paramters >0 , m is the ‘maaa’ call asking the animals to hang around to exploit edge ab , w is the ‘waaa’ call asking the buffalos to explore edge ab . For buffalo k , the probability of buffalo k moving from city ‘a’ to city ‘b’ depends on the combination of two values, viz., the desirability of the move, as computed by some heuristic indicating the prior attractiveness of that move and the summative benefit of the move to the entire herd, indicating how proficient it has been in the past to making that particular move. The denominator values represent the post

indication of the desirability of that move. The paths are improved usually when all buffaloes have completed their solution. The algorithm parameters are set as population=40, $lpr_1 = 0.6$, lpr_2 as 0.5.

Experimental work on TSP

In our next set of experiments, we examined the performance of ABO on some unpopular TSP benchmark datasets. We compared our output with the results obtained by using Honey Bee Mating Optimization (HBMO). We chose to compare our work with the HBMO because this (HBMO) study presently holds the best TSP result in literature (Marinakis, Marinaki, & Dounias, 2011). The result is presented in Table 2.

Table 2: Comparative results of ABO and HBMO

TSP CASES	ABO				HBMO		
	OPT	Best	Rel. Error (%)	CPU Time (s)	Best	Rel. Error (%)	CPU Time (s)
A280	2579	2579	0	0.07	2579	0	8.5
Fl417	11861	11862	0.01	0.03	11861	0	24.67
Pr439	107217	107340	0.11	0.03	107217	0	36.21
Pcb442	50778	50799	0.04	0.06	50778	0	37.12
D493	35002	35024	0.06	0.07	35002	0	45.21
Dsj1000	18660188	18680771	0.11	0.04	18660556	0	80.29
Pr1002	259045	259132	0.03	0.04	259045	0	80.57
U1060	224094	224481	0.17	0.05	224094	0	80.68
Vm1084	232929	232931	0	0.08	232929	0	85.21
Pcb1173	56892	56918	0.05	0.08	56892	0	89.23
D1291	50801	50839	0.07	0.05	50801	0	91.14
Nrw1379	56638	56653	0.03	0.05	56638	0	121.89
Fl1577	22249	22249	0	0.06	22249	0	222.27
Vm1748	336556	336642	0.03	0.07	336712	0.02	257.81
U1817	57201	57201	0	0.07	57201	0	289.12
D2103	80450	80456	0.01	0.08	80450	0	350.78
U2152	64253	64258	0.01	0.12	64267	0.02	357.23
Pcb3038	137694	137700	0	0.14	137694	0	457.29
Fl3795	28772	28772	0	0.21	28783	0.04	461.81
Rl 5915	565530	565800	0.05	0.49	565530	0	557.87
Rl5934	556045	556078	0.01	0.52	556080	0.01	561.21
Pla7379	23260728	23268269	0.03	0.7	23261400	0	780.31
Usa13509	19982859	19993952	0.06	23.54	19982849	0	897.09
Fn14461	182566	182745	0.1	0.28	182612	0.03	498.01
TOTAL				26.93			6471.52

In Table 2, both ABO and HBMO were able to obtain above 99.98% accuracy in all the 24 benchmark TSP instances ranging from 280 to 14,461 nodes/cities which were under investigation. These are commanding performances.

In comparing the computational costs of obtaining results, there is such a gulf in the time taken (and by implication, computer resources) to obtain results: while it took ABO 26.93 seconds to solve all the 24 problems, it took HBMO a whopping 6,471.52 seconds. This shows ABO is more than 240 times faster than the HBMO and since speed is a measure of good algorithm, ABO is a better algorithm. This fact becomes clearer when one observes that in spite of ABO's speed, the novel algorithm was able to obtain over 99.98% accuracy in all the test cases. The speed of the ABO is traceable to its use of relatively fewer parameters to most other popular algorithms in its search for solutions. The ABO's controlling parameters are the learning factors (lp_1 and lp_2). These parameters coupled with the intelligence and communication abilities of the buffalos are effective tools of the algorithm in achieving good results.

ABO ON THE ASYMMETRIC TRAVELLING SALESMAN'S PROBLEM

It is pertinent, at this juncture, to test the performance of the African Buffalo Optimization (ABO) on the Asymmetric Travelling Salesman's Problems. To do this, we carried out experiments on 15 out of the 19 benchmark optimization problems available on TSPLIB95 (Reinelt, 1991). The choice of the datasets is informed by their complexity and popularity in literature. The results obtained from this exercise was compared with those obtained from three other heuristic algorithms available in Tsp-solve (Hurwitz & Craig) . The comparative algorithms are Addition heuristics, Assign heuristics, Loss and Patching heuristics (Rocha, Fernandes, & Soares, 2004). The Addition Heuristics employs the construction method in its search and tour development; the Loss Heuristics uses a technique described in (Van der Cruyssen & Rijckaert, 1978) and the Patching Heuristics engages in solving an assignment problem and later integrates the subtours into one tour using the patching technique (Karp & Steele, 1985)

This comparison is relevant because there are three basic methods of solving asymmetric Travelling Salesman's Problem in literature and these are the Constructive method, the Improvement method and the Composite method that uses a combination of the first two (Helsgaun, 2000). The ABO solves the asymmetric Travelling Salesman's Problem using the composite method. It starts by using the constructive method, especially for problems involving less than 100 nodes but turns to improve upon the construction as the number of nodes increases. This, coupled with the ABO's use of very few parameters enables the ABO to arrive at solutions faster than many other methods. Basically, tour construction method builds a tour by simply adding a new node that has not been added or visited at each step/iteration. When the tour has been constructed, the buffalos return to the starting node, avoiding any already visited node as much as possible. An example of a method that uses strict Construction technique is the nearest neighborhood algorithm (Suchal & Návrat, 2010) . Meanwhile, the tour improvement method gets a tour improved through making improvements/exchanges on the already existing tours. Examples are the 2-opt algorithm and the Lin-Kernighan algorithm (Helsgaun, 2000). The composite method, on the hand, as in ABO, starts solving the problem by constructing a tour through the addition of unvisited nodes and then performs improvement exchanges depending on the location of the best buffalo.

This helps the algorithm to arrive at better solutions. The result is presented in Table 3 below.

As close look at Table 3 shows that the ABO outperforms the other methods in obtaining the optimal solutions. The ABO obtained optimal solution in three ATSP instances: Ftv38, Ftv53 and Ftv64 in addition to obtaining over 99.5% accuracy in the remaining instances. Meanwhile the other methods were only able to obtain optimal result in one instance each and that is Br17. Their performances in other instances were rather not encouraging. For instance, the cumulative relative error percentage of the ABO in all ATSP instances is 1.53% to Addition Heuristics 102.52%; Loss Heuristics 59.43 and Patching Heuristics 67.68%. From this analysis, it is obvious that ABO outperformed other methods.

Table 3. ABO and other Heuristics on ATSP

TSP instance	Opt	ABO			Addition Heuristics			Loss Heuristics			Patching Heuristics		
		Best	Rel. Err%	Iter	Best	Rel. Err%	Iter	Best	Rel. Err%	Iter	Best	Rel. Err%	Iter
Br17	39	39.17	0.43	17	39	0	100	39	0	100	39	0	100
Ry48p	14422	14440	0.12	186	14939	3.65	200	15254	5.77	100	14857	3.02	100
Ftv33	1286	1287	0.08	79	1482	15.24	200	1372	6.69	600	1409	9.56	200
Ftv35	1473	1474	0.07	107	1491	1.22	100	1508	2.38	300	1489	1.09	200
Ftv38	1530	1530	0	126	1634	6.79	200	1547	1.11	700	1546	0.98	100
Ftv44	1613	1614	0.06	58	1733	7.44	3400	1673	3.72	300	1699	5.33	100
Ftv47	1776	1777	0.06	6	1793	0.96	700	1787	0.56	1600	1846	14.45	300
Ftv55	1608	1610	0.12	117	1781	10.76	3400	1747	8.64	200	1657	3.05	100
Ftv64	1839	1839	0	10	2054	11.69	500	1890	2.77	500	1871	1.74	900
Ftv70	1950	1955	0.26	46	2168	10.9	400	2074	6.36	100	2004	2.77	600
Kro124p	36230	36275	0.12	13	40524	11.85	200	41121	13.5	500	40106	10.70	100
Ftv53	6905	6905	0	126	8088	17.13	300	7383	6.92	500	7847	13.64	100
Ftv70	38673	38753	0.21	179	40566	4.89	500	39065	1.01	1900	39197	1.35	100
		1.53%			102.52%			59.43%			67.68%		

African buffalo optimization with Lin-Kernighan Algorithm

To conclude our validation process of the African Buffalo Optimization algorithm, it is necessary to compare its performance with that of the famous Lin-Kernighan method developed by Helsgaun. It is imperative to compare the ABO with Lin-Kernighan since the latter, aside from being the most popular of all Lin-Kernighan approaches, it is about the fastest algorithm in solving asymmetric Travelling Salesman’s Problems. This comparison deals with the examination of the success rate of both approaches and the time taken to arrive at solutions. The success rate is obtained by calculating the percentage successful run of the algorithm in question to get the optimal or near-optimal result of the particular TSP instance. The result is presented in Table 4 below.

The good runs of the ABO continues when compared with Lin-Kernighan algorithm (See Table 4). In terms of time used to obtain their results, both algorithms performed extremely well with the Lin-Kernighan performing a little better than the ABO. It took the ABO a cumulative time of 0.3 second to solve all the problems to Lin-Kernighan’s cumulative total of 0.0 second. Moreover, experimental data obtained shows that the ABO obtained 100% success in reaching optimal or near-optimal solutions to all the test cases. The same cannot be said of the Lin-Kernighan which obtained a disappointing 21% success in P43, 47% in Ftv35, 53% in Ftv38, 95%

in Kro124p and 99% in Ry48p. This, again, underscores ABO's superior performance when in competition with many popular methods.

Table 4. ABO and Lin-Kernighan algorithm on ATSP

TSP Instances	No of Cities	Optimal Values	ABO		LIN- KERNIGHAN	
			Success Rate %	Time (secs)	Success Rate %	Time (secs)
Br17	17	39	100	0	100	0
Ry48p	48	14422	100	0	99	0
Ftv33	34	1286	100	0	100	0
Ftv35	36	1473	100	0	47	0
Ftv38	39	1530	100	0	53	0
Ftv44	45	1613	100	0	100	0
Ftv47	48	1776	100	0	100	0
Ft53	53	6905	100	0	100	0
Ftv55	56	1608	100	0	100	0
Ftv64	65	1839	100	0	100	0
Ftv70	71	5620	100	0.1	100	0
Kro124p	100	36230	100	0.1	95	0
Ft53	53	6905	100	0	100	0
Ft70	70	38673	100	0	100	0
P43	43	5620	100	0.1	21	0

CONCLUSION

This paper introduces the African Buffalo Optimization algorithm. It highlights the basic flow of the novel algorithm. The ABO algorithm benchmarked on some challenging global optimization test functions and over 24 Symmetric and 15 Asymmetric Travelling Salesman's Problems was validated by comparing its performance with other methods that use similar or different memory matrix and information propagation techniques in their search for solutions. The results obtained from using the ABO in all the comparative instances are quite competitive. In fact, in most cases, the ABO outperformed the other methods that use different techniques to construct their tours. It is worthy of note that the speed of the ABO in arriving at solutions is one remarkable contribution of the novel algorithm. It is therefore our conclusion that the ABO is a worthy addition to the Swarm Intelligence family of optimization algorithms.

In view of the competitive performance of the African Buffalo Optimization (ABO) to solving the benchmark test functions and TSP instances in this study, the authors hope to apply the ABO to solving other optimization problems like Job Scheduling, Knapsack Problem, Vehicle Routing and solving urban transport problems.

REFERENCES

- Ali, M. M., Khompatraporn, C., & Zabinsky, Z. B. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of global optimization*, 31(4), 635-672.
- Arora, J. (2004). *Introduction to optimum design*: Academic Press.
- Baritomba, B., & Hendrix, E. M. (2005). On the investigation of stochastic global optimization algorithms. *Journal of global optimization*, 31(4), 567-578.
- Ben-Israel, A. (1966). A Newton-Raphson method for the solution of systems of equations. *Journal of Mathematical analysis and applications*, 15(2), 243-252.
- Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1): Athena Scientific Belmont, MA.
- Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3), 268-308.
- Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*: Oxford university press.
- Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1), 58-73.
- Di Caro, G., & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 317-365.
- Eigen, M. (1973). *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*: mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt.
- Farmer, J. D., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1), 187-204.
- Gandomi, A. H., Yang, X.-S., & Alavi, A. H. (2013). Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1), 17-35.
- Ghanem, R. G., & Spanos, P. D. (2003). *Stochastic finite elements: a spectral approach*: Courier Corporation.
- Haftka, R. T., & Gürdal, Z. (2012). *Elements of structural optimization* (Vol. 11): Springer Science & Business Media.
- Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1), 106-130.
- Hurwitz, C., & Craig, R. GNU tsp solve version 1.4. *Computer software*, http://www.or.deis.unibo.it/research_pages/tspsoft.html.
- Karaboga, D., & Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1-4), 61-85.
- Karp, R. M., & Steele, J. M. (1985). Probabilistic analysis of heuristics. *The traveling salesman problem*, 181-205.
- Kennedy, J. (2010). Particle swarm optimization *Encyclopedia of Machine Learning* (pp. 760-766): Springer.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598), 671-680.

- Kumbharana, N., & Pandey, G. M. (2013). A Comparative Study of ACO, GA and SA for Solving Travelling Salesman Problem. *International Journal of Societal Applications of Computer Science*, 2(2), 224-228.
- Luenberger, D. G. (1973). *Introduction to linear and nonlinear programming* (Vol. 28): Addison-Wesley Reading, MA.
- Mahdavi, M., Fesanghary, M., & Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied mathematics and computation*, 188(2), 1567-1579.
- Marinakakis, Y., Marinaki, M., & Dounias, G. (2011). Honey bees mating optimization algorithm for the Euclidean traveling salesman problem. *Information Sciences*, 181(20), 4684-4698.
- Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69, 46-61.
- Olafsson, S. (2006). Metaheuristics. *Handbooks in operations research and management science*, 13, 633-654.
- Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2011). *The Bees Algorithm—A Novel Tool for Complex Optimisation*. Paper presented at the Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006.
- Prins, H. (1996). *Ecology and behaviour of the African buffalo: social inequality and decision making* (Vol. 1): Springer Science & Business Media.
- Rani, D., Jain, S. K., Srivastava, D. K., & Perumal, M. (2012). 3 Genetic Algorithms and Their Applications to Water Resources Systems. *Metaheuristics in Water, Geotechnical and Transport Engineering*, 43.
- Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4), 376-384.
- Rocha, A. M. A., Fernandes, E. M., & Soares, J. L. C. (2004). Solution of asymmetric traveling salesman problems combining the volume and simplex algorithms.
- Said, Y., & Wegman, E. (2009). Roadmap for optimization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1), 3-17.
- Schaffer, J. D. (1985). *Multiple objective optimization with vector evaluated genetic algorithms*. Paper presented at the Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985.
- Stueckle, S., & Zinner, D. (2008). To follow or not to follow: decision making and leadership during the morning departure in chacma baboons. *Animal Behaviour*, 75(6), 1995-2004.
- Suchal, J., & Návrát, P. (2010). Full text search engine as scalable k-nearest neighbor recommendation system *Artificial Intelligence in Theory and Practice III* (pp. 165-173): Springer.
- Van der Cruyssen, P., & Rijckaert, M. (1978). Heuristic for the asymmetric travelling salesman problem. *Journal of the Operational Research Society*, 697-701.
- Vanderplaats, G. N. (2007). *Multidiscipline design optimization*: Vanderplaats Research & Development, Incorporated.
- Venter, G. (2010). Review of optimization techniques. *Encyclopedia of aerospace engineering*.
- Wilson, D. S. (1997). Altruism and organism: Disentangling the themes of multilevel selection theory. *The American Naturalist*, 150(S1), s122-S134.
- Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1), 67-82.

- Yang, X.-S. (2011). Review of meta-heuristics and generalised evolutionary walk algorithm. *International Journal of Bio-Inspired Computation*, 3(2), 77-84.
- Zang, H., Zhang, S., & Hapeshi, K. (2010). A review of nature-inspired algorithms. *Journal of Bionic Engineering*, 7, S232-S237.
- Zheng, R. T., Ngo, N., Shum, P., Tjin, S., & Binh, L. (2005). A staged continuous tabu search algorithm for the global optimization and its applications to the design of fiber bragg gratings. *Computational Optimization and Applications*, 30(3), 319-335.