Experimental Exploration of Evolutionary Algorithms and their Applications in Complex Problems: Genetic Algorithm and Particle Swarm Optimization Algorithm

Article *in* British Journal of Healthcare Assistants · April 2023

DOI: 10.14738/jbemi.102.14427

3 authors, including:

Abdul Joseph Fofanah
Milton Margai Technical University
16 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Saidu Koroma
Milton Margai College of Education and Technology
5 PUBLICATIONS **1** CITATION
SEE PROFILE

British Journal of Healthcare and Medical Research - Vol. 10, No. 2
Publication Date: April 25, 2023
DOI:10.14738/jbemi.102.14427.
Fofanah, A. J., Koroma, S., & Bangura, H. I. (2023). Experimental Exploration of Evolutionary Algorithms and their Applications in Complex Problems: Genetic Algorithm and Particle Swarm Optimization Algorithm. British Journal of Healthcare and Medical Research, Vol - 10(2). 364-401.

SERVICES FOR SCIENCE AND EDUCATION

Experimental Exploration of Evolutionary Algorithms and their Applications in Complex Problems: Genetic Algorithm and Particle Swarm Optimization Algorithm

Abdul Joseph Fofanah
Mathematics and Computer Science,
Milton Margai Technical University, F
reetown, Western Area, Sierra Leone

Saidu Koroma
Mathematics and Computer Science,
Milton Margai Technical University,
Freetown, Western Area, Sierra Leone

Hamza Ibrahim Bangura
Physics and Computer Science,
School of Technology, Njala University,
Freetown, Sierra Leone

ABSTRACT

The primary aim of this study was to experiment and compare the empirical evidence of evolutionary algorithms and their applications using the Genetic Algorithm (GA) and Particle Swamp Optimization (PSO) algorithm. The objectives of the study included reviewing recent scholarly scientific research papers and practices, and algorithms with existing genetic algorithm techniques, identifying the need for lessons learned and leveraging best health practices garnered from the existing problems of optimization, reviewing and identifying the technological tools used in the healthcare domain, algorithms, and the effectiveness of GA and PSO techniques and to develop and analyse GA algorithms using datasets to predict and determine which of the two algorithms performs better. Evolutionary algorithms which are normally viewed as components of both evolutionary computing and bio-inspired computing use mechanisms inspired by nature and are indeed capable of solving problems through processes that emulate the behaviours of living organisms. Genetic algorithms (GAs), evolution strategies (ESs), differential evolution (DE), and particle swarm optimization (PSO) are typical examples. Therefore, evolutionary algorithms are typically used to provide good, estimated solutions to problems that cannot be solved easily using other techniques, and as such numerous optimization problems fall into this category. This work aims to provide a systematic evaluation of the literature on contemporary PSO methods for knowledge discovery in the area of illness detection. Although the major objective is to present recommendations for future enhancement and development in this field, the systematic analysis reveals the possible study topics of PSO techniques as well as the research gaps. The essential ideas, theoretical underpinnings, and traditional application domains are covered in this study. It is fervently anticipated

that the findings will help researchers to thoroughly examine PSO algorithms for disease diagnosis. According to the current state of PSO strategies in healthcare, several difficulties that can be taken on to advance the sector are mentioned.

Keywords: Genetic Algorithm, Particle Swarm Optimization, Healthcare System, Evolutionary Algorithm, Crossover, Mutation, and Genetic Operators

INTRODUCTION

Differential Evolution Algorithm

The stochastic population-based search method, known as Differential Evolution (DE), was proposed by Storn and Price [1]. This search method reveals excellent competence in solving a wide variety of optimization problems with different attributes from numerous fields and many real-world application problems [2]. Comparable to all other evolutionary algorithms (EAs), contemporary studies suggest that the evolutionary process of DE use's mutations, crossover, and selection operators at each generation to reach the worldwide optimum.

Moreover, the algorithms most efficacious in use at present are the evolutionary algorithms (EAs). In terms of Differential Evolution (DE), the target vector describes everyone in the population. The mutation is used to create a distorted vector, which disturbs a target vector using the difference vector of other entities in the population. Subsequently, the crossover operation produces the trial vector by joining the parameters of the mutation vector with the parameters of a parent vector derived from the populace. As a final point, the fitness value and selection operation determine which of the vectors should be chosen for the following creation by executing a one-to-one completion between the created trail vectors and the equivalent parent vectors [3]. Succinctly put, further studies state that; the presentation of DE essentially hinges on the mutation method; and the crossover operator. Besides, the inherent control parameters (population size NP, scaling factor F, and the crossover rate CR) play a significant role in balancing the multiplicity of the population; and coming together with the speed of the algorithm. The study further points out that, the advantages are simplicity of implementation, reliability, speed, and robustness [2].

Consequently, the study on DE has been extensively applied in resolving numerous global applications of science and engineering, such as {0–1} Knapsack Problem [4], financial markets dynamic modelling [5], feature selection [6], admission capacity planning in higher education [7], and El-Quality SA and Mohamed AW [7], and solar energy [8], for more applications, interested readers can refer to [10]. Nevertheless, DE has numerous flaws, as all other evolutionary search methods do concern the NFL (no free lunch) theorem. In general, DE has a good worldwide exploration ability that can reach the region of global optimum, but it is sluggish at the exploitation of the solution [11].

Furthermore, it is noted that; the parameters of DE are problem-dependent, and it is difficult to adjust them for different problems. Additionally, the study conducted by Das et al. [12] pointed out that; DE performance decreases as the search space dimensionality increases. Lastly, the performance of DE declines significantly when the difficulties of premature convergence and/or stagnation happen [13-14]. Subsequently, numerous researchers have suggested numerous methods to improve the rudimentary DE. The literature used by [14], [15] suggested amendments, improvements; and developments on DE which are primarily focused on

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

adjusting control parameters in an adaptive or self-adaptive manner while there are a few attempts in developing new mutations rule. Two evolutionary process components are proposed in their work; to overcome these disadvantages of differential evolution. One or both can be combined with the DE family of algorithms to enhance their search capabilities on the difficult and multifaceted optimization difficulties.

1. Firstly, a new directed mutation operator is introduced, so as to enhance global and local search capabilities and simultaneously increase the convergence speed of DE.
2. Secondly, to update the values of CR in each generation, a simple adaptive scheme without extra parameters is proposed. This guided the search process by the previous and current knowledge of their successful ratios that produced better trial vectors in the last generation.

Moreover, one thing that is important to note is that AGDE has been tested on 28 benchmark test functions developed for the 2013 IEEE Congress on Evolutionary Computation [16]. Extensive numerical experiments and comparisons indicate that the proposed (AGDE) algorithm is superior and competitive with four other contemporary evolutionary algorithms principally in the case of high-dimensional complex optimization problems with different characteristics.

Differential Evolution

The differential evolution section gives a brief summary of the fundamental differential evolution (de) algorithm. A study conducted on differential evolution which is commonly known as de/rand/1/bin shows that a preliminary arbitrary population, denoted by p, consists of np individuals [1] and fan hy, lampinen j [17]. Each individual is represented by the vector $x_i = (x_{1,i}, x_{2,i}, ..., x_{d,i})$, where d is the number of dimensions in solution space. As the population will be altered with the running of the evolutionary process, the generation periods in de are expressed by g = 0,1, ..., gmax, where gmax is the maximal times of generations. In the case of the ith individual of p at the g generation, it is signified by $x_i^G = (x_{1,i}^G, x_{2,i}^G, \dots, x_{D,i}^G)$. It is also explicitly clear that the lower and upper bounds in each dimension of the search space are recorded by $x_l = (x_{1,l}, x_{2,l}, ..., x_{d,l})$ and $x_u = (x_{1,u}, x_{2,u}, ..., x_{d,u})$ respectively. Nevertheless, the original population p^0 is generated at random according to a uniform distribution within the lower (x_l) and upper (x_u) boundaries. It is therefore seen that after initialization, these individuals are altered by differential evolution (de) operators (mutation and crossover) to produce a trial vector. Consequently, to choose the vector that should subsist to the following generation, a comparison between the parent and its trial vector is executed [18]. The differential evolution (de) phases are thus discussed below:

Initialization

It is seen that an initial population P^0 must be produced in order to establish a starting point for the optimization process. Normally, each jth component (j = 1, 2,....., D) of the ith individuals (i = 1, 2,....., NP) in the initial population, P^0 is obtained as follow:

$$\boldsymbol{x_{j,i}^0 = x_{j,i} + rand\,(0,1).\left(x_{j,u} - x_{j,l}\right)} \quad [1]$$

where rand (0,1) returns a uniformly distributed random number in [0, 1].

Mutation

The study shows that at generation G, for a single target vector x_i^G, a mutant vector x_i^G is generated according to the following:

$$\boldsymbol{x_I^G = x_{r1}^G F.\left(x_{r2}^G - x_{r3}^G\right). r_1 \neq r_2 \neq r_3 \neq i} \quad [2]$$

When the indices are randomly selected $r_1, r_2, r_3 \in \{1, 2, ..., NP\}$. F appears to be a real number and it controls the amplification of the difference vector (x_{r2}^G - x_{r3}^G). According to Storn and Price [1], the range of F is in [0, 2]. In this study, if a component of a mutant vector

violates the search space, then the value of this element is generated as a new value using (1). The other regularly used mutation strategies are:

"DE/best/1"[19]: $v_i^G = x_{best}^G + F\,.(x_{r1}^G - x_{r2}^G)$ [3]

"DE/best/2"[19]: $v_i^G = x_{best}^G + F\,.(x_{r1}^G - x_{r2}^G) + F.(x_{r3}^G - x_{r4}^G)$ [4]

"DE/rand/2"[19]: $v_i^G = x_{r1}^G + F.(x_{r2}^G - x_{r3}^G) + F.(x_{r4}^G - x_{r5}^G)$ [5]

"DE/current-to-best/1" [19]: $v_i^G = x_i^G + F.\left(x_{rbest}^G - x_i^G\right) + F.(x_{r1}^G - x_{r2}^G)$ [6]

"DE/current-to-rand/1" [19]: $v_i^G = x_i^G + F.\left(x_{r1}^G - x_i^G\right) + F.(x_{r2}^G - x_{r3}^G)$ [7]

It is seen that the indices r1, r2, r3, r4, and r5 are mutually different integers that are generated at random within the range [1, 2,...,NP], and are also dissimilar from the index i. These indices are generated at random once for each mutant vector. The F scale factor is a positive control parameter for scaling the difference vector. The x_{best}^G is the best individual vector that possesses the best fitness value in the population at generation G.

Crossover

The two primary crossover types are binomial and exponential. The binomial crossover type is the one mainly discussed. In discussing the binomial crossover, the target vector is blended with the changed vector, which produces the trial vector u_i^G, as shown in the following scheme:

$$\boldsymbol{u_{ji}^G = \begin{cases} v_{j,i}^G, & if\ (rand_{ji} \leq CR or j = j_{rand}) \\ x_{j,i}^G, & otherwise \end{cases}} \quad [8]$$

where randj, i (i ∈ [1, NP] and j ∈ [1, D]) is a uniformly dispersed arbitrary number in [0,1], CR ∈ [0, 1] called the crossover rate that controls the number of components that are inherited from the mutant vector. Also, jrand is a dispersed uniform random number in [1, D] that ensures that at least one component of the trial vector is inherited from the changed vector.

Selection

Differential Evolution acclimatizes a greedy selection strategy. It could be observed that if and only if the trial vector u_i^G produces a very good or a better fitness function value than x_i^G, then

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

u_i^G is fixed to x_i^{G+1}. Otherwise, the previous vector x_i^G is maintained. For a minimization problem, the selection scheme is shown as follows:

$$x_i^{G+1} = \begin{cases} u_{i,}^{G} & f(u_i^G) \leq f(x_i^G) \\ x_i^G, otherwise \end{cases} . \quad [9]$$

Novel Mutation Scheme

It is significant to note that DE/rand/1 is the elementary transformation method developed by Storn and Price [1], and it is recounted to be the most efficacious and extensively used design in the literature [12]. Evidently, in this approach, three vectors are randomly chosen from the population for mutation, and the base vector is then selected at random among the three. The other two vectors form the difference vector that is added to the base vector. Subsequently, the difference vector is able to maintain population diversity and global search capability with no bias toward any specific search direction. It however decelerates the convergence speed of DE algorithms [19]. Moreover, DE/rand/2 strategy is like the previous scheme that has extra two vectors that can create another difference vector, which might lead to improved trepidation than one-difference-vector-based strategies [19].

Additionally, DE/rand/2 strategy can deliver more numerous differential trial vectors than the DE/rand/1/bin strategy which escalates its investigation ability of the search space. On the contrary, avaricious strategies like DE/best/1, DE/best/2, and DE/current-to-best/1 subsume the information of the best solution found so far in the evolutionary process to increase the local search tendency that leads to a fast convergence speed of the algorithm. However, the multiplicity of the population and exploration competence of the algorithm may worsen or might be absolutely lost through a very minute number of generations, i.e., at the commencement of the optimization process, which causes glitches such as stagnation and/or premature convergence. Subsequently, in order to conquer the inadequacies of both types of mutation strategies, most of the recent successful algorithms utilize the strategy candidate pool that combines different trail vector generation strategies, that have diverse characteristics and distinct optimization capabilities, with dissimilar control parameter settings to be able to deal with a variety of problems with dissimilar features at different stages of evolution [19-20], [21]. Conversely, taking into account the feebleness of the current greedy strategies, (Zhang J, and Sanderson AC 2009) introduced a novel differential evolution (DE) algorithm, named JADE. This new differential evolution (DE) algorithm, is meant to develop optimization performance through implementing a new mutation strategy, called “DE/current-top best” with an optional external archive and updating control parameters in an adaptive manner.

Therefore, proposing new mutation strategies that can substantially develop the exploration capability of DE algorithms and increase the likelihood of achieving promising and successful results in multifaceted and large-scale optimization glitches is still an open challenge for evolutionary computation research. This research, therefore, uses a new mutation rule with a view to be balancing the global exploration ability and the local exploitation propensity that increases the convergence rate of the algorithm. The projected mutation approach for this investigation uses two randomly selected vectors of the top and bottom 100p% individuals in the modern population of size NP. However, the third vector is randomly chosen from the

middle [NP-2(100p%)] individuals. In view of the above, the proposed mutation vector is generated in the following manner:

$$x_i^{G+1} = x_r^G + F \,.\left(x_{p-best}^G - x_{p-worst}^G\right), \qquad [10]$$

where x_r^G is chosen as a random vector from the middle $NP - 2(100P\%)$ individuals, and x_{p-best}^G and $x_{p-worst}^G$ are chosen at random as one of the top and bottom 100p% individuals in the present population, respectively, with p ∈ (0%, 50%), F is the mutation factors that are independently generated according to a uniform distribution in (0.1,1).

Truly, it is seen that the primary concept of the projected new mutation is derived from the fact that each one of the vectors learns from the location of the top best and the bottom worst entities among the whole population of a particular generation. Evidently, it is generally observed from equation (1), that the amalgamation of the objective function value in the mutation scheme has two benefits. Firstly, the target vectors are not always attracted toward the same best position found so far by the entire populace. Therefore, the premature convergence at local optima can be virtually avoided by following the same direction of the top best vectors which conserves the investigation capability. Secondly, circumventing the route of the bottommost worst vectors can increase the exploitation propensity by guiding the search procedure to the hopeful regions of the search space, i.e., it distillates the exploitation of some sub-regions of the search space.

Therefore, it could be seen that the global solution can be easily reached if all vectors follow the same directions as the best vectors besides, they also follow the opposite directions of the worst vectors among the randomly selected vectors. Consequently, the directed perturbations in the proposed mutation resemble the concept of gradient as the difference vectors are directed from the worst vectors to the best vectors with different weights [23]. Hence, it is substantially used to travel through the landscape of the objective function being optimized in different sub-regions around the best vectors within the search space through an optimization process. As a result of the above, it is found that by using and distributing the best and worst information about the DE population, the projected directed mutation has balanced both the global exploration capability and local exploitation propensity.

PARAMETER ADAPTATION SCHEMES IN AGAPE

The efficacious execution of the DE algorithm is significantly reliant on the choice of its two control parameters: The scaling factor F and crossover rate CR (Price et al. [2], [14], Omran et al.[24]). The two parameters play an important role because they greatly influence the efficiency, effectiveness, and robustness of the algorithm. Moreover, it is problematic to determine the optimal values of the control parameters for a range of glitches with different attributes at dissimilar phases of evolution. Generally, the scaling factor F is a significant parameter that controls the evolving rate of the population i.e., it is closely related to the convergence speed [19]. The small F value encourages the exploitation propensity of the algorithm that makes the search focus on the neighbourhood of the current solutions; hence it can enhance the convergence speed. Nevertheless, it may as well lead to premature convergence [6]. In addition to this, a large F value increases the exploration capability of the algorithm, which can make the mutant vectors dispense extensively in the search space, and

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

can upsurge the multiplicity of the population [21]. Conversely, it may slow down the search [6] with respect to the scaling factors in the proposed algorithm, at each generation G. The scale factors F, of each individual target vector is independently generated according to a uniform distribution in (0.1,1) to enrich the search behaviour. The constant crossover (CR) of the system reflects the chance with which the trial individual inherits the actual individual's genes, i.e., which and how many components are mutated in each element of the current population ([20], [23]). Practically put, the constant crossover (CR) controls the multiplicity of the [21] To tell the truth, if constant crossover (CR) is high, it will increase the population diversity. Nonetheless, the steadiness of the algorithm may diminish. Alternatively, small values of CR increase the likelihood of stagnation which may weaken the exploration ability of the algorithm to open up the new search space. Furthermore, CR is usually more sensitive to problems with different characteristics such as unimodality and multimodality, separable and non-separable problems. For separable problems, CR from the range (0, 0.2) is the best, while for multi-modal, parameter-dependent problems Cr in the range (0.9, 1) is suitable [25]. Alternatively, there are a wide variety of approaches for adapting or self-adapting control parameter values through the optimization process. Most are of these methods based on generating random values from uniform, normal or Cauchy distributions or by generating different values from a pre-defined parameter candidate pool besides using the previous experience (of generating better solutions as offspring) to guide the adaptation of these parameters [18], [21]. On the other hand, other studies make use of full information on the distribution conditions of each vector-individual in successive populations dynamically through generations to generate suitable control parameters [18], [25].

The presented work is different completely from all other previous studies in the following major aspects:

- AGDE uses new adaptation schemes to update CR during generations without using extra parameters or determining specific learning periods.
- AGDE uses a pre-determined specific candidate pool for generating values for CR based on other researchers' studies. Thus, AGDE benefits from the previous experience and results of other research by focusing on choosing the most appropriate values during the evolution process from a narrow set of possible values of CR.

Therefore, taking into consideration all the above-mentioned guideline information about the major control parameters of DE, a novel adaptation scheme is used for CR, respectively.

Crossover Adaptation

At each generation G, the crossover probability CR of each individual target vector is independently generated according to one of the following two uniform distributions. (1) $CR_1 \in [0.05, 0.15]$; (2) $CR_2 \in [0.9, 1]$. [11]

In fact, the lower and upper limits of the ranges for these proposed sets are recommended by (Wang et al., 2011) to deal effectively with problems of different characteristics such as unimodality and multimodality, separable and non-separable problems. Obviously, at each generation G, these two groups are adaptively chosen based upon their experiences of producing optimistic solutions during the evolution process as follows:

$$If\ G\ =\ 1$$

$$CR_i^1 = \begin{cases} CR_1, \\ CR_{2,Otherwise} \end{cases} if\ u(0,1) \le 1/2$$

Else [12]

$$CR_i^1 = \begin{cases} CR1, \\ CR2, \end{cases} \ if\ \ if\ u(0,1) \le p_1$$

Denote $pj\ , j\ =\ 1, 2, \ldots, m$ as the probability of selecting the jth set, where m is the total number of sets and it is set to 2 $and\ the\ sum\ of\ pj\ is\ 1. pj$ is then initialized as$1/j, i.e. 1/2$. The roulette wheel selection method is used to select the appropriate settings for each target vector based on the probability $pj\ . pj$ is continuously updated during generations in the following manner:

$$P_j^{G+1} = ((G - -1)XP_j^s + ps_j^{G-1})/G \quad [13]$$

$$ps_j^G = \frac{s_j^G}{\sum_{j=1}^m s_j^G} \quad [14]$$

Where

$$s_j^G = \frac{ns_j^G}{\sum_{G=1}^G ns_j^G + \sum_{G=1}^G nf_j^G} + \in, \quad [15]$$

where G is the generation counter; $ns_j^G and\ nf_j^G$ are the respective numbers of the offspring vectors generated by the jth set that survive or fail in the selection operation during the last G generations.

s_j^G is the success ratio of the trial vector generated by the jth set and that successfully enters the next generation;

ps_j^G is the probability of selecting the jth set in the current generation. $\in$ is a small constant value to avoid the possible null success ratios. $\in\ is\ set\ to$ 0.01, and in the experiments $s_j^G is$ prevented from becoming zero. It is very much explicit from Eq. (14) that the probability p_j is the weighted mean of both the previous and current success ratios to consider the complete history of the experience of a specific set during generations. Additionally, in the case of null success ratios of both sets, the p_j is reinitialized as $1/2$ to overcome the likelihood of stagnation. The core idea of the fitness-based adaptation scheme for the crossover rate is based on the following two fundamental principles. In the first case, which is the initial stage of the search process, the difference among individual vectors is large because the vectors in the population are completely dispersed or the population diversity is largely due to the random distribution of the individuals in the search space that requires a relatively smaller crossover value. Then, as the population evolves through generations, the multiplicity of the population decrease as the vectors in the population is clustered because each individual gets closer to the best vector found so far.

Subsequently, in this stage, in order to advance diversity and increase the convergence speed, the value of CR must be large value. Secondly, as previously mentioned, these two sets are

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

designed as recommended in the studies by [25], [21], to deal effectively with problems having different characteristics such as unimodality and multimodality, separable and non-separable problems.

In general, it is viewed that the adaptive control parameters which have dissimilar values during the optimization process in the succeeding generations increase the algorithm with controlled randomness. This controlled randomness, however, enriches the worldwide optimization presentation of the algorithm in terms of investigation and exploitation competencies. It can therefore be concluded that the projected adaptation schemes for the adaptive change of the values of the crossover rate can exceptionally benefit from the previous and current success ratios during the evolution process. These proposed adaptation schemes can in turn considerably balance the common trade-off between population multiplicity and convergence speed.

Concept of GA

Considering other algorithms based on metaheuristic techniques, the Genetic Algorithm (GA) is a metaheuristic technique inspired by the laws of Darwinian in genetics. This ensures finding valuable solutions to complex problems. Some random solutions referred to in this context as individuals are generated with each containing many features. These features are called chromosomes in this technique. Inspired by the law of genetics, crossover and mutations occur in chromosomes (features) to produce a second generation of individuals with more diverse features.

The two most central techniques for diversifying individuals are crossover and mutation. During crossover, two chromosomes are selected. Then a crossover point along each chromosome is selected followed by the interchange of the two chromosomes (features). A typical scenario of the phenomena is illustrated in Figure 1. The technique to persuade diversity in the population of individuals is called candidate solutions. Diagram (a) of figure 1, indicates during the crossover, that one part of a chromosome (feature) is interchanged by another fragment of another chromosome, and diagram (b) shows that during mutations, one or more datasets on a chromosome are converted to various ones. These modifications or changes will generate new individuals whose fittest or more optimal solutions will survive.

A genetic algorithm is a search-based algorithm used for solving optimization problems in machine learning. Genetic algorithm is important because it helps to solve difficult problems that would have taken a long time to solve. It has been used in various real-life applications such as data centres, electronic circuit design, code-breaking, image processing, and artificial creativity.

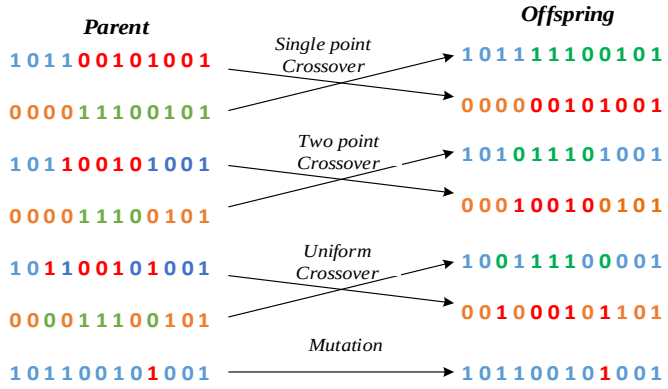

Figure 1: Crossover, one part of a chromosome is interchanged by another fragment of another chromosome, and mutations in one or more datasets on a chromosome are converted to various ones.

RESEARCH CONTEXT AND CHALLENGES

Previous research studies have examined the application of machine learning techniques for the prediction and classification of heart disease but were not too efficient. However, this study focuses on the impacts of specific machine learning techniques and not on the optimization of these techniques using optimized methods. In addition, few researchers attempt to use hybrid optimization methods for an optimized classification of machine learning. The most proposed studies in the literature exploit optimized techniques such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) with a specific ML technique such as SVM, KNN, or Random Forest. In this work, the Fast Correlation-Based Feature Selection (FCBF) method is applied as a first step (pre-treatment). When all continuous attributes are discretised, the attribute selection attributes relevant to mining, from among all the original attributes, are selected. Feature selection, as a pre-processing step to machine learning, is effective in reducing dimensionality, eliminating irrelevant data, increasing learning accuracy, and improving understanding of results. In the second step, PSO and ACO are applied to select the relevant characteristics of the data set. The best subset of characteristics selected by the characteristic selection methods improves the accuracy of the classification. Therefore, the third step applies classification methods to diagnose heart disease and measures the classification accuracy to evaluate the performance of characteristic selection methods. The main objective of this piece of work is the prediction of heart disease using different classification algorithms such as K-Nearest Neighbour, Support Vector Machine, Naïve Bayes, Random Forest, and a Multilayer Perception, Artificial Neural Network optimized by Particle Swarm Optimization (PSO) combined with Ant Colony Optimization (ACO) approaches. The MATLAB data-mining tool is used to analyse data from heart disease. The main contributions of this research are:

- Extraction of classified accuracy useful for heart disease prediction
- Remove redundant and irrelevant features with the Fast Correlation-Based Feature selection (FCBF) method.
- Optimizations with Particle Swarm Optimization (PSO) then we consider the result of PSO as the initial values of Ant Colony Optimization (ACO) approaches.
- Comparison of different data mining algorithms on the heart disease dataset.
- Identification of the best performance-based algorithm for heart disease prediction.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

One significant challenge of using GA is how to accelerate the evolutionary process dynamically in a huge search space. Existing methods usually require large-scale computing resources (e.g., a hundred GUPs). However, the study of a well-organized GA for automatically building a concise CNN-based model for image denoising is still lacking.

Therefore, proposing new mutation strategies that can substantially develop the exploration capability of DE algorithms and increase the likelihood of achieving promising and successful results in multifaceted and large-scale optimization glitches is still an open challenge for evolutionary computation research.

Optimization Problem of GA

In general, genetic algorithm resolves optimization glitches by mimicking the principles of biological evolution, repetitively changing a population of individual points using rules that are modelled on gene combinations in biological reproduction. In genetic algorithm, a population of possible solutions to an optimization problem (referred to as animals, people, organisms, or phenotypes) progresses toward greater solutions. Conventionally, explanations are represented in binary as strings of 0s and 1s, although other encodings are also always plausible. Each of the candidate solutions has a set of properties (its chromosomes or genotype) that can be altered and improved. The term generation is used to describe the population in each iteration of the evolution, which typically begins with a population of individuals that are generated at random. The fitness of every member of the population is assessed once in every generation. The fitness of a generation is normally the value of the objective function in the optimization problem being addressed. A new generation is created by stochastically selecting the fittest people from the current population, recombining their genomes, and perhaps introducing random mutations. The following algorithm iteration uses the fresh generation of candidate solutions. The algorithm typically comes to an end when the population has reached a desirable level of fitness, or the maximum number of generations has been produced.

An ordinary genetic algorithm needs:

- A fitness function to assess the solution domain and a genetic representation of the problem domain.
- A potential answer is typically represented as an array of bits. The use of arrays of various types and structures is fundamentally the same. The fundamental benefit of these genetic representations is that because of their fixed size, their pieces are simple to align and allow for straightforward crossover procedures. It is also possible to employ representations of variable length, but crossover implementation is more difficult in this situation. In gene expression programming, a combination of both linear chromosomes and trees is examined. Genetic programming explores representations in the form of trees, while evolutionary programming explores representations in the form of graphs.
- A GA initiates a population of solutions after defining the genetic representation and the fitness function, and then improves it by repeatedly using the mutation, crossover, inversion, and selection operations.

Initialization

Depending on the nature of the issue, the population can range from a few hundred to thousands of potential solutions. The beginning population is frequently produced at random,

allowing for all conceivable resolutions. On occasion, the answers may be "seeded" in regions where the best answers are most likely to be found.

Selection

A fraction of the current population is chosen to breed a new generation throughout each succeeding generation. A fitness-based approach is used to choose individual solutions, with fitter solutions (as determined by a fitness function) often having a higher chance of being chosen. Some methods of selection evaluate each solution's fitness and give preference to the best ones. Other techniques merely rate a representative sample of the population because the initial procedure could take a long period.

The fitness function, which is defined over the genetic representation, assesses the effectiveness of the solution that is represented. The fitness function always depends on the problem. In the knapsack issue, for instance, the goal is to maximize the overall worth of the items that can fit within a rucksack with a specific amount of capacity. An array of bits could be used to represent a solution, with each bit standing for a separate object and its value (0 or 1) indicating whether or not the object is in the knapsack. Such representations are not always accurate since items' sizes sometimes surpass the knapsack's storage capacity. If the representation is, then the fitness of the solution is the total value of everything in the knapsack. If the representation is correct, the fitness of the solution equals the total value of all the objects in the knapsack, else it is 0.

Some problems make it difficult or even impossible to define the fitness expression; in these situations, a simulation or even interactive genetic algorithms may be used to estimate the fitness function value of a phenotype (for example, computational fluid dynamics is used to estimate the air resistance of a vehicle whose shape is encoded as the phenotype). By combining the genetic operator’s crossover (also known as recombination) and mutation, the next step is to produce a second-generation population of solutions from the ones that were initially chosen.

Genetic Operators

A pair of "parent" solutions are chosen from the pool that was previously chosen to breed in order to produce each new solution. By engaging the aforementioned crossover and mutation techniques to construct a "child" solution, a new solution is created that often shares many traits with its "parents." For every new child, new parents are chosen, and this process is repeated until a fresh population of solutions is produced that is of the correct size. Some study reveals that more than two "parents" produce superior quality chromosomes, even though reproduction techniques based on the utilization of two parents are more "biology inspired."

These activities ultimately lead to a population of chromosomes in the subsequent generation that differs from the first generation. Since only the best creatures from the first generation are chosen for breeding, along with a small percentage of less fit solutions, the population's average fitness should have grown as a result of this operation. These less effective approaches guarantee genetic variation within the parental genetic pool and, consequently, guarantee the genetic diversity of the following generation of children.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

In order to discover appropriate settings for the issue class under consideration, tweaking variables like the mutation probability, crossover probability, and population size is worthwhile. Genetic drift could result from a relatively low mutation rate. The genetic algorithm may not fully converge if the recombination rate is too high. If elitist selection is not used, a high mutation rate could result in the loss of good solutions. A appropriate population size ensures enough genetic variety for the issue at hand, but if set to a value greater than necessary, it can result in a waste of computational resources.

Heuristics

Other heuristics may be used in addition to the major operators mentioned above to speed up or strengthen the calculation. The speciation heuristic discourages population homogeneity and delays convergence to a less ideal solution by penalizing crossover between candidate solutions that are too similar.

Termination

Up until a termination condition is met, this generational process is repeated. Typical termination circumstances include:

- Finding a solution that meets the minimal requirements
- Reached a fixed number of generations
- The time and money allotted for computation have been used.
- The fitness of the top-ranked solution is nearing or has reached a point where more iterations don't improve the situation.
- Manual examination
- Combination of the aforementioned

The Theory of the Building Blocks

Although it is easy to create genetic algorithms, it is, however, challenging to comprehend their behavior. It is particularly challenging to comprehend why, when used to solve real-world issues, these algorithms frequently produce answers that are highly fit.

An explanation of a heuristic that carries out adaptation by locating and redistributing "building blocks," or low order, short defining-length schemata with above average fitness.

The idea is that a genetic algorithm conducts adaptation by effectively and implicitly using this heuristic.

The heuristic is described as follows by Goldberg:

- "To create strings with possibly higher fitness, short, low order, and highly fit schemata are sampled, concatenated [crossed over], and resampled. In a sense, the complexity of our problem has been decreased by using these specific schemata [the building blocks]; rather than creating high-performance strings by testing every possible combination, we create better and better strings using the best partial answers from earlier samplings." We have already given these highly suited schemata a distinctive name: building blocks because they are so crucial to the operation of genetic algorithms and have low defining lengths and low order. A genetic algorithm seeks near-optimal

performance by the juxtaposition of short, low-order, high-performance schemata, or building blocks, just as a toddler builds amazing fortresses out of basic wooden blocks.

- Despite there being no agreement on whether the building-block hypothesis is valid, it has continually been assessed and utilized as a reference over time. For instance, numerous estimations of distribution procedures have been put out in an effort to create a setting where the hypothesis would hold true. Although promising findings for specific classes of problems have been published, doubts about the generality and/or applicability of the building-block hypothesis as a justification for GA effectiveness persist. In fact, a fair lot of research has been done to try to understand its limitations from the standpoint of distribution algorithm estimation.

Limitations of GA

When compared to other optimization algorithms, using a genetic algorithm has some drawbacks:

- Artificial evolutionary algorithms frequently have a segment that is the most prohibitive and constrained: repeated fitness function evaluation for complex tasks. Complex high-dimensional, multimodal problems frequently call for highly pricey fitness function analyses in order to find the best solution. In practical issues like structural optimization issues, evaluating a single function may take many hours to several days of exhaustive simulation. Such problems cannot be solved by standard optimization techniques. It could be required in this situation to forgo an exact evaluation in favor of a computationally effective approximate fitness measurement. It is clear that combining approximation models may be one of the most promising methods for successfully applying GA to difficult real-world issues.
- The complexity of genetic algorithms does not scale well. That is, the size of the search space frequently grows exponentially in regions where the number of elements subject to mutation is high. Because of this, applying the technique to issues like creating an engine, a house, or a plane is incredibly challenging. Such issues must be reduced to the most basic representation in order to be tractable by evolutionary search. As a result, we frequently see evolutionary algorithms encoding designs for fan blades rather than engines, building shapes rather than precise construction blueprints, and airfoils rather than complete aircraft designs. The second complexity difficulty is how to prevent components that have evolved to be good solutions from going through more damaging mutations, especially when their fitness assessment calls for them to work well along with other parts.
- The "better" option is simply superior in light of other options. Because of this, the stop condition is not always obvious.
- GAs frequently converges towards local optimums or even arbitrary spots rather than the problem's overall optimum. This indicates that it lacks the "know-how" to forgo immediate fitness in favour of long-term fitness. The chance of this depends on the fitness landscape's shape; some difficulties might make it simple to reach a global optimum, while others might make it simpler for the function to locate local optimum points. Although the No Free Lunch theorem establishes that there is no universal solution to this problem, it can be solved by adopting a different fitness function, speeding up a mutation, or utilizing selection procedures that preserve a diversified population of solutions.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

- The use of a "niche penalty," which reduces the representation of any group of individuals with a sufficiently high degree of similarity (niche radius) in future generations while preserving other (less similar) individuals, is a typical strategy for maintaining diversity. Nevertheless, dependent on the nature of the issue, this tactic might not work. When the majority of the population is too similar to one another, another strategy is to simply replace a portion of the population with randomly produced individuals. Genetic algorithms (and genetic programming) require diversity since a homogeneous population does not produce novel solutions when it is crossed. Diversity is not necessary for evolutionary programming or methods because mutation is more frequently used.
- It is challenging to operate on dynamic data sets since genes start to converge early on toward solutions that might not hold true for later data. It has been suggested that this can be fixed by increasing genetic diversity and preventing early convergence. These methods include either increasing the probability of mutation when the quality of the solution declines (a process known as triggered hypermutation) or sporadically introducing completely new, randomly generated elements into the gene pool. With the so-called "comma strategy," in which new parents are solely chosen from offspring and existing parents are not preserved, evolutionary programming and tactics can once more be put into practice. On problems with motion, this might work better.
- Decision issues are examples of problems that general algorithms (GAs) cannot effectively address since there is no way to converge on the solution. In certain situations, a random search could be just as quick to uncover a solution. However, the ratio of successes to failures is a sufficient fitness metric if the circumstance permits the success/failure trial to be repeated with (potentially) different results.
- Other optimization methods may be more effective than genetic algorithms in terms of speed of convergence for particular optimization issues and problem cases. Alternative and supplementary algorithms include methods based on integer linear programming, ant colony optimization, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g., ant colony optimization, particle swarm optimization). The degree of problem understanding affects whether genetic algorithms are appropriate; well-known problems frequently have superior, more specialized techniques.

Optimization Problem of PSO

Particle swarm optimization (PSO) is indeed a computer technique used in computational science that attempts to iteratively improve a candidate solution with respect to a specified quality metric. By using a population of potential solutions, which are here referred to as particles, and moving them across the search space in accordance with a straightforward mathematical formula over the particle's position and velocity, it solves problems. In addition to being led toward the best-known positions in the search space, which are updated as other particles find better positions, each particle's movement is also impacted by its local best-known position. The swarm should move toward better answers as a result of this.

Kennedy, Eberhart, and Shi are the famous authors of PSO, which was initially created to simulate social behavior by stylizing the movement of creatures in a bird flock or fish school. The algorithm was made simpler, and optimization was seen to be taking place. The book by

Kennedy and Eberhart discusses a variety of PSO and swarm intelligence's philosophical facets. Poli conducts a detailed analysis of PSO applications. Recently, Bonyadi and Michalewicz presented a thorough overview of theoretical and experimental PSO research.

Particle Swarm Optimization is described as metaheuristic because it can search very huge spaces of potential solutions and makes little to no assumptions about the problem being optimized. Furthermore, unlike traditional optimization techniques like gradient descent and quasi-newton methods, PSO does not employ the gradient of the issue being improved, negating the need for the optimization problem to be differentiable. Metaheuristics like PSO, though, do not ensure that an ideal solution will ever be identified.

PSO Algorithm

A swarm of potential solutions is how the PSO algorithm's fundamental variation operates. A few straightforward equations are used to move these particles about in the search space. The positions of the particles in the swarm, as well as their own best-known positions within the search space, serve as a guide for their travels. These will eventually start to direct the swarm's motions once better sites are found. It is hoped, but not guaranteed, that repeating the procedure will lead to the eventual discovery of a workable solution.

Formally, the cost function that needs to be reduced is defined as$f$: RnR. The function accepts a candidate solution as an argument in the form of a vector of real numbers, and as an output, it returns a real number that represents the value of the candidate solution's objective function. There is no knowledge of f's gradient. Finding a solution, a where $f(a)$ $f(b)$ for all b in the search space-which would indicate that an is the global minimum-is the objective.

Let S represent the total number of swarm particles, with each particle having a position in the search space of $xi\ Rn$ and a velocity of $vi\ Rn$. Let **g** represent the best-known position of the entire swarm, and let pi represent the best-known position of particle i. The following is a simple PSO algorithm:

for each particle $\boldsymbol{i = 1, \dots, s\ do}$

Initialize the particle's position with a *uniformly distributed* random vector: $\boldsymbol{x_i} - \boldsymbol{U(b_{1o}, b_{up})}$

Initialize the particle's best position to its initial position: $\boldsymbol{p_i \leftarrow x_i}$

if $\boldsymbol{f(p_i) < f(g)\ then}$

update the swarm's best-known position: $\boldsymbol{g \leftarrow p_i}$

Initialize the particle's velocity: $\boldsymbol{v_i \sim U(-|b_{up} - b_{1o}|)}$

while a termination criterion is not met **do**:

for each particle $\boldsymbol{i = 1, \dots, s\ do}$

for each dimension $\boldsymbol{d = 1, \dots, n\ do}$

pick random numbers: $\boldsymbol{r_p, r_g \sim U(0, 1)}$

Update the particle's velocity: $\boldsymbol{v_{i,d} \leftarrow W v_{i,d} + \phi_p r_p (P_{i,d} - x_{i,d}) + \phi_g r_g (g_d - x_{i,d})}$

Update the particle's position: $\boldsymbol{x_i \leftarrow x_i + v_i}$

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

if $f(x_i) < f(p_i)$ ***then***

Update the particle's best-known position: $P_i \leftarrow x_i$

if $f(P_i) < f(g)$ ***then***

Update the swarm's best-known position: $g \leftarrow P_i$

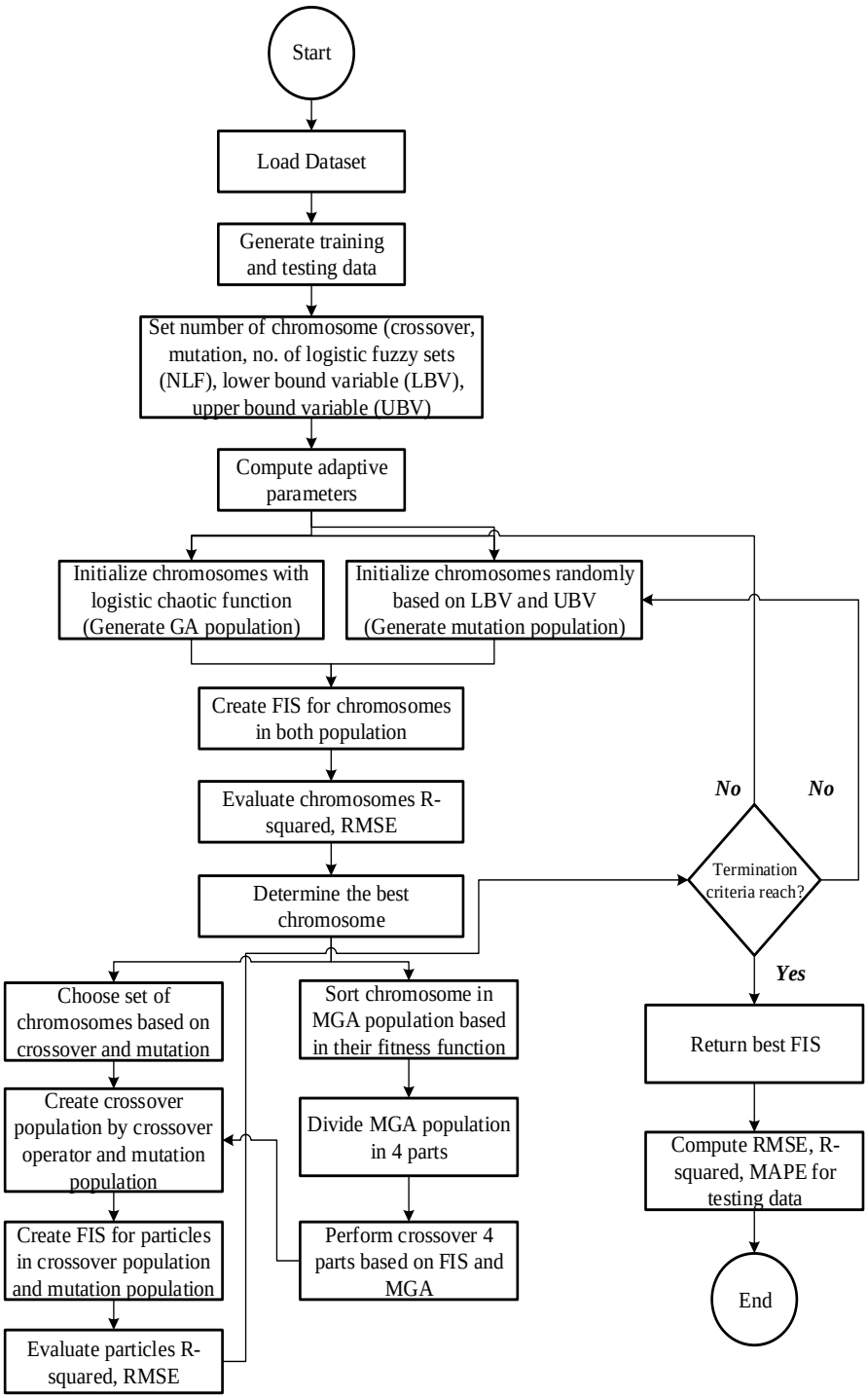

Figure 2: GA based on ANFIS

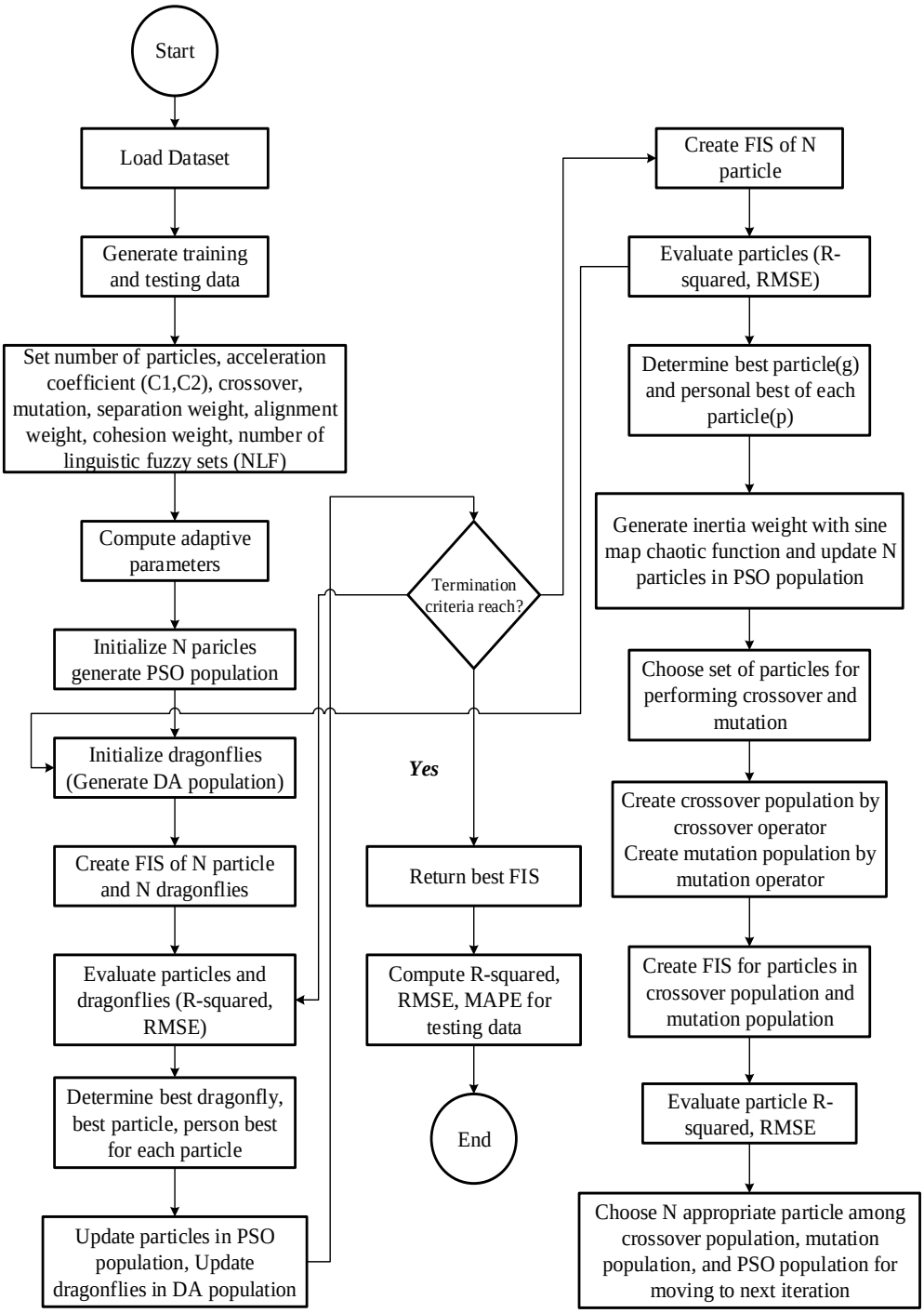

Figure 3: PSO based on ANFIS

REVIEW OF RELATED LITERATURE

Genetic Algorithm

The study conducted by Holland J. [27], states that Genetic Algorithms (GAs) belong to evolutionary algorithms and are inspired by natural biological evolution. Typically, a GA is made up of a "population" P of N "individuals", and has operations including initialization, individual selection, parent's crossover, and children mutation. A sequence of operations is referred to as an evolutionary "generation". It is viewed that, the competition among

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

individuals is replicated by a fitness function that chooses the fittest people over the feebler ones. However, during the population progression process, all individuals enter an iterative competition, where a new population is evolved in each generation, consisting of the survivors, children generated from the crossover of survivors, and mutated individuals from the children.

The Genetic Algorithm is among the experimental algorithms for combinatorial optimization [28]. The other two similar algorithms are simulated annealing (SA) and Tabu search (TS). From one perspective, all of them can be applied to numerous combinatorial optimization problems. On the other hand, they also have different properties. Firstly, GA requires the highest computational cost to find the best solution. Secondly, the quality of the best solution obtained by GA is better than simulated annealing and comparable to Youssef et al. [28]. More significantly, GA can integrate domain-specific knowledge in all optimization or combinatorial stages to dictate the search strategy, but SA and TS lack such a feature. Neural network combination (NNC) requires incorporating many constraints (domain knowledge) that are critical to a search strategy. Therefore, we choose GA over other heuristic algorithms for neural network evolution.

According to earlier studies conducted, it is noted that Genetic Algorithm has been extensively utilized as a heuristic search [29], an optimization method by Horn et al. [30], and also has been applied in machine learning methods by Booker and Holland [31], function optimization Goldberg et al. [32], and feature selection Yang and Honavar [33]. Recently, [34] applied GA to explore CNN architectures automatically for image classification. These approaches focus on exploring the structural modules and connections among layers. They first explore the representations of the networks with a binary coding scheme based on Cartesian genetic programming (CGP) Miller and Turner [35] and then train the networks with back-propagation. However, the study of a well-organized GA for automatically building a concise CNN-based model for image denoising is still lacking.

GA in Healthcare System

In recent years, various research has been conducted to determine the effectiveness of GA applications in healthcare systems and it complexity of the algorithm. According to Fofanah A.J et al. [36], GA and ML algorithms can be combined to predict the production of medicine and heart diseases with and accuracy of 89 percent for 40 years and above patients' threshold. Their approach provides the essential functionality of GA optimization method to solve complex problems, which results obtained is effective and dependent of higher training set. Secondly, to determine the intelligence computational science of the GA through algorithmic, method and better prediction approaches. Their architecture for the application provides some leverages and how effective the GA when combined with other approaches to solve complex problems. However, there exist some challenges in computational time especially using classical computers to perform the simulation exercises [37]. This provides the argument that as more AI applications become more effective the more issues related to complexity and optimization problems being diminished.

Neural Network Evolution

The studies conducted by Stanley and Miikkulainen [38]; Yao [39] in the early 1990s, endeavoured to combine evolutionary algorithms and artificial neural networks, and these eventually formed the branch of Evolutionary Artificial Neural Networks (EANN), which is also

part of Automated Machine Learning (AutoML). Numerous AutoML studies conducted by researchers such as He et al. [40]; Gijsbers et al. [41]; Liu et al. [42] have been suggested. Nevertheless, most of them focused on the traditional machine learning algorithms rather than deep CNN. Even though the studies conducted by Frank et al., Shi et al. [43] included CNN in their Bayesian Optimization-based AutoML framework. It however focuses on finding several good hyperparameter settings on fixed CNN architectures, and not a flexible and general solution. According to contemporary studies, there is more literature on EANN techniques. However, other studies fundamentally classified them as evolutionary algorithms (e.g., GA) based Xie and Yuille; Suganuma et al. [44] and reinforcement learning (RL) François-Lavet [45-46].

The majority of these methods focus on searching structural design combinations, such as the layer connections, the type of layers, and the way of connecting layers (e.g., summation, cascading). Nonetheless, the performance of a CNN is contingent on not only a structural module (e.g., residual block, interception) but also a combination of hyperparameters (e.g., loss functions, optimizers, activation functions). For example, Wang et al. [48] suggest a single image super-resolution (SISR) technique in order to accomplish state-of-the-art performance by exploring a suitable combination of various loss functions (e.g., MSE, perceptual loss Johnson et al. (2016), texture matching loss Liu et al. [49], adversarial training loss Goodfellow et al., (2014). As more and more optimization techniques are being suggested, a substantially large amount of choices and combinations of various hyperparameters are increasing the difficulty of manually designing an effective CNN quickly. Therefore, it is necessary and sufficient to create a neural network evolution framework that can automatically explore not only a suitable structure module but also the fittest hyperparameter combination.

Evaluation Method and Metrics

To exhaustively evaluate the performance of the proposed DEFSTH method, I conducted an evaluation using three predictive performance measures: accuracy (ACC), F1 score, and area under the ROC curve (AUC). The accuracy measures the ratio of correctly classified website instances regardless of their class, while the F1 score presents a harmonic mean of the precision and recall, averaged over all decision classes, whereas precision refers to the positive predictive value and recall refers to the true positive rate. Besides the aggregate measure of performance across all possible classification thresholds is presented by an AUC metric.

In order to objectively assess the presentation of the suggested DEFSTH technique and compare the performance against the conventional classification approaches, a gold standard conducted a 10-fold cross-validation (1995) technique. The 10-fold cross-validation devises a given dataset into train and test sets at a ratio of 90:10.

In a similar manner, the procedure is repeated 10 times, each time using a different test set for validation. The performance of Logistic Regression, Naive Bayes (NB), K-Nearest-Neighbours (KNN), Decision Tree (DT), and Multilayer Perceptron (MLP) classifiers was measured at the first experiment without any feature selection utilization or any kind of pre-processing. Moreover, while conducting the second experiment, measuring the performance of the suggested DEFSTH technique, the same classifiers were used as in the first experiment to adequately compare the obtained results. All performance classifiers were initialized with the Scikit-learn (2011) default settings. All the presented results in the following section are

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

averaged ACC, F1 score, and AUC values, obtained on the test websites instances over all runs for each of the 10 folds, if not specified otherwise.

Practically put, the performance of the canonical DE algorithm principally rests on the chosen mutation/crossover strategies and the associated control parameters. Moreover, due to DE limitations as previously aforementioned, during the past 15 years, many researchers have been working on the enhancement of DE. Consequently, many researchers have suggested innovative methods to overcome its glitches and enhance its performance [14]. A broad idea of these procedures is suggested in this piece of work. Firstly, many trial and-errors experiments have been conducted to regulate the control parameters. In their famous study, Storn and Price [2] proposed that NP (population size) between 5D and 10D, and 0.5 is a good initial value of F (mutation scaling factor). The operational value of F usually lies in a range between 0.4 and 1. The CR (crossover rate) is an initial good choice of CR=0.1; however, since a large CR often accelerates convergence, it is appropriate to first try CR as 0.9 or 1 in order to check if a quick solution is possible. In an earlier study, Gamperle et al. (2002) mentioned that a good choice for NP is between 3D and 8D, with F=0.6 and CR lies in [0.3, 0.9].

Nevertheless, Rokkonen et al. [25] further concluded that F = 0.9 is an excellent negotiation between convergence speed and convergence rate. Furthermore, CR is contingent on the nature of the problem, so CR with a value between 0.9 and 1 is suitable for non-separable and multimodal objective functions, while a value of CR is between 0 and 0.2 when the objective function is separable. On the other hand, owing to the apparent inconsistencies regarding the rules for determining the appropriate values of the control parameters from the literature, some methods have been designed with a view of adjusting control parameters in an adaptive or self-adaptive manner instead of manual tuning procedure. An early study by Qin [50], introduced an adaptive DE (ADE) algorithm which is based on the idea of controlling population diversity and implemented a multi-population approach. Liu and Lampinen [17], [42], in their early study, suggested a Fuzzy Adaptive Differential Evolution (FADE) algorithm. In such a case, Fuzzy logic controllers were used to regulating F and CR. The study further revealed that numerical experiments and comparisons on a set of famous benchmark functions indicated that the FADE Algorithm outstripped the rudimentary DE algorithm. Similarly, Zhang et al., [51] suggested a well-organized method, named jDE, for self-adapting control parameter settings.

The study further reveals that this method encodes the parameters into each individual and adapts them by means of evolution. The results showed that jDE is better than, or at least comparable to, the standard DE algorithm, (FADE) algorithm, and other state-of-the-art algorithms, especially when considering the quality of the solutions obtained. In a similar context, Omran et al. [24] suggested a self-adaptive differential evolution (SDE) algorithm. F was self-adapted using a mutation rule comparable to the mutation operator in the rudimentary DE. The experiments conducted also showed that SDE generally outstripped DE algorithms and other evolutionary algorithms. In yet another study, Qin et al. [50] presented a self-adaptive differential evolution (SaDE). The primary idea of SaDE is to instrument two mutation schemes such as “DE/rand/1/bin” and “DE/best/2/bin” concurrently, as well as adapt mutation and crossover parameters. The presentation of SaDE was calculated on a suite of 26 several yardstick problems and it was compared with the conventional DE and three adaptive DE variants. The experimental results revealed that SaDE yielded improved quality solutions and had a greater success rate.

In a similar context, stimulated by SaDE algorithm and motivated by the success of different self-adaptive DE approaches, Mallipeddi et al. [20] developed a self-adaptive DE, called EPSDE, based on the ensemble approach. In EPSDE, a pool of distinct mutation strategies along with a pool of values for each control parameter co-occurs during the course of the evolution technique and competes to produce offspring. The performance of EPSDE was calculated on a set of bound-constrained problems and was compared with conventional DE and other state-of-the-art parameter adaptive DE variants. The resulting comparisons showed that EPSDE algorithm outperformed conventional DE and other state-of-the-art parameter adaptive DE variants in terms of solution quality and robustness. Similarly, inspired by the significant results obtained by other researchers in past years, Wang et al. [21] suggested an innovative method, called composite DE (CoDE). This technique used three trial vector generation strategies and three control parameter settings. It randomly combines them to generate trial vectors. The performance of CoDE has been evaluated on 25 benchmark test functions developed for IEEE CEC2005 and it was very competitive with other compared algorithms.

Recently, super-fir multicriteria adaptive DE (SMADE) is proposed by Caraffini et al. [52], which is a Memetic approach based on the hybridization of the covariance matrix adaptive evolutionary strategy (CMAES) with modified DE, namely Multicriteria Adaptive DE (MADE). MADE makes use of four mutation/crossover strategies in an adaptive manner which are rand/1/bin, rand/2/bin, rand-to-best/2/bin, and current-torand/1. The control parameters CR and F are adaptively adjusted during the evolution. At the beginning of the optimization process, CMAES is used to generate a solution with a high quality which is then injected into the population of MADE.

The performance of SMADE has been evaluated on 28 benchmark test functions developed for IEEE CEC2013 and the experimental results are very promising. On the other hand, improving the trial vector generation strategy has attracted much research. In order to improve the convergence velocity of DE, Fan and Lampinen [13] suggested a trigonometric mutation scheme, which is considered a local search operator, and combined it with DE/rand/1 mutation operator to design the TDE algorithm. In another study, Zhang, and Sanderson (2009) introduced a new differential evolution (DE) algorithm, named JADE, to improve optimization performance by implementing a new mutation strategy “DE/current-top-best” with an optional external archive and by updating control parameters in an adaptive manner. Simulation results show that JADE was better than, or at least competitive to, other classic or adaptive DE algorithms such as Particle swarm and other evolutionary algorithms from the literature in terms of convergence performance. Along the same lines, to overcome the premature convergence and stagnation problems observed in the classical DE, Islam et al. [53] suggested modified DE with p-best crossover, named MDE_PBX. The innovative mutation strategy is similar to DE/current-to-best/1 scheme. It selects the best vector from a dynamic group of q% of the randomly selected population members. Furthermore, their crossover strategy uses a vector that is randomly selected from the p top-ranking vectors according to their objective values in the current population (instead of the target vector) to enhance the inclusion of generic information from the elite individuals in the current generation.

The parameter p is linearly reduced with generations to favour the investigation at the commencement of the search and exploitation during the later stages by gradually downsizing the elitist portion of the population. Das et al. [14] suggested two kinds of topological

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

neighbourhood models for DE in order to accomplish a better balance between its explorative and exploitative tendencies. In this method, called DEGL, two trial vectors are created by the worldwide and local neighbourhood-based mutation. These two trial vectors are combined to form the actual trial vector by using a weight factor. The performance of DEGL has been evaluated on 24 benchmark test functions and two real-world problems and showed very competitive results. In order to solve unconstrained and constrained optimization problems, Mohamed et al. [17] suggested an innovative directed mutation based on the weighted difference vector between the best and the worst individuals in a particular generation, is introduced. The new directed mutation rule is combined with the modified rudimentary mutation strategy DE/rand/1/bin, where only one of the two mutation rules is applied with the probability of 0.5. The suggested mutation rule is shown to improve the local search ability of the rudimentary differential evolution (DE) and to get a better trade-off between convergence rate and robustness.

Numerical experiments on well-known unconstrained and constrained benchmark test functions and five engineering design problems have shown that the new approach is efficient, effective, and robust. Similarly, to enhance worldwide and local search capabilities and concurrently increases the convergence speed of DE, Mohamed [17] introduced a new triangular mutation rule based on the convex combination vector of the triplet defined by the three randomly chosen vectors and the difference vector between the best and the worst individuals among the three randomly selected vectors. In this algorithm, called IDE, the mutation rule is combined with the rudimentary mutation strategy through a non-linear decreasing probability rule. A restart mechanism is also suggested to avoid premature convergence. IDE is tested on a famous set of unconstrained problems and shows its superiority to state-of-the-art differential evolution variants. Practically, it can be observed that the main modifications, improvements, and developments in DE focus on adjusting control parameters in an adaptive or self-adaptive manner. However, a few improvements have been applied to modify the structure and/or mechanism of the rudimentary DE algorithm or to suggest new mutation rules to improve the local and worldwide search ability of DE and overcome the glitches of stagnation or premature convergence.

The transparent representation of the sets of parameters used with an algorithm has a significant influence on the performance of machine learning procedures. Finding an optimum parameter arrangement can be viewed as a search problem, with the primary objective of increasing the quality of a machine learning model, done based on some performance metrics (e.g., accuracy).

One of the major problems for models to perform the task with relative accuracy is owing to the complex interactions between the parameters. Individual arrangement of the parameters might bring about suboptimal outcomes while making combinations of all different types is mostly impossible owing to the curse of dimensionality.

RESEARCH METHODOLOGY

The succinct but also promising CNN-based denoiser counts on a particular learning strategy (e.g., Residual learning) and a selection of hyper-parameter amalgamations (e.g., DNCNN). Therefore, in this study, the aim is to build a simple but effective CNN structure by paying attention to exploring the effective combinations of CNN hyper-parameters instead of the

structural blocks and layer connections. Despite all these, the proposed framework is also flexible because "genes" can represent any component (e.g., block type, loss function, learning strategy) of a CNN. In the experimental results, GA and PSO are the major optimization algorithms used to solve complex problems related to education, health systems, and image processing algorithms. The technique here is to understand how these algorithms are trained and at the same time provide better productivity in solving world-related problems. A very important challenge of using a genetic algorithm (GA) is how to speed up the evolutionary process dynamically in a gigantic search space. Contemporary techniques usually need large-scale computing resources (e.g., hundred gups). Nevertheless, in practice, these computational platforms are not available to most users. In order for me to address this issue, in this study, I present an Optimized Genetic Algorithm (OGA) with an Experience-based Greedy Exploration Strategy (EGES), and also Transfer Learning is leveraged to further expedite the GA evolution on the large dataset.

Experience-Based Greedy Exploration

The Genetic Algorithm (GA) was optimized with an experience-based greedy exploration strategy, which determines how and when to update gene sets. Experience represents CNN components (e.g., hyperparameters) learned from the last generation. The fine-gene sets θf were initialized with the genes from the top-performance CNNs which evolved in the previous evolutionary environment. This strategy enables locating the best individuals at an early stage. Thus, it does not need to explore the entire search space. The approach used stores and transfers such experience to the next generation.

Transfer Learning

Another novel contribution of the approach is using a transfer learning strategy [54] that allows the explored genes to be transferable among training data of different sizes or various modalities. For instance, I may use a small dataset to quickly optimize the gene-set space first, and then explore CNNs on a larger dataset by initializing a new population using the fine genes identified from the small dataset. Transfer learning further accelerates the evolution rate of the optimization.

Fitness Evaluation

The fitness function F (Pi) returns the restored image quality measure as a fitness score to each individual Pi. A Fitness score performs the following functions: (1) evaluating individual fitness; (2) updating gene sets; (3) serving as a stopping rule. Hence, the fitness function is critical for designing an effective GA-based method. Algorithm 1 presents the details of the proposed GA for exploring promising CNNs to handle medical image denoising.

Computational Complexity

It has been an arduous task to calculate the computational intricacy of an evolutionary metaheuristic. It could be found that the complexity of an evolutionary metaheuristic is largely dependent on the genetic operators (mutation, crossover, and selection), the execution, which may have a very important effect on the overall intricacy, the representation of the individuals, and the population, and the fitness function. Overall, a genetic algorithms complexity is O (GNE), where G represents the number of generations, N represents the population size, and E is represented by the size of the individuals which, in this case, is the number of periods of training each neural network. It is seen that the fitness function is not taken into consideration simply

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

because it is dependent on a specific application. In this case, the computational complexity is the same as the original genetic algorithm simply because I did not change the internal mechanism of the genetic algorithm; rather, I only applied transfer learning and greedy initialization strategy to the outside of the genetic algorithm.

The Complexity of GA Architecture

It is interesting to note that crossover and mutation operators are the core parameters of the Genetic Algorithm. By way of increasing the possibility of producing additional multifaceted simulations for better efficiency, an index is required. This index is essential to measure the effectiveness of the evaluated situation without conducting the assessment whose objective is to enhance them. On the other hand, it was problematic to set up an analytical function in different scenarios. The first is between the performance of the automated driver system (ADS) and the second is the appraisal scenarios, and so on. The analytical hierarchy process (AHP) was adopted to improve the situation complexity. This is however different from the objective function derived from equation [1].

The leverage factors of any complex problem are numerous and complicated depending on the multiple scenarios being conducted for the assessment. It is very difficult to determine the significance of each factor when designing the evaluation situation quantitatively. Comparative analysis among the factors belonging to the same parent node was required by this tree structure. The AHP is obtained by the significant degree of the node at the bottom layer. This is illustrated mathematically by:

$$T_k = \prod_{(i,j)\in\mathbb{R}} R_{i,j} \qquad [16]$$

Where T_k is the significance degree, $R_{i,j}$ is the relative significance of the node, $b_{i,j}$ and $\mathbb{R}$ is the set composed of the index of the path from $b_{f,k}$ to the root.

An evaluation scenario, X, is denoted by the values of situation elements, that is:

$\mathbb{x}_1 = \{X_1, X_2 \ldots\ldots X_{2n}\}$, where X_i denotes the value of the $i-th$ situation element. To determine the exact performance limit, X_i is generated randomly in its continuous range, consequently X_i is not equal to the discrete value, $b_{f,j}$. The linear interpolation is used to calculate the significance degree of $X_i \epsilon (b_{f,k-1}, b_{f,k})$ and the situation complexity index is obtained by the full function of:

$$C(X) = \sum_{i=1}^{m} T_i = \frac{T_{k-1} + X_i - V_{f,k-1}}{V_{f,k} - V_{f,k-1}} * (T_k - T_{k-1}) \qquad [17]$$

$$where\ C(X)\ is\ the\ situation\ complexity\ index.$$

Crossover Operator

From the period when the crossover point is selected at random, the traditional GA has no prior knowledge about the efficiency of the offspring. The possibility of missing the best one is high; consequently, good offspring exist in the candidate ones. Certainly, the whole crossover

operator (for singleton and manifold operators). This performs the single-point crossover at every position. The number of offsprings will eventually improve or increase by the geometric progression, provided all candidate offspring are selected to mutate. The neighbouring competition mechanism was enhanced to choose the final offspring pair in relation to their situation complexity index, which had a positive correlation with the probability of selection.

In the case of the manifold mutation operator, the traditional operator only generates one population, among which it has a low possibility to include good individuals. In order to increase the possibility of including good individuals, the manifold mutation operators are developed. On the population for $N\ times$, it performs traditional canonical mutation. The neighbouring competition mechanism is adopted to choose the final offspring in relation to their overall simulations' complexity index given by $X_i^m = Mut(X^C, W_m)$. By randomly selecting a candidate mutation offspring population as the final on (X^m) with the probability that:

$$W_i^m = \frac{e^{\alpha x C(X_i^m)}}{\sum_{k=1}^{N} e^{\alpha x C(X_k^m)}} \quad [18]$$

$where\ Mut(X^C, W_m)\ donotes\ the\ canonical\ mutation\ on\ X^C\ with\ the\ mutation\ probability, W_m[Ref]$

Parameters used to Determine the Performance of GA

Mean Square Error:

The Mean Squared Error (MSE) involves a measure that gives an indicator of how closely a fitted line is to the data points. In such a case, it is obtained by squaring the value for each data point. These data points are then used to multiply by the vertical distance between the point and the matching y value on the curve fit. In the case of a linear fit with two parameters, one can then add up all those values for all the data points and divide by the number of points minus two. ** Positive values are not canceled by negative ones according to the squaring process. The fit is more accurate to the data when the Mean Squared Error is lower. Regardless of what is plotted on the vertical axis, the MSE has the units squared.

Since the RMSE can be immediately translated into measurement units, it is a more accurate indicator of quality of fit than the correlation coefficient. The RMSE and observed variation in measurements taken at a typical point can be compared. A satisfactory match requires some similarity between the two. **To take into consideration the fact that the mean is calculated from the data rather than an external reference, the number of points - 2 must be used instead of just the number of points. Although this is a tiny change, for many trials, n is large enough that it barely registers. The RMSE and MSE are given by the formula:

$$MSE = \frac{\sum(\hat{y}_i - y_i)^2}{n}$$

$$RMSE = \sqrt{\frac{\sum(\hat{y}_i - y_i)^2}{n}}$$

$where\ "\boldsymbol{\Sigma}"\ is\ a\ symbol\ that\ means\ "sum",$

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

Also, "ŷ$\boldsymbol{i}$" *is the* predicted *value for the ith observation*,
"$\boldsymbol{yi}$" *is the observed value for the ith observation and*
"$\boldsymbol{n}$" *is the sample size*

The more frequently utilized is the RMSE because it is expressed in the same units as the response variable when evaluating how well a model fits a dataset. In contrast, the MSE is expressed in response variable squared units.

EXPERIMENTAL RESULTS

GA Results Obtained

In this section, I presented two sets of analysis using the GA algorithm with the engine dataset and are presented in Table 1 for 100 parameters and 25 populations using the GA parameters: percentage crossover (0.4), number of offspring (2*round (Pc*Pop/2), percentage mutation (0.7), mutation rate (0.5), and selection pressure (8). According to the results obtained after conducting two tests (100 and 50 parameters), with the number of iterations at 1-19, the best cost is 41.5664, and with 1-5 iterations (50 parameters), the best cost is 41.8784. Furthermore, the range of iterations produced by individual iterations @47, @48, @49, and @50, after some intelligence computations for 100 parameters while at 50 parameters, iterations pause at 34 and 39 with the best cost of 36.8182 and 36.0739, respectively. Overall, 100 iterations produce the best cost (30.8814), and at 50 iterations produce the worst cost (35.7872). Note that, the lower the intelligence cost the better the performance of the GA as represented in Table 1 below.

An experimental analysis was conducted against the GA using parameter metrics (MSE, RMSE, and error standard deviation). According to the results obtained from the train data with target and output determinants (Figure 4), the MSE and RMSE are 1168.3103 and 34.1806, respectively. This is followed by an intelligent computation with an error mean of 0.33844 and an error standard deviation of 34.2264. The analysis shows that for 100 parameters (iterations) the GA performance increases at -100 and +100 deviation from the mean. This could be attributed to the evolutionary algorithm for generation after generation as put forward by Charles Dwain in his study on the theory of evolution. Charles Dwain's theory was primarily centered on natural selection, a process that occurs over successive generations and is defined as the differential reproduction of genotypes. Natural selection requires heritable variation in a given trait and differential survival and reproduction associated with the possession of that trait.

Table 1: GA intelligence computation at 100 and 50 iterations

Number of Iterations @100 Parameters	Best Cost @ 100 Parameters	Number of Iterations @ 50 Parameters	Best Cost @50 Parameters
1-19	**41.5664**	**1-5**	**41.8784**
20-21	**41.1535**	**6-8**	**39.3993**
22-25	**40.4545**	**9-11**	**39.0312**
26-27	**40.3922**	**12-13**	**37.5491**
28-33	**39.4566**	**14-33**	**37.1175**
34-40	**38.5338**	**34**	**36.8182**
41-46	**38.2106**	**35-38**	**36.7077**
47	**36.8193**	**39**	**36.0739**
48	**36.2308**	**40-45**	**36.0484**
49	**35.9001**	**46-50**	**35.7872**
50	**34.8256**		
52-59	**34.2957**		
60-63	**34.0297**		
64-84	**32.3583**		
85-86	**32.1857**		
87-91	**32.0986**		
92-93	**31.7868**		
94-95	**30.9178**		
96-100	30.8814		

Comparing the test data verse, the train data, the train data analysis tends to concentrate massively toward the mean, and minimal error is obtained (-0.94) than the test data (0.34). The trained data was slightly directed away from the mean, while the test data was mainly directed toward the negative half of the distribution. This shows that the GA performance in regards to prediction and mitigating errors from the test data will enhance an effective means of predicting parameters that affect our challenges in healthcare, education, socio-economical and other domains. However, we found that the computation complexity of the algorithm requires time and resources to enhance the results. This analysis was carried out with a 2.3GHZ processor, Intel i7, 16GB RAM, and 1600 MHZ DDR3 computer.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

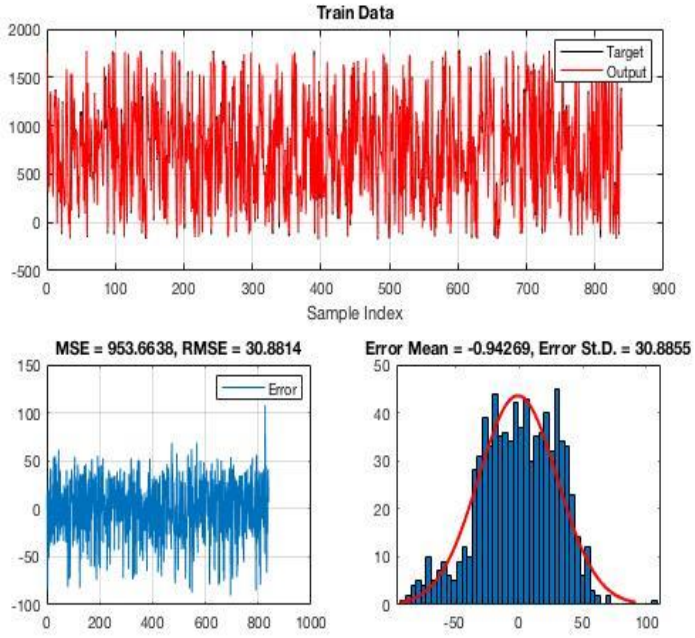

Figure 2: Chart showing result obtained from train data @ 100 parameters (iterations)

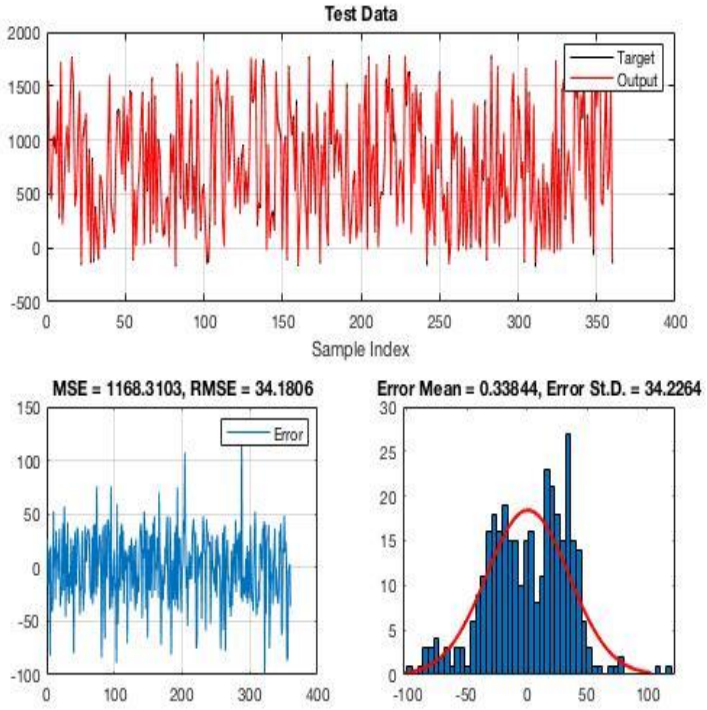

Figure 4.3: Chart showing result obtained from test data @100 parameters (iterations)

PSO Results Obtained

In this section, the study presents an analysis of PSO based on the ANFIS algorithm. For PSO, the sequence of solutions' convergence has been studied. These investigations led to

recommendations for choosing PSO parameters that are thought to cause convergence up to a certain point and prevent the particles in the swarm from diverging. Pedersen, however, objected to the analyses as being overly simplistic because they presuppose the swarm to have just one particle, that stochastic variables are not used, and that the points of attraction—the particle's best-known position p and the swarm's best-known position g-remain constant throughout the optimization process. It was demonstrated, nonetheless, that these simplifications have no impact on the limits identified by this research for parameters where the swarm is convergent. The modeling assumption used for the stability analysis of PSO has been significantly weakened in recent years, with the most current generalized finding applying to various PSO versions and utilizing what was demonstrated to be the minimal necessary modeling assumptions.

The parameters used to determine the PSO include a maximum number of iterations or parameters (100 and 50) computed at two different times, population size (swam size), inertia weight (1), inertia weight dumping ratio (0.99), personal learning weight (1), and global learning coefficients (2). The results are tabulated in Table 2. According to the results, the iteration continues to compute with the same cost at 100 parameters from 1-52 iterations with 41.2434 best cost and changes at 53 iterations. The computational intelligence continues until the 80th iteration with the best cost of 35.4107 obtained. The computation effect individually at 80, 87, 90, and 97. Additionally, the best cost was obtained at 98 and was maintained until 100 iterations. Alternatively, for 50 parameters, the number of iterations between1-39 computed the best cost of 41.3530. Single computation for best cost occurred at 38.3564 and 37.7367 with 45 and 46 iterations, respectively. The best cost function was computed at 47 iterations and maintained the best cost function until it reaches 50.

Table 2: PSO intelligence computation at 100 and 50 iterations

Number of Iterations @ 100 Parameters	Best Cost @ 100 Parameters	Number of Iterations @ 50 Parameters	Best Cost @50 Parameters
1-52	**41.2434**	**1-39**	**41.3530**
53-62	**39.5213**	**40-42**	**40.6157**
63-71	**38.5540**	**43-44**	**39.5064**
73-77	**36.8875**	**45**	**38.3564**
78-79	**35.8647**	**46**	**37.7367**
80	**35.4107**	**47-50**	**34.9202**
81-82	**35.2140**		
83-86	**34.5878**		
87	**33.6582**		
88-89	**32.8952**		
90	**32.4368**		
91-92	**31.7854**		
93-94	**30.9721**		
95-96	**30.9223**		
97	**30.8077**		
98-100	**30.7239**		

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

Figure 2 indicates the test data obtained from the PSO algorithm with RMSE, error mean, and standard deviation error against computation of 36.99, -7.97, and 36.17, respectively. A negatively skewed distribution in statistics is one where more values are concentrated on the right side (tail) of the distribution graph while the left tail of the distribution graph is longer. The excessive skewness of the data may cause the statistical tests to provide false results. To get the data as close to the normal distribution as possible, a transformation technique is used. Typically, only after the data has undergone all necessary transformations are the statistical tests done. The test data indicates that the data is skewed towards the left of the mean of the normal distribution curve, and indeed indicates the unreliability of the distribution of the data points (towards the left of the curve) as represented in Figure 2.

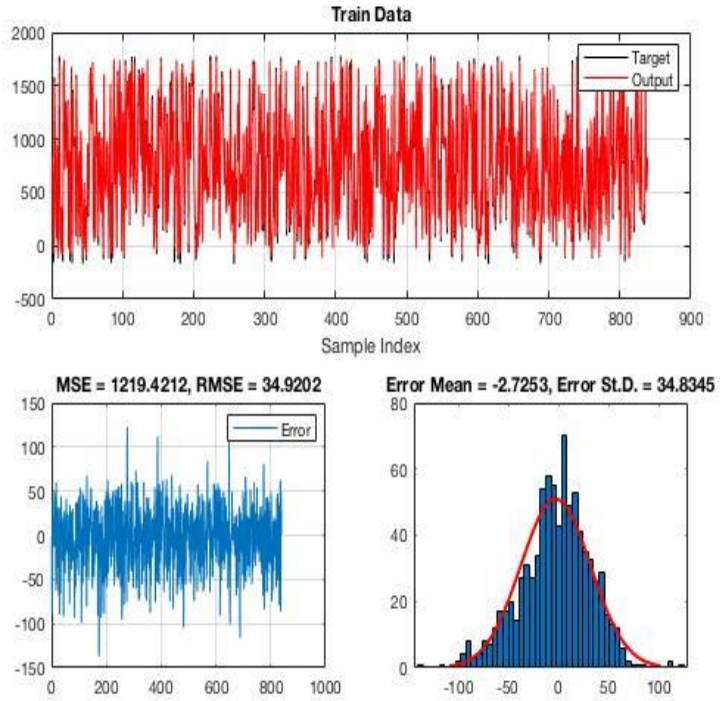

Figure 4: Chart showing PSO algorithm test data result

Starting from the trained data illustrated below (Figure 4), the deviations between the sample mean (average) and the actual population mean are computed by the standard error of the mean (SEM). It is always seen that SEM, by definition, is smaller than SD; and indeed, SEM gets smaller as the samples get larger. It is also found that the standard error of the mean, or simply standard error, shows how different the population mean is probable to be from a sample mean. Moreover, it tells you how much the sample mean would vary if you were to repeat a study using new samples from within a single population. In a nutshell, the standard error of the mean (SEM) calculates how big of a difference there is between the mean of a sample and the mean of the population. The square root of the total number of items in the sample (sample size) divides the SD to obtain the SEM. The standard deviation (SD) on the other hand gauges a dataset's dispersion from its mean. In addition to its being commonly used in statistics, SD is frequently used in finance as a stand-in for an investment's volatility or riskiness.

It is generally viewed that a simple division of the standard deviation by the square root of the sample size yields the SEM. By assessing the sample-to-sample variability of the sample mean, the standard error regulates the accuracy of the sample mean. As a measure of the population's actual mean, the sample mean is estimated using the standard error of the mean. Because the standard error of the mean is less sensitive to variation in sample size compared to standard deviation (SD), the sample mean is said to predict the population's true mean as the sample size increases more accurately.

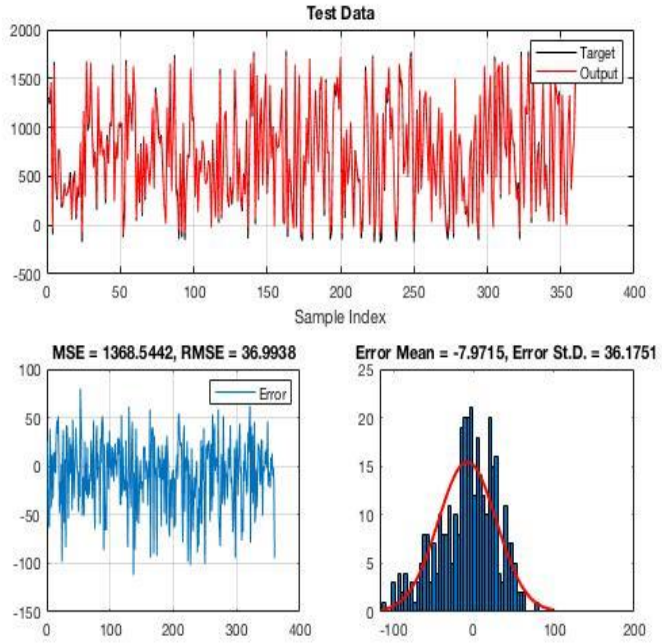

Figure 5: Chart showing PSO algorithm train data result

Comparing GA and PSO Results

This study suggests using Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) to build direct-driven permanent magnet synchronous generators (PMSGs) for use in wind turbine applications. These machines have a megawatt (MW) level power rating. The generator's limitations and specifications are listed. To achieve certain goals like optimizing efficiency, increasing torque, improving power factor, etc., the proposed design scheme optimizes various PMSG characteristics like Pole pair number, Linear current density, Air gap thickness, Rotor outer diameter, Relative width of the permanent magnet, etc. We compare the outcomes of the GA algorithm to the PSO algorithm (Figure 4.6).

Particle Swarm Optimization (PSO) is found to perform better than the Genetic Algorithm because it concurrently conducts global and local searches, whereas the Genetic Algorithm focuses primarily on the global search. Results demonstrate that the suggested PSO optimization method yielded competitive designs in comparison to the GA algorithm and was simple to create and apply (Figure below). With a focus on how each operator influences search behavior in the issue space, the operators of each paradigm are examined. The research's objectives are to offer more understanding of how each paradigm functions and to make suggestions for how performance might be enhanced by combining elements of several

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

paradigms. Figure 4.6 and Figure 4.7 shows a thematic difference between GA and PSO algorithms.

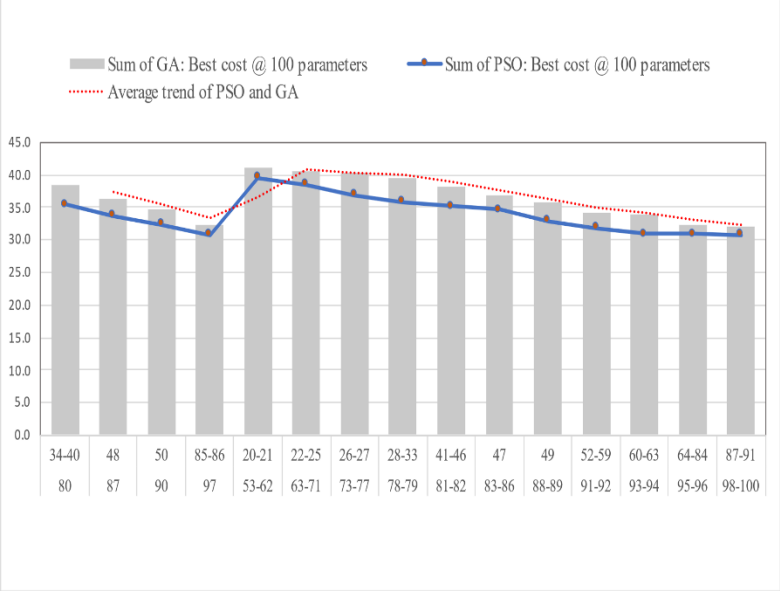

Figure 6: Comparing GA and PSO best cost @100 parameters

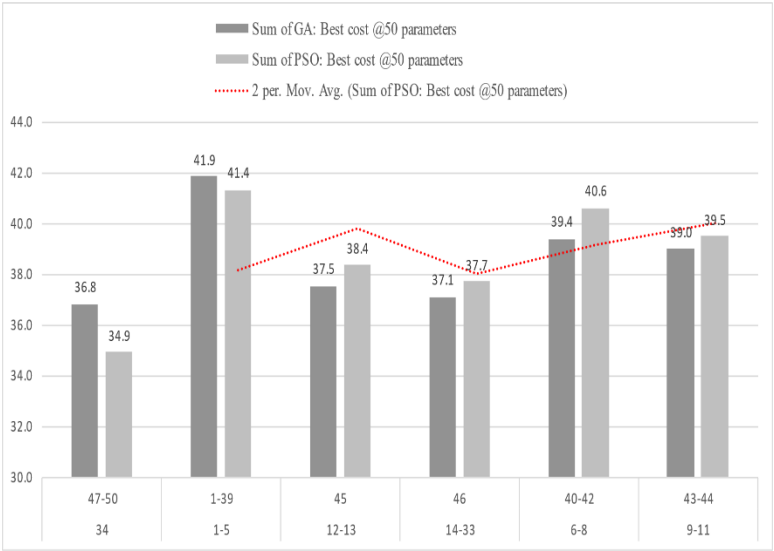

Figure 7: Comparing GA and PSO best cost @50 parameters

Summary of Findings

The results obtained by the GA algorithm and those by the PSO algorithm were compared. The performance of Particle Swarm Optimization is found to be better than the Genetic Algorithm. This is so because the PSO algorithm carries out global search and local searches simultaneously, whereas the Genetic Algorithm concentrates mainly on the global search.

Genetic Algorithms (GAs) are known to be adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are generally based on the ideas of natural selection and genetics. These simply imply that they are intelligent

exploitation of random search provided with historical data to help direct the search into the region of better performance in solution space. Genetic algorithms are frequently used to generate high-quality solutions for optimization problems and search problems.

Moreover, genetic algorithms simulate the process of natural selection which implies that those species that can adapt to changes in their environment are able to survive and replicate and then go to the next generation. In simple succinct terms, they simulate "survival of the fittest" among individuals of consecutive generations for solving a problem. Each generation consists of a population of individuals and each individual represents a point in search space and a possible solution. Every individual with the population is represented as a string of bits/ integers/characters/floats etc. This string is analogous to the Chromosome.

Genetic algorithms when critically viewed, are found to be based on an analogy with the genetic structure and behaviour of chromosomes of the population. The following is the foundation of gas based on this analogy:

- ✓ Individuals in the population compete for resources and mate.
- ✓ Those individuals who are successful (fittest) then mate to create more offspring than others.
- ✓ Genes from the "fittest" parent propagate throughout the generation, and sometimes parents create offspring which is better than either parent.
- ✓ Thus, each successive generation is more suited to its environment.

CONCLUSIONS AND FUTURE WORK

A relatively new heuristic technique called Particle Swarm Optimization (PSO) is based on the swarming behaviors of real creatures. PSO and the GA are both evolutionary search methods, meaning that they switch from one set of points to another inside an iteration while clearly improving upon the prior values utilizing a combination of probabilistic and deterministic principles. On the other hand, the GA is a well-known and widely used algorithm with a wide range of uses and iterations. Although GA and PSO are both significant components of evolutionary optimization methods, they both have drawbacks that restrict their use to a small number of issues. A combination of GA and PSO can be utilized to solve these issues and boost the optimization of each of them.

As a technique appearing not long time, PSO algorithm has received wide attentions in recent years. Advantages of PSO algorithm can be summarized as follows:

- ✓ It has excellent robustness and can be used in different application environments with a little modification.
- ✓ It has strong distributed ability, because the algorithm is essentially the swarm evolutionary algorithm, so it is easy to realize parallel computation.
- ✓ It can converge to the optimization value quickly.
- ✓ It is easy to hybridize with other algorithms to improve its performance.

A highly parallel, unpredictable, and adaptable optimization technique based on "survival of the fittest" is known as a genetic algorithm. The "chromosome" group that the problem solution represents is duplicated, crossed, and altered. In order to arrive at the best or most agreeable answer to the issue, it has evolved from generation to generation and ultimately converged to the most adaptive Group. Its benefits include straightforward operations and

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

principles, strong generality, the capacity for implicit parallelism, and the capability of global solution search, which is frequently employed in combinatorial optimization problems. The genetic algorithm's crossover operator, which controls the genetic algorithm's overall convergence, is its most crucial component. Children inheriting their parents' outstanding traits and their children's viability is the most crucial factor in crossover operator design. Through local search, the performance of the offspring is to be enhanced. Popular diagnostic methods include X-rays, MRI, and other medical imaging techniques like computed tomography (CT). These methods are also subject to noise.

Finally, it is viewed that evolutionary computation provides an opportunity to expand our technological abilities beyond deep learning. Building on guided, exuberant exploration, it can create surprising and more complex solutions than those of human designs. Using this technology, artificial intelligence (AI) can potentially improve many industries, such as agriculture, health care, finance, homeland security, and online retail. While deep learning, a machine learning technique that helps teach computers to do what comes naturally to humans has brought us this far, evolutionary computation can bring us to the AI of the future — known as creative AI.

References

[1] Storn, R. (1995). Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, 11 .

[2] Price, K., Storn, R. M., & Lampinen, J. A. (2006). Differential evolution: a practical approach to global optimization. Springer Science & Business Media.

[3] Li, X. and Yin, M., 2016. Modified differential evolution with self-adaptive parameters method. Journal of Combinatorial Optimization, 31(2), pp.546-576.

[4] Zhu, H., He, Y., Wang, X. and Tsang, E.C., 2017. Discrete differential evolutions for the discounted {0-1} knapsack problem. International Journal of Bio-Inspired Computation, 10(4), pp.219-238.

[5] Hachicha, N., Jarboui, B. and Siarry, P., 2011. A fuzzy logic control using a differential evolution algorithm aimed at modelling the financial market dynamics. Information Sciences, 181(1), pp.79-91.

[6] Dong, C.R., Ng, W.W., Wang, X.Z., Chan, P.P. and Yeung, D.S., 2014. An improved differential evolution and its application to determining feature weights in similarity-based clustering. Neurocomputing, 146, pp.95-103.

[7] El-Quliti, S.A., Ragab, A.H.M., Abdelaal, R., Mohamed, A.W., Mashat, A.S., Noaman, A.Y. and Altalhi, A.H., 2015. A nonlinear goal programming model for university admission capacity planning with modified differential evolution algorithm. Mathematical Problems in Engineering, 2015.

[8] El-Qulity, S.A. and Mohamed, A.W., 2016. A generalized national planning approach for admission capacity in higher education: a nonlinear integer goal programming model with a novel differential evolution algorithm. Computational Intelligence and Neuroscience, 2016.

[9] ElQuliti, S.A.H. and Mohamed, A.W., 2016. A large-scale nonlinear mixed-binary goal programming model to assess candidate locations for solar energy stations: an improved real-binary differential evolution algorithm with a case study. Journal of Computational and Theoretical Nanoscience, 13(11), pp.7909-7921.

[10] Greenwood, G.W., 2009. Using differential evolution for a subclass of graph theory problems. IEEE Transactions on Evolutionary Computation, 13(5), pp.1190-1192.

[11] Noman, N. and Iba, H., 2008. Accelerating differential evolution using an adaptive local search. IEEE Transactions on evolutionary Computation, 12(1), pp.107-125.

[12] Das, S., Abraham, A., Chakraborty, U.K. and Konar, A., 2009. Differential evolution using a neighborhood-based mutation operator. IEEE transactions on evolutionary computation, 13(3), pp.526-553.

[13] Lampinen, J. and Zelinka, I., 2000, June. On stagnation of the differential evolution algorithm. In Proceedings of MENDEL (pp. 76-83).

[14] Das, S. and Suganthan, P.N., 2010. Differential evolution: A survey of the state-of-the-art. IEEE transactions on evolutionary computation, 15(1), pp.4-31.

[15] Subhasis, S. 2021. An Introduction to Particle Swarm Optimization (PSO) Algorithm

[16] Liang JJ, Qin BY, Suganthan PN, Hernndez-Diaz AG (2013) Problem definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization, Zhengzhou University/ Nanyang Technological University, Zhengzhou, China/Singapore, Technical Report 201212

[17] Lampinen, J. and Zelinka, I., 2000, June. On stagnation of the differential evolution algorithm. In Proceedings of MENDEL (pp. 76-83).

[18] Mohamed, A.W., Sabry, H.Z. and Farhat, A., 2011, December. Advanced differential evolution algorithm for global numerical optimizatiom. In 2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE) (pp. 156-161). IEEE.

[19] Qin, A.K., Huang, V.L. and Suganthan, P.N., 2008. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE transactions on Evolutionary Computation, 13(2), pp.398-417.

[20] Mallipeddi, R., Suganthan, P.N., Pan, Q.K. and Tasgetiren, M.F., 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. Applied soft computing, 11(2), pp.1679-1696.

[21] Wang, Y., Cai, Z. and Zhang, Q., 2011. Differential evolution with composite trial vector generation strategies and control parameters. IEEE transactions on evolutionary computation, 15(1), pp.55-66.

[22] Zhang, J. and Sanderson, A.C., 2009. JADE: adaptive differential evolution with optional external archive. IEEE Transactions on evolutionary computation, 13(5), pp.945-958.

[23] Feoktistov, V., 2006. Differential evolution (pp. 1-24). Springer US.

[24] Omran, M.G. and Engelbrecht, A.P., 2006, June. Self-adaptive differential evolution methods for unsupervised image classification. In 2006 IEEE Conference on Cybernetics and Intelligent Systems (pp. 1-6). IEEE.

[25] Ronkkonen, J., Kukkonen, S. and Price, K.V., 2005, September. Real-parameter optimization with differential evolution. In 2005 IEEE congress on evolutionary computation (Vol. 1, pp. 506-513). IEEE.

[26] Zhang, X.X. and Chen, W.R., 2010. Dynamic multi-group self-adaptive differential evolution algorithm with local search for function optimization. ACTA ELECTONICA SINICA, 38(8), p.1825.

[27] Holland, J.H., 2000. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. Evolutionary computation, 8(4), pp.373-391.

[28] Youssef, H., Sait, S.M. and Adiche, H., 2001. Evolutionary algorithms, simulated annealing and tabu search: a comparative study. Engineering Applications of Artificial Intelligence, 14(2), pp.167-181.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.

[29] Adrianto, D., 2014. Comparison using particle swarm optimization and genetic algorithm for timetable scheduling. Journal of Computer Science, 10(2), p.341.

[30] Horn, J., Nafpliotis, N. and Goldberg, D.E., 1994, June. A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence (pp. 82-87). Ieee.

[31] Booker, L.B., Goldberg, D.E. and Holland, J.H., 1989. Classifier systems and genetic algorithms. Artificial intelligence, 40(1-3), pp.235-282.

[32] Goldberg, D.E. and Richardson, J., 1987, July. Genetic algorithms with sharing for multimodal function optimization. In Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms (Vol. 4149). Hillsdale, NJ: Lawrence Erlbaum.

[33] Yang, J. and Honavar, V., 1998. Feature subset selection using a genetic algorithm. In Feature extraction, construction and selection (pp. 117-136). Springer, Boston, MA.

[34] Bansal, N., Sharma, A. and Singh, R.K., 2019. An Evolving Hybrid Deep Learning Framework for Legal Document Classification. Ingénierie des Systèmes d'Information, 24(4).

[35] Turner, A.J. and Miller, J.F., 2017. Recurrent cartesian genetic programming of artificial neural networks. Genetic Programming and Evolvable Machines, 18(2), pp.185-212.

[36] Fofanah, A. J., Bundu, H. R., & Kargbo, J. G. (2022). A generic heart diseases prediction and application of genetic algorithms in healthcare systems: Genetic algorithm and machine learning algorithm approaches. International Journal of Health Sciences, 6(S3),12264–12290. https://doi.org/10.53730/ijhs.v6nS3.9024

[37] Fofanah, A.J. and Hwase, T.K., 2022. An Intelligence Computation of Genetic Algorithm and Its Application in Healthcare Systems: Algorithms, Methods, and Predictions. American Journal of Health Research, 10(6), pp.225-256.

[38] Stanley, K.O. and Miikkulainen, R., 2002, July. Efficient reinforcement learning through evolving neural network topologies. In Proceedings of the 4th Annual Conference on genetic and evolutionary computation (pp. 569-577).

[39] Yao, X. and Higuchi, T., 1999. Promises and challenges of evolvable hardware. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 29(1), pp.87-97.

[40] He, X., Zhao, K. and Chu, X., 2021. AutoML: A survey of the state-of-the-art. Knowledge-Based Systems, 212, p.106622.

[41] Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B. and Vanschoren, J., 2019. An open source AutoML benchmark. arXiv preprint arXiv:1907.00909.

[42] Liu, P., El Basha, M.D., Li, Y., Xiao, Y., Sanelli, P.C. and Fang, R., 2019. Deep evolutionary networks with expedited genetic algorithms for medical image denoising. Medical image analysis, 54, pp.306-315.

[43] Shi, X., Mueller, J., Erickson, N., Li, M. and Smola, A.J., 2021. Benchmarking multimodal automl for tabular data with text fields. arXiv preprint arXiv:2111.02705.

[44] Xin, D., Wu, E.Y., Lee, D.J.L., Salehi, N. and Parameswaran, A., 2021, May. Whither automl? understanding the role of automation in machine learning workflows. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (pp. 1-16).

[45] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J., 2018. An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4), pp.219-354.

[46] Li, Y., 2017. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.

[47] Xiao, M., Yang, B., Wang, S., Zhang, Z., Tang, X. and Kang, L., 2022. A feature fusion enhanced multiscale CNN with attention mechanism for spot-welding surface appearance recognition. Computers in Industry, 135, p.103583.

[48] Wang, Q., Bu, S., He, Z. and Dong, Z.Y., 2020. Toward the prediction level of situation awareness for electric power systems using CNN-LSTM network. IEEE Transactions on Industrial Informatics, 17(10), pp.6951-6961.

[49] Liu, K., Kang, G., Zhang, N. and Hou, B., 2018. Breast cancer classification based on fully connected layer first convolutional neural networks. IEEE Access, 6, pp.23722-23732.

[50] Qin, A.K., Huang, V.L. and Suganthan, P.N., 2008. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE transactions on Evolutionary Computation, 13(2), pp.398-417.

[51] Zhang, Q., Chen, H., Heidari, A.A., Zhao, X., Xu, Y., Wang, P., Li, Y. and Li, C., 2019. Chaos-induced and mutation-driven schemes boosting salp chains-inspired optimizers. Ieee Access, 7, pp.31243-31261.

[52] Caraffini, F., Neri, F., Cheng, J., Zhang, G., Picinali, L., Iacca, G. and Mininno, E., 2013, June. Super-fit multicriteria adaptive differential evolution. In 2013 IEEE congress on evolutionary computation (pp. 1678-1685). IEEE.

[53] Islam, S.M., Das, S., Ghosh, S., Roy, S. and Suganthan, P.N., 2011. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(2), pp.482-500.

[54] Long, M., Zhu, H., Wang, J. and Jordan, M.I., 2017, July. Deep transfer learning with joint adaptation networks. In International conference on machine learning (pp. 2208-2217). PMLR.

URL: http://dx.doi.org/10.14738/jbemi.102.14427.