Mathematical and Computational Applications

MDPI

Article

Evolutionary Algorithms Enhanced with Quadratic Coding and Sensing Search for Global Optimization

Abdel-Rahman Hedar [1,2,*], Wael Deabes [1,3], Majid Almaraashi [4] and Hesham H. Amin [1,5]

1 Department of Computer Science in Jamoum, Umm Al-Qura University, Makkah 25371, Saudi Arabia; wadeabes@uqu.edu.sa (W.D.); hhabuelhasan@uqu.edu.sa (H.H.A.)
2 Department of Computer Science, Faculty of Computers & Information, Assiut University, Assiut 71526, Egypt
3 Computers and Systems Engineering Department, Mansoura University, Mansoura 35516, Egypt
4 Department of Computer Sciences, College of Computing and Information Technology, University of Jeddah, Jeddah 23218, Saudi Arabia; malmaraashi@uj.edu.sa
5 Department of Electrical Engineering, Computers and Systems Section, Faculty of Engineering, Aswan University, Aswan 81528, Egypt; hhamin@aswu.edu.eg
* Correspondence: hedar@aun.edu.eg or ahahmed@uqu.edu.sa; Tel.: +966-55-0086-411 or +20-10-0070-4940

Received: 26 November 2019; Accepted: 14 January 2020; Published: 16 January 2020

Abstract: Enhancing Evolutionary Algorithms (EAs) using mathematical elements significantly contribute to their development and control the randomness they are experiencing. Moreover, the automation of the primary process steps of EAs is still one of the hardest problems. Specifically, EAs still have no robust automatic termination criteria. Moreover, the highly random behavior of some evolutionary operations should be controlled, and the methods should invoke advanced learning process and elements. As follows, this research focuses on the problem of automating and controlling the search process of EAs by using sensing and mathematical mechanisms. These mechanisms can provide the search process with the needed memories and conditions to adapt to the diversification and intensification opportunities. Moreover, a new quadratic coding and quadratic search operator are invoked to increase the local search improving possibilities. The suggested quadratic search operator uses both regression and Radial Basis Function (RBF) neural network models. Two evolutionary-based methods are proposed to evaluate the performance of the suggested enhancing elements using genetic algorithms and evolution strategies. Results show that for both the regression, RBFs and quadratic techniques could help in the approximation of high-dimensional functions with the use of a few adjustable parameters for each type of function. Moreover, the automatic termination criteria could allow the search process to stop appropriately.

Keywords: evolutionary algorithms; genetic algorithm; evolution strategies; regression; neural networks; quadratic coding; quadratic search

1. Introduction

Evolutionary Algorithms (EAs) are considered to be one of the core methods applied in the area of computational intelligence [1]. Generally, EAs constitute a class of the main global search tools which can be adapted to deal with many forms of nonlinear and hard optimization problems. Developing applicable versions of EAs is hugely required to meet the fast-growing of optimization applications in all aspects of science [2,3]. Recently, there is an extreme interest in adapting EAs to be used with other computational intelligence techniques in different application areas. Although there are a vast number of attempts to modify the EAs to solve global optimization problems, most of them invoke only heuristic design elements, and they do not use mathematical mechanisms [4,5].

On the other hand, EAs have been applied in various optimization problems such as continuous, discrete, and combinatorial problems with and without constraints as well as mixed optimization problems [6,7]. Moreover, EAs are considered to be a milestone in the computational intelligence filed [8]. Thus, developing practical schemes of EAs is remarkably required to meet the fast increasing of many applications in different real-life science [2,9]. However, EAs still have no automatic termination criteria. Thus, EAs algorithms cannot determine when or where they can terminate, and a user should pre-specify a criterion for that purpose. Typically, methods of termination criteria are such as when there has been no improvement in a pre-specified number of generations, when reaching a pre-specified number of generations, or when the objective function value has reached a pre-specified value. Another important designing issue is that controlling the randomness of some evolutionary operations should be considered in designing effective evolutionary-based search methods. This has motivated many researchers and was one of the main reasons for inventing the "memetic algorithms" [10,11]. Moreover, more deterministic operations have been highly recommended in developing the next generation genetic algorithms [12].

The main goal of this research is to construct an effective and intelligent method that looks for optimal or near-optimal solutions to non-convex optimization problems. A global search method is to solve the general unconstrained global optimization problem:

$$\min_{x \in X} f(x). \tag{1}$$

In Equation (1), f is a real-value function that is defined in the search space $X \subseteq R^n$ with variables $x \in X$. Many methods of EAs have been suggested to deal with such problems [13,14]. This optimization problem has also been considered by different heuristic methods such as; tabu search [15–17], simulated annealing [18–20], memetic algorithms [11], differential evolution [21,22], particle swarm optimization [23,24], ant colony optimization [25], variable neighborhood search [26], scatter search [27,28] and hybrid approaches [29–31]. Multiple applications in various areas such as computer engineering, computer science, economic, engineering, computational science and medicine can be expressed or redefined as problem in Equation(1), see [2,32] and references therein. The considered problem is an unconstrained one; however, there are several constraint-handling techniques that have been proposed to extend some of the previously mentioned research to deal with the constrained version of Problem (1) [33–35]. Some of the common techniques are to use penalty functions, fillers and dominance-based technique [36,37].

In this research, the proposed methods are expressed using quadratic models that partially approximate the objective function. Two design models have been invoked to construct these methods. In the first model called Quadratic Coding Genetic Algorithm (QAGA), trial solutions are encoded as coefficients of quadratic functions, and their fitness is evaluated by the objective values at quadratic optimizers of these models. Through generations, these quadratic forms are adapted by the genetic operators: selection, crossover, and mutation. In the second model, Evolution Strategies (ESs) are modified by exploiting search memory elements, automatic sensing conditions, and a quadratic search operator for global optimization. The second proposed method is called Sensing Evolution Strategies (SES). Specifically, the regression method and Artificial Neural Networks (ANN) are used as an approximation process. Explicitly, Radial Basis Functions (RBFs) are invoked as an efficient local ANN technique [38].

The obtained results show that the quadratic approximation and coding models could improve the performance of EAs and could reach faster convergence. Moreover, the mutagenesis operator is much effective and cheaper than some local search techniques. Furthermore, the final intensification process can be started to refine the elite solutions obtained so far. In general, the numerical results indicate that the proposed methods are competitive with some other versions of EAs.

The rest of this paper is structured as follows. Related work is discussed in Section 2. In Section 3, the components of the proposed quadratic models are illustrated. The details of the proposed methods

QCGA and SES are explained in Sections 4 and 5, respectively. In Section 6, numerical experiments aiming at analyzing and discussing the performance of the proposed methods and their novel operators are presented. Finally, the conclusion makes up Section 7.

2. Related Work

The main contributions of this research are to use the wise guidance of mathematical techniques through quadratic models and to design smarter evolutionary-based methods that can sense the progress of the search to achieve finer intensification, wider exploration, and automatic termination.

In designing the proposed methods, EAs are customized with different guiding strategies. First, an exploration and exploitation strategy is proposed to provide the search process with accelerated automatic termination criteria. Specifically, matrices called Gene Matrix (GM) are constructed to sample the search space [1,39,40]. The role of the *GM* is to aid the exploration process. Typically, *GM* can equip the search with novel diverse solutions by applying a unique type of mutation that is called "mutagenesis" [39,40]. Principally, mutagenesis may be defined as a nature mechanism by which the genes of an individual are changed, which lead to the mutation process. Thus, this mechanism is a driving force of evolution process [40]. Thus, the mutagenesis operator alters some survival individuals to accelerate the exploration and exploitation processes [40]. On the other hand, *GM* is used to lead the search process to know how far the exploration process has gone to judge a termination point. Moreover, the mutagenesis operation lets the proposed method, the GA with Gene Matrix, perform like a so-called "Memetic Algorithm" [41,42] to accomplish a faster convergence [40].

Practical termination criteria are one of the main designing issues in EAs. Typically, research is scant of termination criteria for EAs. However, in [43], an empirical method is used to determine the maximum number of needed generations by using the problem characteristics to converge for a solution. Moreover, eight termination criteria have been discussed in [44] that proposed a way of using clustering techniques to test the distribution of specific generation individuals in the search space. Different search memory mechanisms have been adapted as termination criteria to reflect the wideness of the exploration process [39,40,45]. There are four classes of termination criteria of EAs [39]. The first class is called T_{Fit} criterion, and it is based on calculating the best fitness function values over generations. This method keeps tracking the best fitness function values in each generation, and it terminates the algorithm if the values do not significantly change [46,47]. A T_{Pop} criterion is the second class measuring the population over generations. In each generation, the distances between chromosomes are measured and used to terminate the algorithm if these distances are very close. Work in [48] uses T_{Pop} criterion for termination by adding distances among individuals and making sure it is smaller than a predefined threshold. The other two classes are T_{Bud} and T_{SFB}. The T_{Bud} criterion applies a computational budget such as the maximum number of generations or function evaluations for termination [49–51]. In the T_{SFB} criterion, a search feedback measure evaluates the exploration and exploitation processes. Measures such as the distribution of the generated solutions or the frequencies of visiting regions inside the search space are invoked to test the search process. In [45], the exploration process is examined by a diverse index set of points created at the beginning of the search process, which tries to guarantee a well-distribution of the generated points over the search space. The algorithm is terminated when most of the regions get visited.

These termination criteria may struggle from some difficulties [45,52]. Applying T_{Fit} only in EAs can easily make it trapped in local minima. Also, it is costly to have the entire population or a part of it convergent when using T_{Pop}, while it is sufficient to have one chromosome closely reach the global minima. Gaining prior information about the search problem is a big concern when invoking T_{Bud} criteria. Lastly, however using T_{SFB} looks effective; it may suffer the dimensionality problem, and it is complex to save and test huge historical search information. Therefore, implementing mixed termination criteria in EAs to overcome these problems is appreciated [53,54].

Evolutionary computing and artificial neural networks are main classes in computational intelligence [55,56]. Most of the research focus on how to use EAs to get more trained ANN [57,58].

There is relatively little research studying how to do the opposite by using ANN to design better EA-based methods or even to construct global optimization methods such in [59,60]. In this research, the Radial Basis Functions (RBFs) are invoked as an efficient local ANN technique to help the EAs to reach promising regions and directions. Generally, other types of ANN could be used for the same task, such as the well-known multi-layer perceptron (MLB) with a training algorithm such as the back-propagation algorithm. However, RBFs is a local learning ANNs that could overcome the problem of long iterations learning process of other types of ANNs. In RBFs, the number of iterations is bounded maximum to the number of input samples. In other words, the middle layer adds one neuron per any new input sample, as shown in Figure 1.

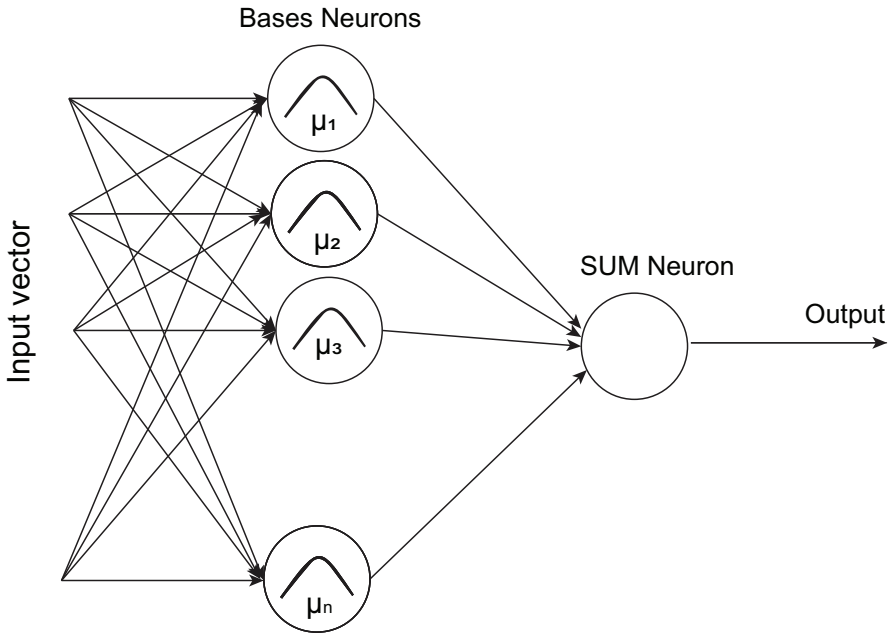

Figure 1. Representation of RBF architecture.

3. Quadratic Approximation Methods

One of the main steps in the proposed algorithms is to find out the best and fastest approximation for a high-dimensional function. Generally, two methods have been applied for this purpose; quadratic regression and radial basis function as a local supervised ANN. In both methods, the following Equation (2) and its derivative are used for the fitting of the parameters:

$$f(x) = x^T Ax + bx + c, \tag{2}$$

where A is an $n \times n$ matrix, b is a vector with n-dimension, and c is a scalar. These individuals' quadratic forms are adapted by the genetic operators through generations to fit the given objective function locally.

The least-square quadratic regression (also known as a nonlinear or second-order regression) is used because it is a fast and simple method for fitting. Moreover, it produces a good approximation and has very encouraging properties that can solve high-dimensional functions by finding A, b, and c in Equation (2). Thus, the idea is to choose a quadratic function that minimizes the sum of the squares of the vertical distances between the original fitness function values at generated solutions and an approximate quadratic function with finding coefficients A, b and c, as shown in Figure 2. Then, approximate quadratic optimizers of the quadratic models can be given by Equation (3), which is represented by the derivative of Equation (2).

$$x^* = -\frac{1}{2}A^{-1}b. \tag{3}$$

It is computationally expensive to compute the inverse of a full matrix A in Equation (3), especially for high dimension problems. Hence, it is recommended to apply an efficient relaxation method to approximate A to a diagonal matrix. Therefore, an approximate quadratic solution can be given as:

$$x^* = -\frac{1}{2}\sum_{i=1}^{n}\frac{b_i}{a_{i,i}}, \tag{4}$$

where $a_{i,j}$ and b_i are the entities of A and b, respectively, with $a_{i,i} \neq 0$, $i = 1, \ldots, n$ and $i = 1, \ldots, n$.

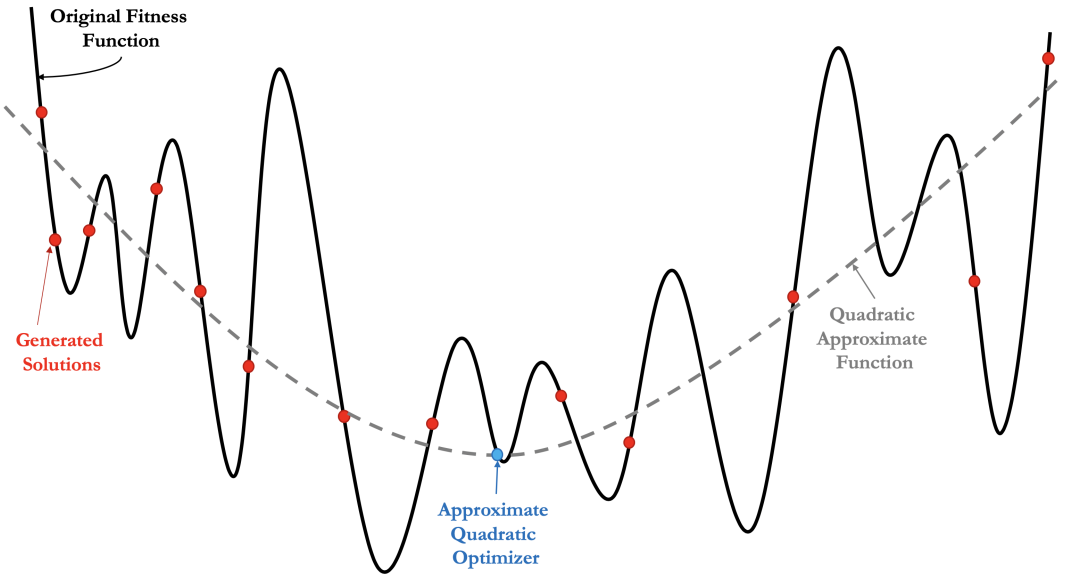

Figure 2. Representation of quadratic approximation.

3.1. Regression

Quadratic regression models are usually generated using least-squares methods [61]. Procedure 1 explains the steps of obtaining an approximate solution using the least-squares regression models. Starting with a set S of previously generated solutions, the least-squares regression method is called to find an approximate quadratic function, as in Equation (2), that fits those solutions. Then, an approximate optimizer can be calculated using Equation (3).

Procedure 1. Quad-LSE(S)

1. *Using least-squares regression, generate a quadratic model as in Equation (2) that fits the solutions in the sample set S.*
2. *Get the coefficients A, b, and c, of the generated approximate quadratic function.*
3. *Return with the minimum point of the approximate quadratic function using Equations (3) and (4).*

3.2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are used in various optimization applications, such as in the areas of complex nonlinear system identification and global approximation. Many researchers have proved that multi-layer feed-forward ANNs, using different types of activation functions, works as a universal approximator [38]. Moreover, continuous nonlinear functions can be approximated and/or

interpolated with feed-forward ANNs and can be used to approximate the outputs of dynamical systems directly. Generally, ANNs are employed with EA in two main ways. Typically, most research focus on optimizing ANNs parameters, especially weights, using evolutionary techniques, especially GA [62]. These results, in better classification of the ANNs, use of the power of both methods, the learning ability of the ANNs, and the best parameter values of EA. On the other hand, little research, such as this one, do the opposite thing by optimizing the evolutionary algorithms using ANNs for various purposes [63–65].

In this research, the speedup of the optimization process has been done using an RBF neural network. The RBF has been used for searching for the approximated quadratic function for a continuous function of variables such as shown in Appendices A and B. Therefore, The RBF builds local representations of the functions and so their parameters. However, RBFs are simpler to be initialized and trained than feed-forward multi-layer perceptrons (MLPs) neural networks. Thus, RBFs overcome the problem of the training iteration process as it is a local learning type of neural network. Therefore, the iterations are bounded as a maximum of the number of input samples. However, RBFs approximation may be very attractive for approximating complex functions in numerical simulations. Moreover, RBFs would allow randomly scattered data to generalize easily to several space dimensions, and it can be accurate in the interpolation process. With all those motivations, this study used the properties of RBFs approximations to develop a new global optimization tool.

As shown in Figure 1, the RBF neural network is a two-layer ANN, where each hidden unit exploits a radial (Gaussian) activation function for the hidden layer processing elements. The output unit applies a weighted sum of all hidden unit outputs. Although the inputs into the RBF network are nonlinear, the output is linear due to the structure of the RBF neural network. By adjusting the parameters of the Gaussian functions in the hidden layer neurons, each one reacts only to a small region of the input space. However, for successful performance and implementation of RBFs is to find reasonable centers and widths of the Gaussian functions. In this research, during simulation, a MATLAB function is used to determine the Gaussian centers and widths using the input data of the desired function from Appendices A and B. Once the hidden layer has completed its local training and all parameters adjustment, the output layer then adds the outputs of the hidden layer based on its weights. The steps of obtaining an approximate solution using the ANN models are shown in Procedure 2.

Procedure 2. Quad-ANN(S)

1. *A set S of generated samples of solutions is used for training the RBF to obtain an approximate quadratic function as in Equation (2).*
2. *Get the coefficients A, b and c of the approximate quadratic function obtained by the RBF network.*
3. *Return with the minimum point of the approximate quadratic function using Equations (3) and (4).*

4. Quadratic Coding Genetic Algorithm

One of the proposed methods in this research is called the Quadratic Coding Genetic Algorithm (QCGA). In the following, the main structure and design of the proposed QCGA method are described. Generally, the QCGA method uses a population of μ individuals or solutions. Each individual represents a quadratic form model as in Equation (2). These quadratic form individuals are adapted by the genetic operators through generations to fit the given objective function locally. Therefore, optimizing these quadratic models can provide approximated solutions for the local minima of a given objective function. The optimizers of the quadratic models can be given by Equation (3), as explained before.

Practically, it is very costly to compute the inverse of a full matrix A in Equation (3), especially for high dimension problems. Hence, using a reasonable relaxation form of this coefficient matrix

is crucial. Thus, as a main step in the proposed method, A is approximated as a diagonal matrix. Therefore, the QCGA individuals are coded in the quadratic models as:

$$w = (w_1, w_2, \ldots, w_n, w_{n+1}, \ldots, w_{2n}, w_{2n+1}), \tag{5}$$

where $w_i, i = 1, \ldots, n$, are equal to the diagonal entities of A, $w_i, i = n+1, \ldots, 2n$, are equal to the entities of b, and $w_{2n+1} = c$. To avoid the non-singularity of A, the entities $w_i, i = 1, \ldots, n$, are initialized and updated to satisfy the conditions $|x_i| \geq \epsilon, i = 1, \ldots, n$, for some predefined small real number ϵ. The individuals fitness values are computed as $f(x^*)$, where $f(\cdot)$ is the given objective function and x^* is calculated as:

$$x^* = -\frac{1}{2}\left(\frac{w_{n+1}}{w_1}, \frac{w_{n+2}}{w_2}, \ldots, \frac{w_{2n}}{w_n}\right). \tag{6}$$

The algorithmic scenario of QCGA is shown in Figure 3. In each iteration of the QCGA algorithm, an intermediate parent population is generated using the linear ranking selection mechanism [66]. Then, the arithmetical crossover and uniform mutation operators [67] are applied to reproduce the offspring. After getting the children, the survival selection is applied to select the survival members based on the members' elitism. Finally, the QCGA uses local search methods of Nelder–Mead [68] and the heuristic descent method [18] as a final intensification step to refine the best solutions found so far.

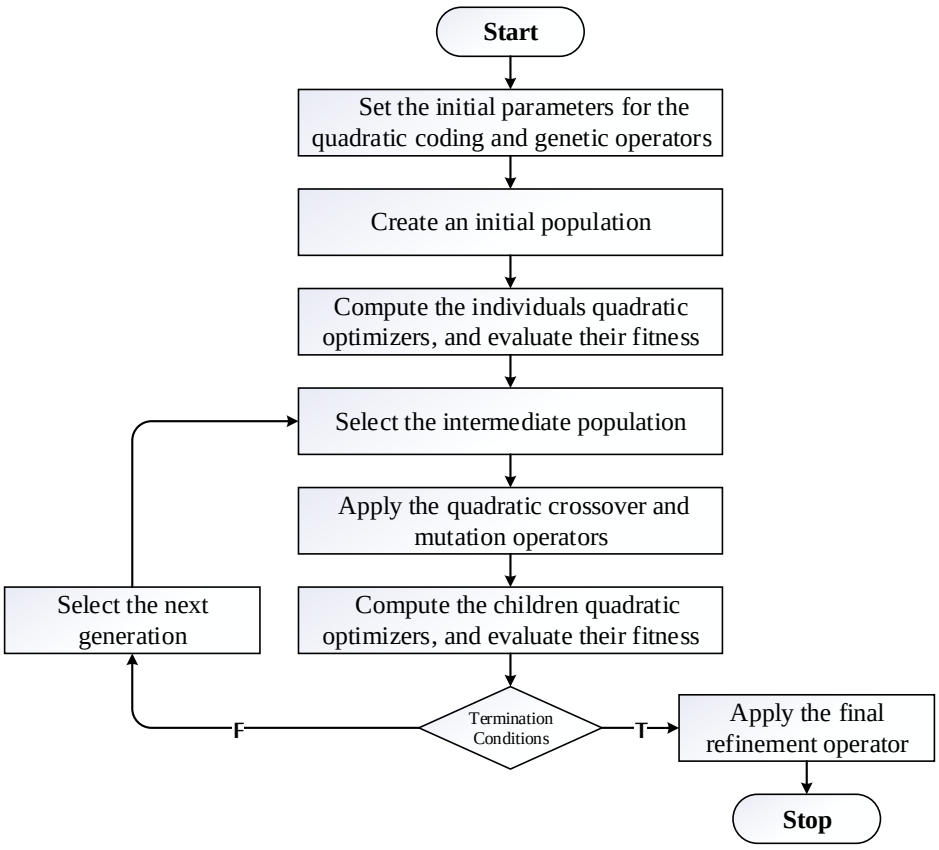

Figure 3. The QCGA Flowchart.

5. Sensing Evolution Strategies

The difficulty in solving global optimization problems arises from the challenge of searching a vast variable space to locate an optimal point, or at least space of points, with appropriate solution quality. It becomes even more challenging when the appropriate space is minimal compared to the complex search space. However, the quality of the initial solutions may impact the performance of the algorithm.

Thus, various global optimization techniques which exploit memory concept in different ways to help in both finding the optimal solution in minimum time and reduce the problem complexity are proposed. Generally, memory concept is used in various research for the sake of assisting exploration and exploitation processes. In [40], a directing strategy based on new search memories; the Gene Matrix (*GM*) and Landscape Matrix (*LM*) is proposed. That strategy can provide the search with new diverse solutions by applying a new type of mutation called "mutagenesis" [39]. The mutagenesis operator is used to change selected survival individuals to accelerate the exploration and exploitation processes. However, the *GM* and *LM* are also used to help the search process to remember how far the exploration process has gone to judge a termination point. In [15,16], tabu search algorithms are used with memory concept in high-dimensional problems to explore the region around an iterate solution more precisely.

The main structure of the proposed Sensing Evolution Strategies (SES) is shown in Figure 4. In the following, full details of the SES method and its algorithm steps are described. The SES method starts with a population of μ real-coded chromosomes. The mutated offspring is generated as in the typical ESs [69,70]. Therefore, a mutated offspring $(\tilde{x}, \tilde{\sigma})$ is obtained from a parent $(\hat{x}, \hat{\sigma})$, where the i-th component of the mutated offspring $(\tilde{x}, \tilde{\sigma})$ is given as:

$$\tilde{\sigma}_i = \hat{\sigma}_i \, e^{\tau' N(0,1) + \tau N_i(0,1)}, \tag{7}$$

$$\tilde{x}_i = \hat{x}_i + \tilde{\sigma}_i N_i(0,1), \tag{8}$$

where $\tau' \propto 1/\sqrt{2n}$ and $\tau \propto 1/\sqrt{2\sqrt{n}}$, and the proportional coefficients are usually set to one. The mutated offspring can be also computed from recombined parent as given in Procedure 3.

Procedure 3. Recombination$(p_1, \ldots, p_\rho)$

1. *If* $\rho > n$ *then return.*
2. *Partition each parent into* ρ *partitions at the same positions, i.e.,* $p_j = [X_1^j \; X_2^j \; \ldots \; X_\rho^j]$, $j = 1, \ldots, \rho$.
3. *Order the set* $\{1, 2, \ldots, \rho\}$ *randomly to be* $\{o_1, o_2, \ldots, o_\rho\}$.
4. *Calculate the recombined child* $x = [X_1^{o_1} \; X_2^{o_2} \; \ldots \; X_\rho^{o_\rho}]$, *and return.*

Adding sensing features to the proposed SES method makes it behave smarter. This can be achieved by generating and analyze search data and memory elements. As a main search memory for the SES, the *GM* is invoked as it has shown promising performance in global optimization [1,39,40]. In *GM*, each x in the search space consists of n genes or variables to get a real-coding representation of each individual in x. First, the range value of each gene is divided into m sub-range to check the diversity of the gene values. Thus, the *GM* is initialized to be a $n \times m$ zero matrix in which each entry of the i-th row refers to a sub-space of the i-th gene, from which *GM* is zeros, and ones matrix and its entries are changed from zeros to ones if new genes are generated within the corresponding sub-spaces during the searching process, as shown in Figure 5. This process can be continued until all entries become ones, i.e., with no zero entry in the *GM*. At that point, when the *GM* is considered full of ones, the search process achieves an advanced exploration process and can be stopped. Therefore, the *GM* is used to provide the search process with efficient termination criteria, and it can help by providing the search process with various diverse solutions.

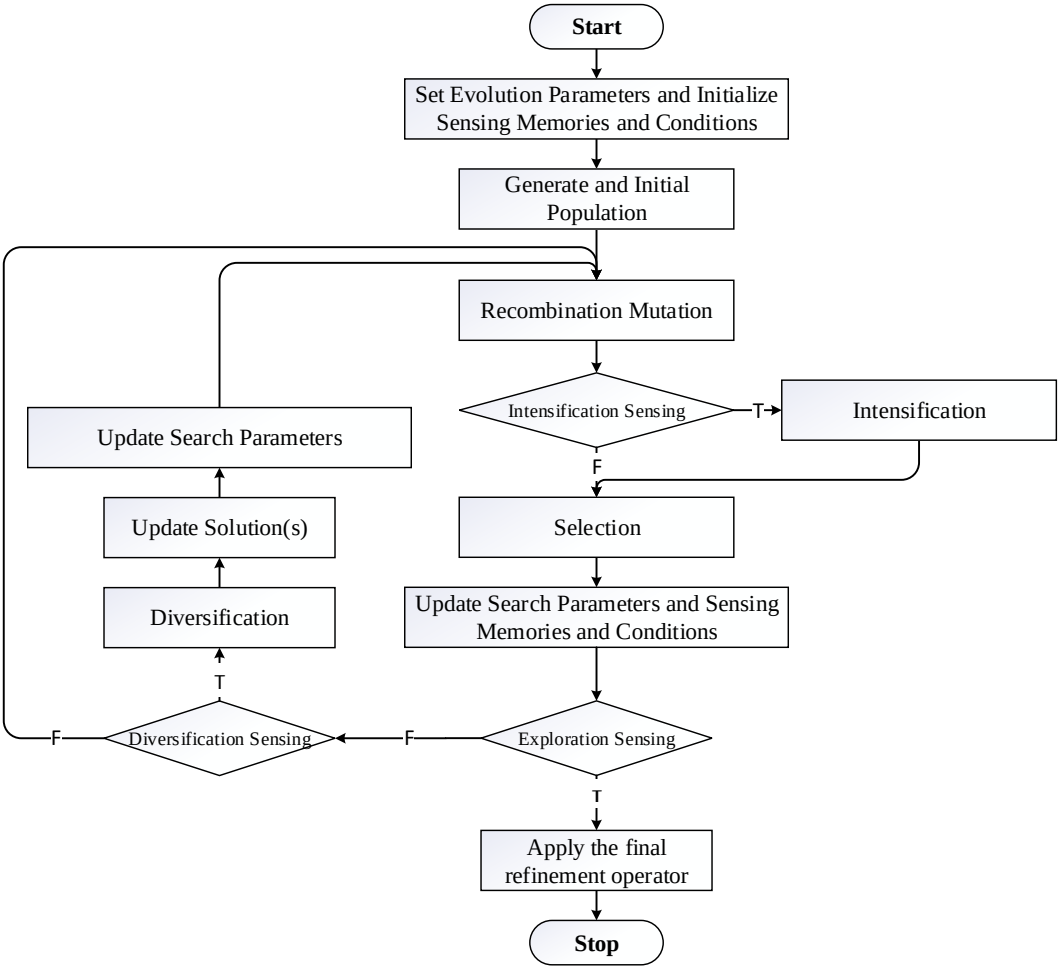

Figure 4. A flowchart for the Sensing Evolution Strategies with Quadratic Search (SESQS).

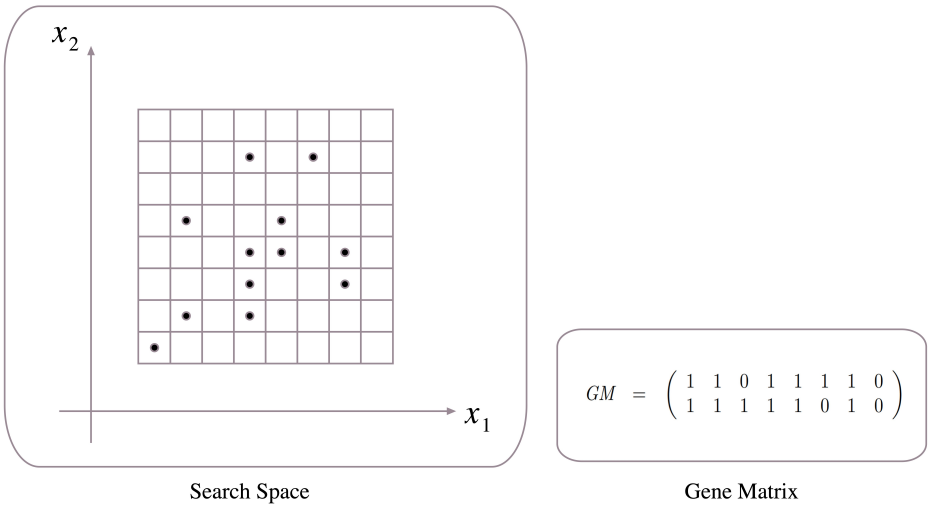

Figure 5. An Example of the *GM* in two dimensions.

Three sensing features have been invoked in the SES method as explained in the following:

- **Intensification Sensing.** This feature seeks to detect promising search regions that have been recently visited. Then, a local fast search can be applied to explore such promising regions deeply. The SES method uses quadratic search models as stated in Procedures 1 and 2 to obtain

approximate premising solutions using Equation (4). Moreover, a promising region can be detected when the generated mutated children are adapted to be close. Then, a quadratic model can be generated to approximate the fitness function in that close region surrounding those mutated children.

- **Diversification Sensing.** This feature aims to avoid entrapment in local solutions and recognize the need for generating new diverse solutions. The SES method checks the improvement rates while the search is going on. Whenever significant non-improvement has been faced, then the mutagenesis operation (Procedure 4) is called to generate new diverse solutions. In that procedure, new solutions are created by generating their gene values within sub-ranges whose corresponding indicators are zeros in the *GM* as shown in Figure 6. Those diverse solutions replace some worst solutions in the current population.
- **Exploration Sensing.** This feature tries to recognize the progress of the search exploration process and to detect an appropriate time for termination whenever a wide exploration process is achieved. When a full *GM* is reached, then the SES method has learned that the exploration method is almost over.

The mutagenesis operation starts with selecting a zero-position in the *GM*, which is corresponding to a zero-visit partition. Selecting such *GM*-entry is done randomly. Then, a new value of the gene related to the selected *GM*-entry is generated within the corresponding partition of the selection, as illustrated in Figure 6. The formal description of this mechanism is stated in Procedure 4.

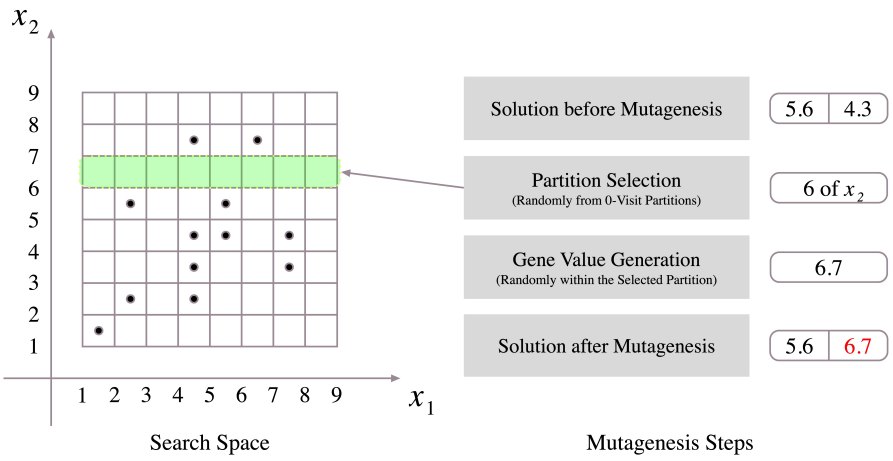

Figure 6. An Example of the mutagenesis steps in two dimensions.

Procedure 4. Mutagenesis(x, GM)

1. *If* GM *is full then return.*
2. *Choose a zero-position* (i, j) *in* GM *randomly.*
3. *Update* x *by setting* $x_i = l_i + (j - r)\frac{u_i - l_i}{m}$, *where* r *is a random number from* $(0, 1)$, *and* l_i, u_i *are the lower and upper bound of the variable* x_i, *respectively.*
4. *Update* GM *by setting* $\mathrm{GM}_{ij} = 1$, *and return.*

The formal description of the proposed SES method is given in the following Algorithm 5.

Algorithm 5. *SES Algorithm*

1. ***Initialization.*** *Create an initial population* $P_0 = \{(x^i, \sigma^i, F(x^i)),\ i = 1, \ldots, \mu\}$, *and initialize the* GM. *Choose the recombination probability* $p_r \in [0, 1]$, *set values of;* ρ, N_w *and* N_{elitet}, *set* $\nu := \lambda/\mu$, *and set the generation counter* $g := 0$.

2. ***Main Loop.*** *Repeat the following steps (2.1) – (2.4) from* $j = 1, \ldots, \mu$.

    2.1 ***Recombination.*** *Generate a random number* $\chi \in [0, 1]$. *If* $\chi > p_r$, *choose a parent* $(\hat{x}^j, \hat{\sigma}^j)$ *from* P_g *and go to Step 2.2. Otherwise, choose* ρ *parents* $p^1, \ldots, p^\rho$ *from* P_g, *and calculate the recombined child* $(\hat{x}^j, \hat{\sigma}^j)$ *using Procedure 3.*

    2.2 ***Mutation.*** *Use the individual* $(\hat{x}^j, \hat{\sigma}^j)$ *to calculate the mutated children* $(\tilde{x}^{j,k}, \tilde{\sigma}^{j,k})$, $k = 1, \ldots, \nu$, *as in Equations (7) and (8).*

    2.3 ***Fitness.*** *Evaluate the fitness function* $\tilde{F}^{j,k} = F(\tilde{x}^{j,k})$, $k = 1, \ldots, \nu$.

    2.4 ***Intensification Sensing.*** *If the individual* $(\hat{x}^j, \hat{\sigma}^j)$ *and his mutated children* $(\tilde{x}^{j,k}, \tilde{\sigma}^{j,k})$, $k = 1, \ldots, \nu$, *are close enough to be approximated by a quadratic model, the apply Procedure 1 or 2 to get an improved point. Replace the worst mutated child with the generated point by the quadratic model if the latter is better.*

3. ***Children Gathering.*** *Collect all generated children in* C_g *containing all* $(\tilde{x}^{j,k}, \tilde{\sigma}^{j,k}, \tilde{F}^{j,k})$, $j = 1, \ldots, \mu$, $k = 1, \ldots, \nu$.

4. ***Update Search Parameters.*** *Update the* GM.

5. ***Exploration Sensing.*** *If the* GM *is full, then go to Step 8.*

6. ***Selection.*** *Choose the best* μ *individuals from* $P_g \cup C_g$ *to compose the next generation* P_{g+1}. *Update the gene matrix* GM.

7. ***Diversification Sensing.*** *If the new population needs new diverse solutions, then apply Procedure 4 to alter the* N_w *worst individuals in* P_{g+1}. *Set* $g := g + 1$, *and go to Step 2.*

8. ***Intensification.*** *Apply a local search method starting from each solution from the* N_{elite} *best ones obtained in the previous search stage.*

The non-convex global optimization problem, defined in Equation (1), is an *NP*-complete problem [71]. Therefore, there is no efficient algorithm to solve such problem in its general form. Metaheuristics are practical solvers for this problem which are generally polynomial algorithms [72,73]. The proposed methods; QCGA and SES, follow the main structures of standard EAs with additive procedures which are at most of order $O(n^3)$.

6. Experimental Results

In the following, the experimental results are discussed. All the proposed methods are programmed using MATLAB. The parameters values used in QCGA and SES algorithms are set based on the typical setting in the literature or determined through our preliminary numerical experiments, as shown in Tables 1 and 2. Two classes of benchmark test functions have been invoked in the experimental results to discuss the efficiency of the proposed methods. The first class of benchmark functions contains 13 classical test functions f_1–f_{13} [16]. Those functions definitions are given in Appendix A. The other class of benchmark functions contains 25 hard test functions h_1–h_{25} [74,75] which are described in Appendix B.

Table 1. QCGA parameters.

Parameter	Definition	Value
μ	The population size	50
p_r	The crossover probability	0.25
p_m	The mutation probability	0.05
m	No. of partitions in *GM*	50
N_{elite}	No. of best solutions used in in the intensification step	1
N_w	No. of worst solutions updated by the mutagenesis process	n

Table 2. SES parameters.

Parameter	Definition	Value
μ	The population size	30
λ	The offspring size	10μ
ρ	No. of mated parents	$\min(n, 5)$
m	No. of partitions in GM	50
p_r	The recombination probability	0.25
N_{elite}	No. of best solutions used in in the intensification step	1
N_w	No. of worst solutions updated by the mutagenesis process	n

In the following, a highlight of the performance analysis on the quadratic coding and quadratic search operators before discussing the whole proposed algorithmic performance of the proposed methods is given.

6.1. Quadratic Coding

The main numerical results are shown in Figure 7 and Table 3. Table 3 shows the average values of best solutions (Avg. $f(x^{best})$), the average numbers of function evaluations (Cost), and the success rates of reaching close to the global optima. We compare the results of the QCGA method with the standard genetic algorithm (GA), and one of the recent genetic algorithm versions called the Adaptive Genetic Algorithm (AGA) [9]. For the QCGA and GA results, there are obtained over 100 independent runs. The AGA results are taken from their original reference, which is averaged over only 30 independent runs. The results in Table 3 and Figure 7 show that the proposed method could improve the performance of the genetic algorithm and could reach faster convergence. Moreover, the proposed method could be competitive with the recent advanced GA versions.

Table 3. Solutions qualities and costs of the proposed and compared methods.

	QCGA			GA			AGA	
f	Avg. $f(x^{best})$	Cost	Rate	Avg. $f(x^{best})$	Cost	Rate	Avg. $f(x^{best})$	Cost
f_1	5.21×10^{-16}	34,931	100%	2.04×10^{-12}	34,646	100%	1.10×10^{-43}	40,000
f_2	3.54×10^{-5}	41,734	100%	7.62×10^{-5}	42,252	98%	2.04×10^{-31}	40,000
f_3	1.79×10^{-9}	38,514	100%	1.46×10^{-8}	38,969	100%	1.21×10^{-4}	40,000
f_4	8.21×10^{-2}	38,951	1%	2.35	41,345	0%	$1.71 \times 10^{+1}$	40,000
f_5	8.04×10^{-11}	40,678	100%	5.58×10^{-1}	39,408	86%	2.70×10^{-11}	40,000
f_6	0	34,843	100%	$4.50 \times 10^{+1}$	34,939	0%	–	–
f_7	9.32×10^{-6}	35,816	100%	2.05	35,194	0%	7.64×10^{-15}	40,000

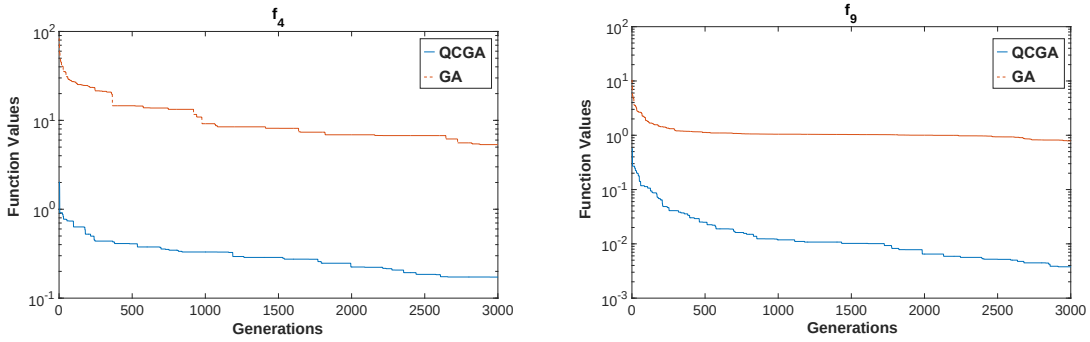

Figure 7. Examples of QCGA and GA performance using f_4 and f_9 test functions.

6.2. Quadratic Models

The performance of the quadratic regression model is tested to show how it can help the EAs to find new, improved solutions. Figure 8 shows the success rates of obtaining improved solutions

using the proposed quadratic models. Two test functions; f_5 and f_{11} (shown in Appendix A), are invoked in this experiment with different dimensions of 2, 5, 10, 20, 30 and 50. The solutions used in this experiment are generated to be close to local or global minima, and also to be far from them. Results show that the success rates of obtaining improved solutions are promising, especially in high dimensions. However, the results shown in Figure 8 illustrate that the performance of this quadratic operation near minima is better than that far from them. Compared to regression models, the RBF is almost working in the same manner. However, the RBF success rate of obtaining improved solutions depends on the type of function since the performance of the quadratic operation at f_5 is better than that at f_{11}. Finally, we may conclude that the proposed quadratic models are promising intensification procedures especially in the vicinity of the local and global minima.

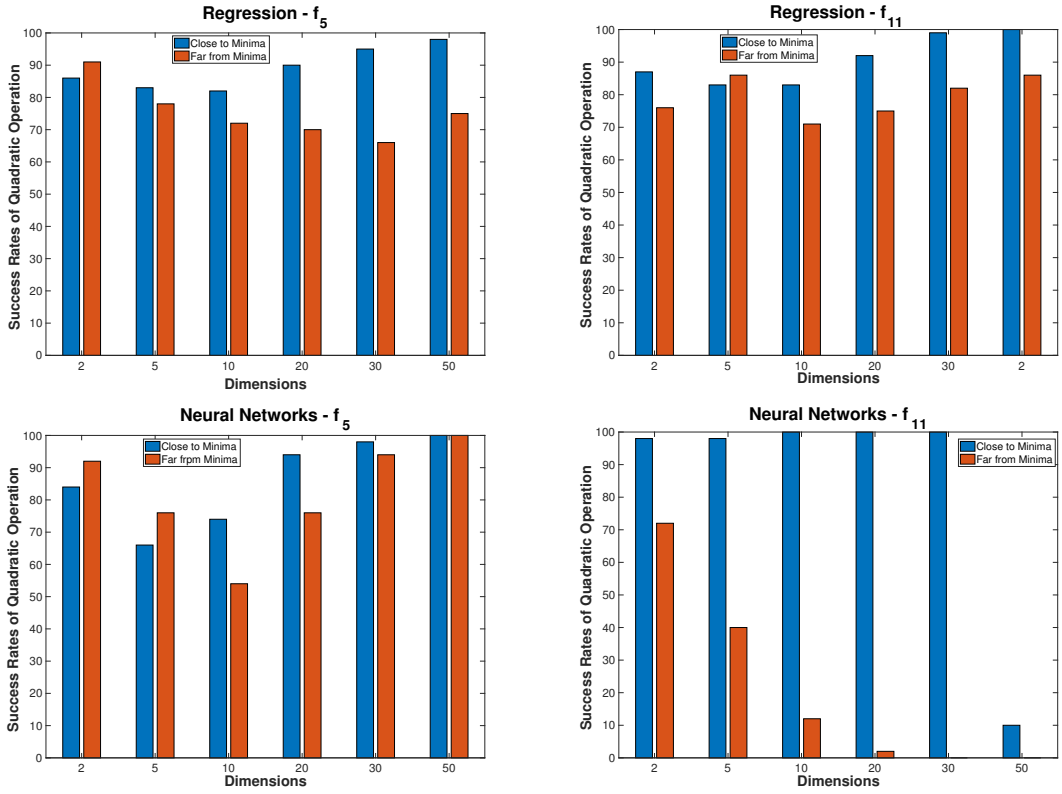

Figure 8. Performance of the quadratic operations near or far from minima.

6.3. Termination Criteria and Processing Time

The automatic termination using the *GM* is illustrated in Figure 9 using test functions f_1, f_3, f_5 and f_7. The QCGA and SES_R codes were continued running after reaching the point of having a full *GM* to check any further improvement after that. It is worthwhile to mention that there almost no improvement after reaching a full *GM* as shown in Figure 9. Therefore, the automatic termination using the *GM* could help EAs to avoid unnecessary computations.

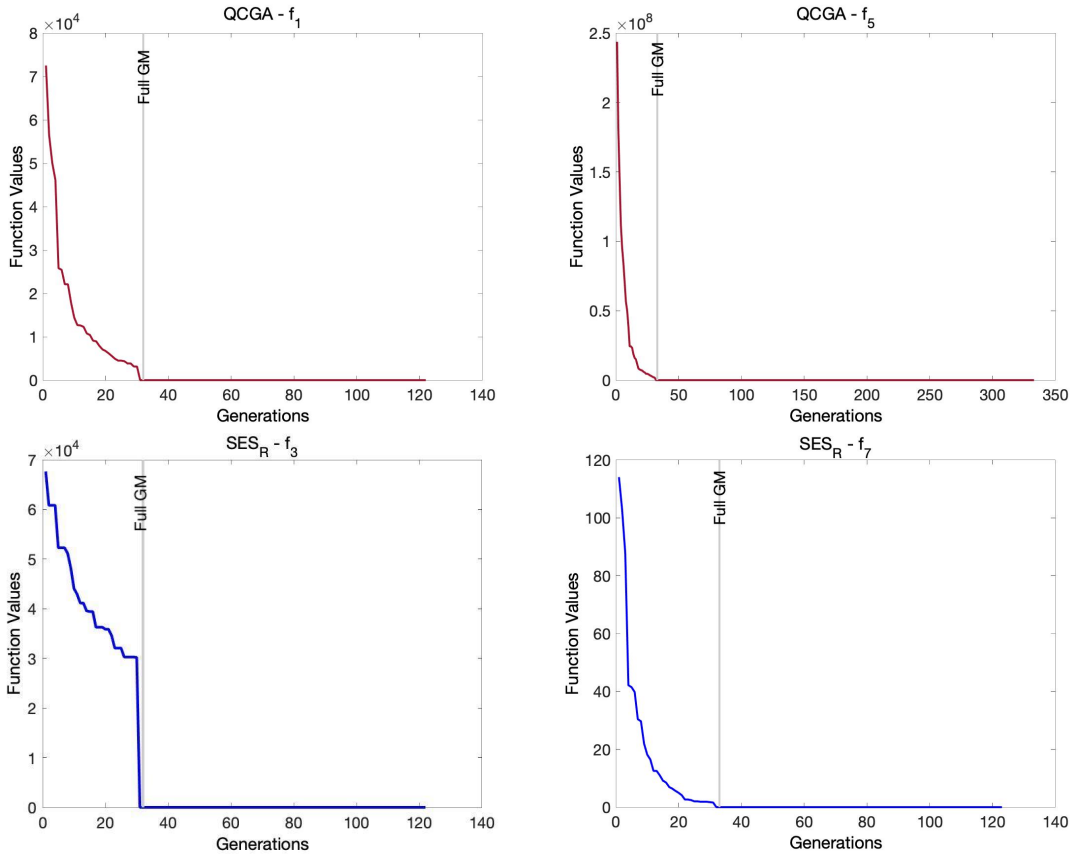

Figure 9. *GM* termination performance.

The runtime of the proposed methods are computed and presented in Figures 10 and 11 which is reported over 25 independent runs. Figure 10 shows the processing time for running the QCGA and the standard GA. The difference between the processing times of the two methods is reasonable, although the QCGA uses additional search procedures more than the standard GA. The processing time for running the all proposed methods are presented in Figure 11. The processing times of these methods are close to each other. However, one can note that the processing times of using the ANN models are greater than those of using the regression models.

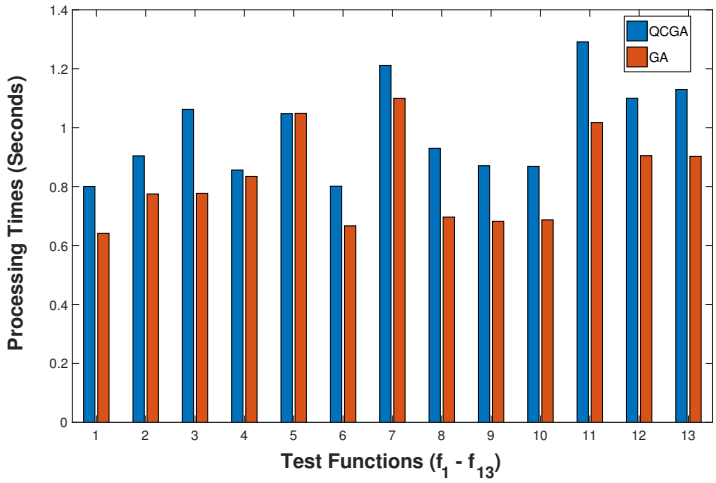

Figure 10. Processing times for the QCGA and GA codes using f_1–f_{13} test functions.

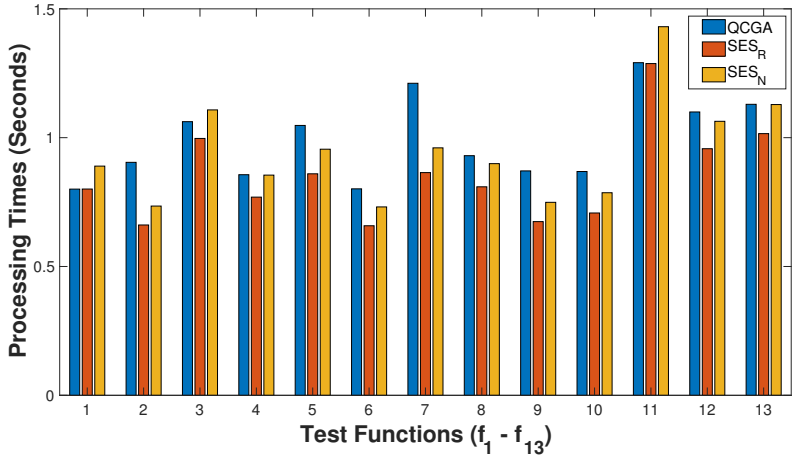

Figure 11. Processing times for proposed methods codes using f_1–f_{13} test functions.

6.4. Global Search Results

Table 4 shows a comparison between various methods such as regression SES_R, RBFs SES_N, and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [76]. The results are not as fair as the termination policies in both methods are different. Thus, the results show that the regression method is more promoting over the RBFs due to the mentioned reason of termination criteria. The results also reveal that the QCGA algorithm has steadily better performance than the ESLAT and CMA-ES in terms of the success rate. The solution costs in Table 5 show the number of iterations that is consumed to get each relevant result. RBFs consume more time in all function iterations than the regression method, while the QCGA consumes less time compared with the ESLAT algorithm.

Table 4. Solutions qualities—average errors and success rates.

f	SES_R	SES_N	QCGA	ESLAT	CMA-ES
f_1	2.8×10^{-19} (100%)	1.7×10^{-17} (100%)	5.2×10^{-16} (100%)	2.0×10^{-17} (100%)	9.7×10^{-23} (100%)
f_2	5.4×10^{-07} (100%)	3.2×10^{-05} (100%)	3.5×10^{-05} (100%)	3.8×10^{-05} (100%)	4.2×10^{-11} (100%)
f_3	8.6×10^{-08} (100%)	5.1×10^{-06} (100%)	1.8×10^{-09} (100%)	6.1×10^{-06} (100%)	7.1×10^{-23} (100%)
f_4	1.1×10^{-01} (7%)	6.5×10^{-01} (5%)	8.2×10^{-02} (1%)	7.8×10^{-01} (0%)	5.4×10^{-12} (100%)
f_5	2.7×10^{-01} (78%)	1.6×10^{-00} (75%)	8.0×10^{-11} (100%)	1.9×10^{-00} (70%)	4.0×10^{-01} (90%)
f_6	2.8×10^{-04} (100%)	1.7×10^{-02} (100%)	0.0×10^{-00} (100%)	2.0×10^{-02} (98%)	1.4×10^{-02} (36%)
f_7	5.5×10^{-03} (8%)	3.3×10^{-01} (5%)	9.3×10^{-06} (100%)	3.9×10^{-01} (0%)	2.3×10^{-01} (0%)
f_8	-2.2×10^{15} (0%)	-2.1×10^{15} (0%)	-2.80×10^{38} (0%)	-2.3×10^{15} (0%)	-7.6×10^{03} (0%)
f_9	6.6×10^{-02} (48%)	3.9×10^{-00} (45%)	3.0×10^{-00} (0%)	4.7×10^{-00} (40%)	5.2×10^{01} (0%)
f_{10}	2.5×10^{-10} (100%)	1.5×10^{-08} (100%)	9.9×10^{-06} (100%)	1.8×10^{-08} (100%)	6.9×10^{-12} (100%)
f_{11}	2.0×10^{-05} (97%)	1.2×10^{-03} (95%)	1.7×10^{-11} (100%)	1.4×10^{-03} (90%)	7.4×10^{-04} (92%)
f_{12}	2.1×10^{-14} (100%)	1.3×10^{-12} (100%)	9.0×10^{-12} (100%)	1.5×10^{-12} (100%)	1.2×10^{-04} (88%)
f_{13}	9.1×10^{-05} (71%)	5.4×10^{-03} (65%)	5.5×10^{-02} (45%)	6.4×10^{-03} (60%)	1.7×10^{-03} (86%)

Table 5. Solutions costs (Numbers of function evaluations).

f	SES_R	SES_N	QCGA	ESLAT	CMA-ES
f_1	34,944	38,827	34,931	69,724	10,721
f_2	30,512	33,902	41,734	60,859	12,145
f_3	36,153	40,169	38,514	72,141	21,248
f_4	34,993	38,880	38,951	69,821	20,813
f_5	33,387	37,096	40,678	66,609	55,821
f_6	28,614	31,793	34,843	57,064	2,184
f_7	25,563	28,403	35,816	50,962	667,131
f_8	30,934	34,371	35,552	61,704	6,621
f_9	27,022	30,024	34,911	53,880	10,079
f_{10}	29,537	32,818	36,256	58,909	10,654
f_{11}	35,604	39,560	35,699	71,044	10,522
f_{12}	31,597	35,108	36,305	63,030	13,981
f_{13}	32,910	36,566	36,596	65,655	13,756

The Wilcoxon rank-sum test [77–79] is applied to measure the performance of all the compared methods. This test is classified as a non-parametric test, which is endorsed in our experiments [80,81]. A comparison between the Wilcoxon test's results and the obtained results from our proposed methods are illustrated in Table 6. The table shows the sum of rankings obtained in each comparison and the p-value associated. Typically, the difference d_i between the performance values of the two methods on i-th out of N results are calculated. Afterwards, based on the absolute values of these differences, they are ranked. The ranks R^+ and R^- are computed as follows:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i),$$
$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i).$$

The equality hypothesis is based on rejecting in case of having the $\min(R^+, R^-)$ is less than or equal to the value of the distribution of Wilcoxon for N degrees of freedom (Table B.12 in [79]). The best method is SES_R, as shown in Table 6 according to the best solutions criteria, while the comparison based on the success rate criteria shows that SES_R is better than the SES_N and ESLAT algorithms. Also, QCGA has an excellent performance better than SES_N and ESLAT. The third comparison criteria, solutions costs, proves that SES_R performs better than all the compared methods except the CMA-ES.

Table 6. Wilcoxon rank-sum test for the results of Tables 4 and 5.

Comparison Criteria	Compared Methods	R^+	R^-	p-Value	Best Method
Best Solutions	SES_R, SES_N	13	78	0.0001	SES_R
	SES_R, QCGA	13	78	0.0001	SES_R
	SES_R, ESLAT	13.5	77.5	0.0003	SES_R
	SES_R, CMA-ES	13	78	0.0001	SES_R
	SES_N, QCGA	0	91	0.3560	-
	SES_N, ESLAT	56	35	0.4418	-
	SES_N, CMA-ES	13	78	0.3560	-
	QCGA, ESLAT	72	19	0.2382	-
	QCGA, CMA-ES	13	78	0.7976	-
	ESLAT, CMA-ES	16	75	0.2184	-
Success Rates	SES_R, SES_N	90.5	0.5	0	SES_R
	SES_R, QCGA	77	14	0.8501	-
	SES_R, ESLAT	90.5	0.5	0	SES_R
	SES_R, CMA-ES	46	45	0.5681	-
	SES_N, QCGA	0.5	90.5	0	QCGA
	SES_N, ESLAT	45.5	45.5	NaN	-
	SES_N, CMA-ES	1.5	89.5	0	CMA-ES
	QCGA, ESLAT	90.5	0.5	0	QCGA
	QCGA, CMA-ES	44	47	0.5488	-
	ESLAT, CMA-ES	1.5	89.5	0	CMA-ES
Solutions Costs	SES_R, SES_N	0	91	0.0313	SES_R
	SES_R, QCGA	1	90	0.0005	SES_R
	SES_R, ESLAT	0	91	0	SES_R
	SES_R, CMA-ES	70	21	0.0029	CMA-ES
	SES_N, QCGA	24	67	0.2592	-
	SES_N, ESLAT	0	91	0	SES_N
	SES_N, CMA-ES	76	15	0.0029	CMA-ES
	QCGA, ESLAT	0	91	0	QCGA
	QCGA, CMA-ES	77	14	0.0029	CMA-ES
	ESLAT, CMA-ES	78	13	0.0004	CMA-ES

6.5. Results for Hard Test Functions

There are many suits of test functions, but one of the most common is the CEC2005 suit which contains 25 functions [74,75], see Appendix B. This benchmark suit includes unimodal functions (h_1–h_5), basic multimodal functions (h_6–h_{12}), extended multimodal functions (h_{13}–h_{14}), and hybrid composition functions (h_{15}–h_{25}). These functions are composed from some classical functions with higher degrees of difficulties like function shifting, rotating and combining.

The proposed QCGA method is applied to these hard test functions with dimensions 10, 30 and 50, as reported in Tables 7–9. The reported results in these tables are global minima, the mean and standard deviation of best objective values obtained by the algorithm as well as the best and the worst value, the mean and standard deviation of the errors between the obtained objective values and the optimal ones, the mean of function evaluations, and the success rate of reaching the global minima within a gap of 10^{-3}. All these results are obtained over 25 independent runs of the QCGA code. The results show that the method could reach the global minima for some functions and close to them for others. The best objective function values obtained by the method are relatively close to the worst ones except to functions h_4 and h_5.

Table 7. The QCGA results of hard functions h_1–h_{25} with dimension $n = 10$.

		Function Values				Errors		Evals.	Success
h	$h^\star$	Mean	Std	Best	Worst	Mean	Std	Mean	(%)
h_1	−450	−450.0	2.15×10^{-12}	−450.0	−450.0	2.37×10^{-12}	2.15×10^{-12}	58,024	100
h_2	−450	−450.0	1.31×10^{-9}	−450.0	−450.0	1.59×10^{-9}	1.31×10^{-9}	58,567	100
h_3	−450	−450.0	1.72×10^{-2}	−450.0	−449.9	5.90×10^{-3}	1.72×10^{-2}	59,838	64
h_4	−450	8030.5	1.82×10^{3}	3327.4	10812.9	8.48×10^{3}	1.82×10^{3}	57,652	0
h_5	−310	1960.4	1.43×10^{3}	293.2	5634.4	2.27×10^{3}	1.43×10^{3}	59,736	0
h_6	390	390.8	1.63×10^{0}	390.0	394.0	7.97×10^{-1}	1.63×10^{0}	59,917	80
h_7	−180	−179.0	8.20×10^{-1}	−180.0	−177.1	9.62×10^{-1}	8.20×10^{-1}	58,943	0
h_8	−140	−120.0	1.86×10^{-4}	−120.0	−120.0	2.00×10^{1}	1.86×10^{-4}	58,525	0
h_9	−330	−329.7	6.11×10^{-1}	−330.0	−328.0	2.79×10^{-1}	6.11×10^{-1}	57,897	80
h_{10}	−330	−272.1	1.43×10^{1}	−302.1	−247.4	5.79×10^{1}	1.43×10^{1}	57,918	0
h_{11}	90	97.4	1.15×10^{0}	95.0	99.3	7.38×10^{0}	1.15×10^{0}	57,793	0
h_{12}	−460	−138.4	6.45×10^{2}	−460.0	1233.6	3.22×10^{2}	6.45×10^{2}	58,240	36
h_{13}	−130	−129.8	1.45×10^{-1}	−130.0	−129.3	1.59×10^{-1}	1.45×10^{-1}	58,620	0
h_{14}	−300	−296.0	2.79×10^{-1}	−296.4	−295.5	3.98×10^{0}	2.79×10^{-1}	59,500	0
h_{15}	120	216.5	2.21×10^{2}	120.0	1020.0	9.65×10^{1}	2.21×10^{2}	57,595	80
h_{16}	120	361.2	4.66×10^{1}	270.9	458.7	2.41×10^{2}	4.66×10^{1}	57,745	0
h_{17}	120	365.7	4.82×10^{1}	254.4	452.3	2.46×10^{2}	4.82×10^{1}	57,698	0
h_{18}	10	910.0	5.30×10^{-9}	910.0	910.0	9.00×10^{2}	5.30×10^{-9}	57,792	0
h_{19}	10	910.0	1.68×10^{-3}	910.0	910.0	9.00×10^{2}	1.68×10^{-3}	57,765	0
h_{20}	10	918.9	4.45×10^{1}	910.0	1132.6	9.09×10^{2}	4.45×10^{1}	57,707	0
h_{21}	360	1616.7	1.64×10^{2}	860.0	1750.3	1.26×10^{3}	1.64×10^{2}	57,764	0
h_{22}	360	1273.5	5.28×10^{1}	1145.8	1388.8	9.13×10^{2}	5.28×10^{1}	57,700	0
h_{23}	360	1615.1	5.17×10^{1}	1523.7	1721.5	1.26×10^{3}	5.17×10^{1}	57,765	0
h_{24}	260	1361.1	3.63×10^{2}	460.0	1613.9	1.10×10^{3}	3.63×10^{2}	57,661	0
h_{25}	260	1383.9	3.29×10^{2}	460.0	1543.7	1.12×10^{3}	3.29×10^{2}	57,646	0

Table 8. The QCGA results of hard functions h_1–h_{25} with dimension $n = 30$.

		Function Values				Errors		Evals.	Success
h	$h^\star$	Mean	Std	Best	Worst	Mean	Std	Mean	(%)
h_1	−450	−450.0	6.23×10^{-10}	−450.0	−450.0	4.38×10^{-10}	6.23×10^{-10}	172,504	100
h_2	−450	−450.0	2.47×10^{-8}	−450.0	−450.0	1.03×10^{-7}	2.47×10^{-8}	177,167	100
h_3	−450	−439.6	2.23×10^{1}	−449.9	−399.8	1.04×10^{1}	2.23×10^{1}	180,218	0
h_4	−450	29718.1	1.59×10^{3}	27415.2	31616.6	3.02×10^{4}	1.59×10^{3}	172,917	0
h_5	−310	21183.2	3.03×10^{3}	18314.0	25710.7	2.15×10^{4}	3.03×10^{3}	173,964	0
h_6	390	875.9	4.24×10^{2}	409.1	1186.5	4.86×10^{2}	4.24×10^{2}	177,557	0
h_7	−180	−180.0	1.15×10^{-10}	−180.0	−180.0	9.86×10^{-7}	1.15×10^{-10}	177,086	100
h_8	−140	−120.0	4.85×10^{-6}	−120.0	−120.0	2.00×10^{1}	4.85×10^{-6}	177,544	0
h_9	−330	−237.3	1.75×10^{1}	−262.3	−218.6	9.27×10^{1}	1.75×10^{1}	172,591	0
h_{10}	−330	−84.2	2.28×10^{1}	−106.1	−52.4	2.46×10^{2}	2.28×10^{1}	172,977	0
h_{11}	90	126.9	2.68×10^{0}	123.1	129.4	3.69×10^{1}	2.68×10^{0}	173,114	0
h_{12}	−460	385.0	5.70×10^{2}	−68.3	1220.3	8.45×10^{2}	5.70×10^{2}	176,014	0
h_{13}	−130	−127.0	1.01×10^{0}	−128.3	−125.9	2.98×10^{0}	1.01×10^{0}	176,551	0
h_{14}	−300	−286.7	3.03×10^{-1}	−287.1	−286.3	1.03×10^{1}	3.03×10^{-1}	182,736	0
h_{15}	120	531.5	1.81×10^{1}	508.7	557.3	4.11×10^{2}	1.81×10^{1}	173,161	0
h_{16}	120	468.9	9.39×10^{1}	364.2	578.0	3.49×10^{2}	9.39×10^{1}	173,415	0
h_{17}	120	508.4	8.68×10^{1}	399.3	607.9	3.88×10^{2}	8.68×10^{1}	173,128	0
h_{18}	10	910.3	1.88×10^{-1}	910.1	910.6	9.00×10^{2}	1.88×10^{-1}	173,191	0
h_{19}	10	910.4	1.38×10^{-1}	910.2	910.6	9.00×10^{2}	1.38×10^{-1}	173,609	0
h_{20}	10	910.6	2.48×10^{-1}	910.4	911.0	9.01×10^{2}	2.48×10^{-1}	172,848	0
h_{21}	360	1609.3	1.04×10^{1}	1597.9	1621.6	1.25×10^{3}	1.04×10^{1}	173,131	0
h_{22}	360	1346.6	4.28×10^{1}	1182.1	1400.3	9.07×10^{2}	4.28×10^{1}	173,098	0
h_{23}	360	1609.9	9.24×10^{0}	1599.5	1621.6	1.25×10^{3}	9.24×10^{0}	173,375	0
h_{24}	260	1242.3	3.11×10^{2}	823.5	1557.1	9.82×10^{2}	3.11×10^{2}	173,035	0
h_{25}	260	1404.3	2.63×10^{2}	942.3	1558.8	1.14×10^{3}	2.63×10^{2}	172,817	0

Table 9. The QCGA results of hard functions h_1–h_{25} with dimension $n = 50$.

		Function Values				Errors		Evals.	Success
h	$h^\star$	Mean	Std	Best	Worst	Mean	Std	Mean	(%)
h_1	−450	−450.0	7.94×10^{-13}	−450.0	−450.0	8.75×10^{-12}	7.94×10^{-13}	290,585	100
h_2	−450	−450.0	1.82×10^{-7}	−450.0	−450.0	2.41×10^{-7}	1.82×10^{-7}	300,898	100
h_3	−450	−443.7	6.20×10^{0}	−449.8	−424.5	6.35×10^{0}	6.20×10^{0}	319,328	0
h_4	−450	65598	1.13×10^{4}	50984	100100	6.60×10^{4}	1.13×10^{4}	287,977	0
h_5	−310	22210	3.89×10^{3}	14404	29409	2.25×10^{4}	3.89×10^{3}	302,175	0
h_6	390	705.7	3.27×10^{2}	433.6	1243.2	3.16×10^{2}	3.27×10^{2}	306,740	0
h_7	−180	−180.0	1.47×10^{-10}	−180.0	−180.0	4.11×10^{-10}	1.47×10^{-10}	300,342	100
h_8	−140	−120.0	9.89×10^{-7}	−120.0	−120.0	2.00×10^{1}	9.89×10^{-7}	298,303	0
h_9	−330	−106.3	2.25×10^{1}	−161.9	−64.3	2.24×10^{2}	2.25×10^{1}	289,411	0
h_{10}	−330	64.9	6.03×10^{1}	−47.4	187.4	3.95×10^{2}	6.03×10^{1}	289,837	0
h_{11}	90	155.4	3.25×10^{0}	148.5	161.4	6.54×10^{1}	3.25×10^{0}	291,468	0
h_{12}	−460	5489	6.17×10^{3}	−455.0	17190	5.95×10^{3}	6.17×10^{3}	298,225	0
h_{13}	−130	−116.2	3.74×10^{0}	−120.8	−105.5	1.38×10^{1}	3.74×10^{0}	299,730	0
h_{14}	−300	−277.2	3.19×10^{-1}	−277.9	−276.7	2.21×10^{1}	3.19×10^{-1}	320,024	0
h_{15}	120	705.5	5.59×10^{1}	663.1	768.8	5.86×10^{2}	5.59×10^{1}	288,695	0
h_{16}	120	530.4	8.50×10^{1}	450.0	619.4	4.10×10^{2}	8.50×10^{1}	288,770	0
h_{17}	120	595.3	4.74×10^{1}	541.0	627.6	4.75×10^{2}	4.74×10^{1}	288,089	0
h_{18}	10	918.3	1.81×10^{0}	916.8	920.3	9.08×10^{2}	1.81×10^{0}	288,668	0
h_{19}	10	918.9	1.45×10^{0}	917.2	920.0	9.09×10^{2}	1.45×10^{0}	288,326	0
h_{20}	10	915.9	3.70×10^{0}	911.9	919.2	9.06×10^{2}	3.70×10^{0}	289,279	0
h_{21}	360	1621.9	1.50×10^{1}	1603.3	1644.7	1.26×10^{3}	1.50×10^{1}	288,564	0
h_{22}	360	1256.5	7.13×10^{1}	1187.9	1334.4	8.97×10^{2}	7.13×10^{1}	288,602	0
h_{23}	360	1633.1	2.26×10^{1}	1598.8	1657.2	1.27×10^{3}	2.26×10^{1}	288,786	0
h_{24}	260	1550.2	1.81×10^{1}	1526.5	1567.5	1.29×10^{3}	1.81×10^{1}	288,319	0
h_{25}	260	1558.3	9.61×10^{0}	1549.6	1573.4	1.30×10^{3}	9.61×10^{0}	288,425	0

To check the quality of the hard functions results displayed in Tables 8 and 9, we compare them with the following benchmark methods:

- Self-adaptive artificial bee colony based on the global best candidate (SABC-GB) method [82].
- Artificial bee colony (ABC/best/1) [83].

The compared results are reported in Tables 10 and 11, as well as Table 12 which shows the statistical tests for the compared results. Tables 10 and 11 show the mean and standard deviation values of the errors between the obtained objective values and the optimal ones for the hard test functions with dimensions 30 and 50. The results of the compared methods SABC-GB and ABC/best/1 are taken from their original references [82,83], respectively. The number of function evaluations used to terminate both of the SABC-GB and ABC/best/1 methods is 300,000 as mentioned in [82].

The best obtained error values among the compared methods in Tables 8 and 9 are marked with bold. Moreover, the numbers of times that each method succeeded in obtaining the best result compared to the others are mentioned at the bottom of each table. For the 30-dimension results in Table 8, the performance of SABC-GB and QCGA regarding obtaining the best errors is close. However, the proposed QCGA method is slightly better than SABC-GB for the 50-dimension results in Table 9.

Table 10. Comparing the QCGA method with other benchmark methods using the hard functions h_1–h_{25} with dimension $n = 30$.

	QCGA		ABC/best/1 [83]		SABC-GB [82]	
h	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
h_1	4.38×10^{-10}	6.23×10^{-10}	1.97×10^{-12}	1.93×10^{-13}	$\mathbf{5.68 \times 10^{-14}}$	0
h_2	$\mathbf{1.03 \times 10^{-7}}$	2.47×10^{-8}	3.55×10^{4}	1.96×10^{3}	1.01×10^{4}	6.19×10^{2}
h_3	$\mathbf{1.04 \times 10^{1}}$	2.23×10^{1}	5.28×10^{7}	9.41×10^{6}	1.08×10^{7}	3.23×10^{5}
h_4	$\mathbf{3.02 \times 10^{4}}$	1.59×10^{3}	5.06×10^{4}	5.02×10^{3}	3.19×10^{4}	4.67×10^{2}
h_5	2.15×10^{4}	3.03×10^{3}	9.34×10^{3}	5.34×10^{2}	$\mathbf{7.41 \times 10^{3}}$	7.94×10^{2}
h_6	4.86×10^{2}	4.24×10^{2}	1.86×10^{2}	4.11×10^{1}	$\mathbf{2.52 \times 10^{-1}}$	1.03×10^{-1}
h_7	$\mathbf{9.86 \times 10^{-7}}$	1.15×10^{-10}	4.70×10^{3}	5.15×10^{-2}	4.70×10^{3}	0.00
h_8	$\mathbf{2.00 \times 10^{1}}$	4.85×10^{-6}	2.09×10^{1}	1.10×10^{-2}	2.09×10^{1}	5.40×10^{-2}
h_9	9.27×10^{1}	1.75×10^{1}	2.03×10^{-12}	2.02×10^{-12}	$\mathbf{5.68 \times 10^{-14}}$	0
h_{10}	2.46×10^{2}	2.28×10^{1}	2.82×10^{2}	1.14×10^{1}	$\mathbf{1.34 \times 10^{2}}$	5.34
h_{11}	3.69×10^{1}	2.68	3.32×10^{1}	1.45	$\mathbf{2.50 \times 10^{1}}$	2.66×10^{-1}
h_{12}	$\mathbf{8.45 \times 10^{2}}$	5.70×10^{2}	8.82×10^{4}	1.61×10^{4}	1.05×10^{4}	2.05×10^{3}
h_{13}	2.98	1.01	7.59	7.45×10^{-1}	$\mathbf{7.39 \times 10^{-1}}$	1.94×10^{-1}
h_{14}	$\mathbf{1.03 \times 10^{1}}$	3.03×10^{-1}	1.33×10^{1}	3.39×10^{-2}	1.27×10^{1}	4.65×10^{-2}
h_{15}	4.11×10^{2}	1.81×10^{1}	3.29×10^{2}	1.45×10^{2}	$\mathbf{2.39 \times 10^{-3}}$	1.81×10^{-3}
h_{16}	3.49×10^{2}	9.39×10^{1}	3.17×10^{2}	1.11×10^{1}	$\mathbf{1.58 \times 10^{2}}$	2.51×10^{1}
h_{17}	3.88×10^{2}	8.68×10^{1}	4.25×10^{2}	8.83	$\mathbf{2.35 \times 10^{2}}$	3.37
h_{18}	$\mathbf{9.00 \times 10^{2}}$	1.88×10^{-1}	9.21×10^{2}	2.16	9.09×10^{2}	1.32
h_{19}	$\mathbf{9.00 \times 10^{2}}$	1.38×10^{-1}	9.21×10^{2}	1.73	9.10×10^{2}	1.13
h_{20}	$\mathbf{9.01 \times 10^{2}}$	2.48×10^{-1}	9.23×10^{2}	3.07	9.09×10^{2}	1.47
h_{21}	1.25×10^{3}	1.04×10^{1}	5.07×10^{2}	5.87×10^{-1}	$\mathbf{5.00 \times 10^{2}}$	0
h_{22}	$\mathbf{9.07 \times 10^{2}}$	4.28×10^{1}	1.04×10^{3}	1.60×10^{1}	9.38×10^{2}	2.68
h_{23}	1.25×10^{3}	9.24	6.12×10^{2}	2.13×10^{1}	$\mathbf{5.34 \times 10^{2}}$	0
h_{24}	9.82×10^{2}	3.11×10^{2}	1.07×10^{3}	3.25×10^{1}	$\mathbf{4.54 \times 10^{2}}$	3.59×10^{2}
h_{25}	$\mathbf{1.14 \times 10^{3}}$	2.63×10^{2}	1.68×10^{3}	5.07	1.64×10^{3}	0
Best	**12**		**0**		**13**	

Table 11. Comparing the QCGA method with other benchmark methods using the hard functions h_1–h_{25} with dimension $n = 50$.

	QCGA		ABC/best/1 [83]		SABC-GB [82]	
h	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
h_1	8.75×10^{-12}	7.94×10^{-13}	6.33×10^{-11}	1.10×10^{-11}	$\mathbf{2.27 \times 10^{-13}}$	4.64×10^{-14}
h_2	$\mathbf{2.41 \times 10^{-7}}$	1.82×10^{-7}	1.06×10^{5}	1.22×10^{4}	6.02×10^{4}	3.79×10^{3}
h_3	**6.35**	6.20	1.84×10^{8}	4.77×10^{7}	5.85×10^{7}	9.96×10^{6}
h_4	$\mathbf{6.60 \times 10^{4}}$	1.13×10^{4}	1.31×10^{5}	7.70×10^{3}	1.30×10^{5}	6.23×10^{3}
h_5	$\mathbf{2.25 \times 10^{4}}$	3.89×10^{3}	2.47×10^{4}	1.20×10^{3}	2.37×10^{4}	1.26×10^{3}
h_6	3.16×10^{2}	3.27×10^{2}	3.24×10^{3}	1.33×10^{3}	**3.77**	1.91
h_7	$\mathbf{4.11 \times 10^{-10}}$	1.47×10^{-10}	6.22×10^{3}	1.85	6.20×10^{3}	0
h_8	$\mathbf{2.00 \times 10^{1}}$	9.89×10^{-7}	2.11×10^{1}	1.82×10^{-2}	2.10×10^{1}	0
h_9	2.24×10^{2}	2.25×10^{1}	2.97×10^{-12}	6.18×10^{-13}	$\mathbf{1.71 \times 10^{-13}}$	0
h_{10}	$\mathbf{3.95 \times 10^{2}}$	6.03×10^{1}	6.61×10^{2}	1.17×10^{1}	4.24×10^{2}	3.92×10^{1}
h_{11}	6.54×10^{1}	3.25	6.46×10^{1}	6.63×10^{-1}	$\mathbf{5.60 \times 10^{1}}$	1.61
h_{12}	$\mathbf{5.95 \times 10^{3}}$	6.17×10^{3}	4.69×10^{5}	4.79×10^{3}	5.78×10^{4}	1.28×10^{4}
h_{13}	1.38×10^{1}	3.74	1.82×10^{1}	1.90	**1.58**	2.61×10^{-1}
h_{14}	$\mathbf{2.21 \times 10^{1}}$	3.19×10^{-1}	2.31×10^{1}	2.34×10^{-1}	2.26×10^{1}	1.98×10^{-1}
h_{15}	5.86×10^{2}	5.59×10^{1}	3.17×10^{2}	8.96×10^{1}	$\mathbf{2.27 \times 10^{1}}$	9.77
h_{16}	4.10×10^{2}	8.50×10^{1}	4.27×10^{2}	1.37	$\mathbf{3.10 \times 10^{2}}$	3.58×10^{1}
h_{17}	4.75×10^{2}	4.74×10^{1}	7.56×10^{2}	3.98×10^{1}	$\mathbf{4.33 \times 10^{2}}$	1.93×10^{1}
h_{18}	$\mathbf{9.08 \times 10^{2}}$	1.81	9.71×10^{2}	9.33	9.39×10^{2}	4.18
h_{19}	$\mathbf{9.09 \times 10^{2}}$	1.45	9.68×10^{2}	1.30	9.37×10^{2}	6.34
h_{20}	$\mathbf{9.06 \times 10^{2}}$	3.70	9.75×10^{2}	3.36	9.35×10^{2}	3.88
h_{21}	1.26×10^{3}	1.50×10^{1}	1.04×10^{3}	1.16	$\mathbf{1.02 \times 10^{3}}$	5.59×10^{-1}
h_{22}	$\mathbf{8.97 \times 10^{2}}$	7.13×10^{1}	1.13×10^{3}	2.46×10^{1}	9.91×10^{2}	3.46×10^{1}
h_{23}	1.27×10^{3}	2.26×10^{1}	1.04×10^{3}	3.69	$\mathbf{1.02 \times 10^{3}}$	0
h_{24}	1.29×10^{3}	1.81×10^{1}	1.36×10^{3}	5.29	$\mathbf{1.08 \times 10^{3}}$	1.37×10^{1}
h_{25}	$\mathbf{1.30 \times 10^{3}}$	9.61	1.85×10^{3}	9.93	1.69×10^{3}	5.95
Best	**14**		**0**		**11**	

The Wilcoxon rank-sum test is applied to estimate the difference between the compared methods in terms of the obtained errors and the numbers of function evaluations. The statistical test outputs are reported in Table 12. These tests clarify that there is no significant difference between the compared methods in terms of the solution errors at the significant level of 0.05. However, it is clear that the QCGA could outperform the others in terms of reducing the computational costs of function evaluations.

Table 12. Wilcoxon rank-sum test for the hard functions results.

Comparison Criteria	**Compared Methods**	n	R^+	R^-	p**-Value**	**Best Method**
Solution Errors	QCGA, ABC/best/1 [83]	30	114	211	0.2687	–
	QCGA, SABC-GB [82]	30	165	160	0.6276	–
	QCGA, ABC/best/1 [83]	50	54	271	0.1031	–
	QCGA, SABC-GB [82]	50	126	199	0.3721	–
Numbers of Function Evaluations	QCGA, ABC/best/1 [83]	30	0	325	9.73×10^{-11}	QCGA
	QCGA, SABC-GB [82]	30	0	325	9.73×10^{-11}	QCGA
	QCGA, ABC/best/1 [83]	50	67	258	7.79×10^{-4}	QCGA
	QCGA, SABC-GB [82]	50	67	258	7.79×10^{-4}	QCGA

7. Conclusions

In this work, we propose a modified GA version called QCGA, which is based on applying a quadratic search operator to solve the global optimization problem. The use of quadratic coding of solutions efficiently assists the algorithm in achieving promising performance. Furthermore, other

quadratic-based techniques are presented as search operators to help the ESs in finding a global solution quickly. Moreover, the gene matrix memory element is invoked, which could guide the search process to new diverse solutions and help in terminating the algorithms. The efficiency of these methods is tested using several sets of benchmark test functions. The comparisons with some standard techniques in the literature indicate that the proposed methods are promising and show the efficiency of them shortening the computational costs. We may conclude that applying quadratic search operators can give them better performance and quick solutions. Therefore, we suggest the extension of the proposed ideas to modify the other metaheuristics. The quadratic search models can be invoked as local search procedures in many methods such as particle swarm optimization, ant colony optimization, and artificial immune systems. Moreover, the quadratic models and coding can also be used in response surface and surrogate optimization techniques. Finally, the proposed methods can also be extended to deal with constrained global optimization problems using suitable constraint-handling techniques.

Author Contributions: Conceptualization, A.-R.H., H.H.A. and M.A., W.D.; methodology, A.-R.H.,W.D., H.H.A. and M.A.; programming and implementation, A.-R.H., H.H.A. and M.A.; writing—original draft preparation, A.-R.H., W.D., H.H.A. and M.A.; writing—review and editing, A.-R.H.,W.D., H.H.A. and M.A.; funding acquisition, A.-R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Plan for Science, Technology, and Innovation (MAARIFAH), King Abdulaziz City for Science and Technology, the Kingdom of Saudi Arabia, award number (13-INF544-10).

Acknowledgments: The authors would like to thank King Abdulaziz City for Science and Technology, the Kingdom of Saudi Arabia, for supporting the project number (13-INF544-10).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Classical Test Functions

Appendix A.1. Sphere Function (f_1)

Definition: $f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$.
Search space: $-100 \leq x_i \leq 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_1(\mathbf{x}^*) = 0$.

Appendix A.2. Schwefel Function (f_2)

Definition: $f_2(\mathbf{x}) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$.
Search space: $-10 \leq x_i \leq 10,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_2(\mathbf{x}^*) = 0$.

Appendix A.3. Schwefel Function (f_3)

Definition: $f_3(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$.
Search space: $-100 \leq x_i \leq 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_3(\mathbf{x}^*) = 0$.

Appendix A.4. Schwefel Function (f_4)

Definition: $f_4(\mathbf{x}) = \max_{i=1,\ldots n}\{|x_i|\}$.
Search space: $-100 \leq x_i \leq 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_4(\mathbf{x}^*) = 0$.

Appendix A.5. Rosenbrock Function (f_5)

Definition: $f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100 \left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right]$.
Search space: $-30 \leq x_i \leq 30,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, \ldots, 1)$, $f_5(\mathbf{x}^*) = 0$.

Appendix A.6. Step Function (f_6)

Definition: $f_6(\mathbf{x}) = \sum_{i=1}^{n}(\lfloor x_i + 0.5 \rfloor)^2$.
Search space: $-100 \leq x_i \leq 100,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_6(\mathbf{x}^*) = 0$.

Appendix A.7. Quartic Function with Noise (f_7)

Definition: $f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$.
Search space: $-1.28 \leq x_i \leq 1.28,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_7(\mathbf{x}^*) = 0$.

Appendix A.8. Schwefel Functions (f_8)

Definition: $f_8(\mathbf{x}) = -\sum_{i=1}^{n}\left(x_i \sin\sqrt{|x_i|}\right)$.
Search space: $-500 \leq x_i \leq 500,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (420.9687, \ldots, 420.9687)$, $f_8(\mathbf{x}^*) = -418.9829n$.

Appendix A.9. Rastrigin Function (f_9)

Definition: $f_9(\mathbf{x}) = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$.
Search space: $-5.12 \leq x_i \leq 5.12,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_9(\mathbf{x}^*) = 0$.

Appendix A.10. Ackley Function (f_{10})

Definition: $f_{10}(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$.
Search space: $-32 \leq x_i \leq 32,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$; $f_{10}(\mathbf{x}^*) = 0$.

Appendix A.11. Griewank Function (f_{11})

Definition: $f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$.
Search space: $-600 \leq x_i \leq 600,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0)$, $f_{11}(\mathbf{x}^*) = 0$.

Appendix A.12. Levy Functions (f_{12},f_{13})

Definition:

$$f_{12}(\mathbf{x}) = \frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - 1)^2(1 + 10\sin^2(\pi y_i + 1))\right] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),\ y_i = 1 + \frac{x_i - 1}{4},\ i = 1, \ldots, n.$$

$$f_{13}(\mathbf{x}) = \frac{1}{10}\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}\left[(x_i - 1)^2(1 + \sin^2(3\pi x_i + 1))\right] + (x_n - 1)^2(1 + \sin^2(2\pi x_n)) + \sum_{i=1}^{n} u(x_i, 5, 100, 4),$$

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

Search space: $-50 \leq x_i \leq 50,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, \ldots, 1)$, $f_{12}(\mathbf{x}^*) = f_{13}(\mathbf{x}^*) = 0$.

Appendix B. Hard Test Functions

Twenty-five hard test functions h_1–h_{25} are listed in Table A1 with their global minima and bounds. The reader is directed to reference [74,75] for more details on these functions.

Table A1. Hard Test Functions.

h	Function Name	Bounds	Global Min
h_1	Shifted sphere function	$[-100, 100]$	-450
h_2	Shifted Schwefel's function 1.2	$[-100, 100]$	-450
h_3	Shifted rotated high conditioned elliptic function	$[-100, 100]$	-450
h_4	Shifted Schwefel's function 1.2 with noise in fitness	$[-100, 100]$	-450
h_5	Schwefel's function 2.6 with global optimum on bounds	$[-100, 100]$	-310
h_6	Shifted Rosenbrock's function	$[-100, 100]$	390
h_7	Shifted rotated Griewank's function without bounds	$[0, 600]$	-180
h_8	Shifted rotated Ackley's function with global optimum on bounds	$[-32, 32]$	-140
h_9	Shifted Rastrigin's function	$[-5, 5]$	-330
h_{10}	Shifted rotated Rastrigin's function	$[-5, 5]$	-330
h_{11}	Shifted rotated Weierstrass function	$[-0.5, 0.5]$	90
h_{12}	Schwefel's function 2.13	$[-100, 100]$	-460
h_{13}	Expanded extended Griewank's + Rosenbrock's function	$[-3, 1]$	-130
h_{14}	Expanded rotated extended Scaffer's function	$[-100, 100]$	-300
h_{15}	Hybrid composition function	$[-5, 5]$	120
h_{16}	Rotated hybrid composition function	$[-5, 5]$	120
h_{17}	Rotated hybrid composition function with noise in fitness	$[-5, 5]$	120
h_{18}	Rotated hybrid composition function	$[-5, 5]$	10
h_{19}	Rotated hybrid composition function with narrow basin global optimum	$[-5, 5]$	10
h_{20}	Rotated hybrid composition function with global optimum on the bounds	$[-5, 5]$	10
h_{21}	Rotated hybrid composition function	$[-5, 5]$	360
h_{22}	Rotated hybrid composition function with high condition number matrix	$[-5, 5]$	360
h_{23}	Non-Continuous rotated hybrid composition function	$[-5, 5]$	360
h_{24}	Rotated hybrid composition function	$[-5, 5]$	260
h_{25}	Rotated hybrid composition function without bounds	$[2, 5]$	260

References

1. Hedar, A.R.; Allam, A.A.; Deabes, W. Memory-Based Evolutionary Algorithms for Nonlinear and Stochastic Programming Problems. *Mathematics* **2019**, *7*, 1126, doi:10.3390/math7111126. [CrossRef]
2. Boussaïd, I.; Lepagnot, J.; Siarry, P. A survey on optimization metaheuristics. *Inf. Sci.* **2013**, *237*, 82–117. [CrossRef]
3. Noghanian, S.; Sabouni, A.; Desell, T.; Ashtari, A. Global optimization: Differential evolution, genetic algorithms, particle swarm, and hybrid methods. In *Microwave Tomography*; Springer: Berlin, Germany, 2014; pp. 39–61.
4. Noack, M.M.; Funke, S.W. Hybrid genetic deflated Newton method for global optimisation. *J. Comput. Appl. Math.* **2017**, *325*, 97–112. [CrossRef]
5. Talbi, E.G. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Ann. Oper. Res.* **2016**, *240*, 171–215. [CrossRef]
6. Bansal, J.C.; Singh, P.K.; Pal, N.R. *Evolutionary and Swarm Intelligence Algorithms*; Springer: Berlin, Germany, 2019.
7. Bozorg-Haddad, O.; Solgi, M.; Loaiciga, H.A. *Meta-heuristic and Evolutionary Algorithms for Engineering Optimization*; John Wiley & Sons: Hoboken, NJ, USA, 2017; Volume 294.
8. Emmerich, M.; Shir, O.M.; Wang, H. Evolution strategies. In *Handbook of Heuristics*; Springer: Cham, Switzerland, 2018; pp. 89–119, doi:10.1007/978-3-319-07124-4_13. [CrossRef]
9. Mahmoodabadi, M.; Nemati, A. A novel adaptive genetic algorithm for global optimization of mathematical test functions and real-world problems. *Eng. Sci. Technol. Int. J.* **2016**, *19*, 2002–2021. [CrossRef]
10. Ong, Y.S.; Lim, M.H.; Zhu, N.; Wong, K.W. Classification of adaptive memetic algorithms: A comparative study. *IEEE Trans. Syst. Man Cybern. B Cybern.* **2006**, *36*, 141–152.

11. Nguyen, Q.H.; Ong, Y.S.; Lim, M.H. A probabilistic memetic framework. *IEEE Trans. Evol. Comput.* **2009**, *13*, 604–623. [CrossRef]
12. Whitley, D.; Chicano, F.; Ochoa, G.; Sutton, A.M.; Tinós, R. Next generation genetic algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference, Berlin, Germany, 15–19 July 2017; pp. 922–941.
13. Ahrari, A.; Kramer, O. Finite life span for improving the selection scheme in evolution strategies. *Soft Comput.* **2017**, *21*, 501–513. doi:10.1007/s00500-015-1805-3. [CrossRef]
14. Toledo, C.F.M.; Oliveira, L.; França, P.M. Global optimization using a genetic algorithm with hierarchically structured population. *J. Comput. Appl. Math.* **2014**, *261*, 341–351. [CrossRef]
15. Hedar, A.R.; Fukushima, M. Tabu search directed by direct search methods for nonlinear global optimization. *Eur. J. Oper. Res.* **2006**, *170*, 329–349. [CrossRef]
16. Hedar, A.R.; Ali, A.F. Tabu search with multi-level neighborhood structures for high dimensional problems. *Appl. Intell.* **2012**, *37*, 189–206. [CrossRef]
17. Mascia, F.; Pellegrini, P.; Birattari, M.; Stützle, T. An analysis of parameter adaptation in reactive tabu search. *Int. Trans. Oper. Res.* **2014**, *21*, 127–152. doi:10.1111/itor.12043. [CrossRef]
18. Hedar, A.R.; Fukushima, M. Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optim. Methods Softw.* **2004**, *19*, 291–308. [CrossRef]
19. Thakur, M. A new genetic algorithm for global optimization of multimodal continuous functions. *J. Comput. Sci.* **2014**, *5*, 298–311. [CrossRef]
20. Yang, C.; Kumar, M. An information guided framework for simulated annealing. *J. Glob. Optim.* **2015**, *62*, 131–154, doi:10.1007/s10898-014-0229-4. [CrossRef]
21. Saleh, A.; Hameed, H. A Novel Hybrid Algorithm of Differential Evolution with Evolving Spiking Neural Network for Pre-synaptic Neurons Optimization. *Int. J. Adv. Soft Comput.* **2014**, *6*, 1–16.
22. Tang, L.; Dong, Y.; Liu, J. Differential Evolution with an Individual-Dependent Mechanism. *IEEE Trans. Evol. Comput.* **2015**, *19*, 560–574. doi:10.1109/TEVC.2014.2360890. [CrossRef]
23. Cheng, S.; Lu, H.; Lei, X.; Shi, Y. A quarter century of particle swarm optimization. *Complex Intell. Syst.* **2018**, *4*, 227–239, doi:10.1007/s40747-018-0071-2. [CrossRef]
24. Esmin, A.A.; Coelho, R.A.; Matwin, S. A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data. *Artif. Intell. Rev.* **2015**, *44*, 23–45, doi:10.1007/s10462-013-9400-4. [CrossRef]
25. Socha, K.; Dorigo, M. Ant colony optimization for continuous domains. *Eur. J. Oper. Res.* **2008**, *185*, 1155–1173. [CrossRef]
26. Mladenović, N.; Dražić, M.; Kovačevic-Vujčić, V.; Čangalović, M. General variable neighborhood search for the continuous optimization. *Eur. J. Oper. Res.* **2008**, *191*, 753–770. [CrossRef]
27. Hvattum, L.M.; Duarte, A.; Glover, F.; Martí, R. Designing effective improvement methods for scatter search: An experimental study on global optimization. *Soft Comput.* **2013**, *17*, 49–62. [CrossRef]
28. Li, K.; Tian, H. A DE-based scatter search for global optimization problems. *Discrete Dyn. Nat. Soc.* **2015**, *2015*, 1–9. [CrossRef]
29. Kumar, S.; Singh, S.K. Hybrid BFO and PSO Swarm Intelligence Approach for Biometric Feature Optimization. *Nat. Inspired Comput.* **2016**, *7*, 1490–1518, doi:10.4018/978-1-5225-0788-8.ch057. [CrossRef]
30. Piotrowski, A.P. Adaptive memetic differential evolution with global and local neighborhood-based mutation operators. *Inf. Sci.* **2013**, *241*, 164–194, doi:10.1016/j.ins.2013.03.060. [CrossRef]
31. Sahnehsaraei, M.A.; Mahmoodabadi, M.J.; Taherkhorsandi, M.; Castillo-Villar, K.K.; Yazdi, S.M. A hybrid global optimization algorithm: Particle swarm optimization in association with a genetic algorithm. In *Complex System Modelling and Control Through Intelligent Soft Computations*; Springer: Berlin, Germany, 2015; pp. 45–86.
32. Hosseini, S.S.S.; Yang, X.S.; Gandomi, A.H.; Nemati, A. Solutions of Non-smooth Economic Dispatch Problems by Swarm Intelligence. In *Adaptation and Hybridization in Computational Intelligence*; Springer: Cham, Switzerland, 2015; Volume 18, pp. 136–153, doi:10.1007/978-3-319-14400-9. [CrossRef]
33. Coello, C.A.C. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Comput. Methods Appl. Mech. Eng.* **2002**, *191*, 1245–1287. [CrossRef]
34. Mallipeddi, R.; Suganthan, P.N. Ensemble of constraint handling techniques. *IEEE Trans. Evol. Comput.* **2010**, *14*, 561–579. [CrossRef]

35. Mezura-Montes, E. *Constraint-Handling in Evolutionary Optimization*; Springer: Berlin, Germany, 2009; Volume 198.
36. Coello Coello, C.A. Constraint-handling techniques used with evolutionary algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference, Denver, CO, USA, 20–24 July 2016; pp. 563–587.
37. Hedar, A.R.; Fukushima, M. Derivative-free filter simulated annealing method for constrained continuous global optimization. *J. Glob. Optim.* **2006**, *35*, 521–549. [CrossRef]
38. Da Silva, I.N.; Spatti, D.H.; Flauzino, R.A.; Liboni, L.H.B.; dos Reis Alves, S.F. *Artificial Neural Networks*; Springer International Publishing: Cham, Switzerland, 2017.
39. Hedar, A.R.; Ong, B.T.; Fukushima, M. Genetic Algorithms with Automatic Accelerated Termination. Available online: http://www-optima.amp.i.kyoto-u.ac.jp/~fuku/papers/G3AT.pdf (accessed on 14 January 2020).
40. Hedar, A.R. Adaptive Memory Matrices for Automatic Termination of Evolutionary Algorithms. In Proceedings of the Fourth International Conference on Informatics & Applications, Takamatsu, Japan, 20–22 July 2015; pp. 1–11.
41. Moscato, P. Memetic algorithms: A short introduction. In *New Ideas in Optimization*; McGraw-Hill Ltd.: London, UK, 1999; pp. 219–234.
42. Lozano, M.; Herrera, F.; Krasnogor, N.; Molina, D. Real-coded memetic algorithms with crossover hill-climbing. *Evol. Comput.* **2004**, *12*, 273–302. [CrossRef]
43. Gibbs, M.S.; Maier, H.R.; Dandy, G.C.; Nixon, J.B. Minimum number of generations required for convergence of genetic algorithms. In Proceedings of the IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 565–572.
44. Jain, B.J.; Pohlheim, H.; Wegener, J. On termination criteria of evolutionary algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 7–11 July 2001; Morgan Kaufmann Publishers: Burlington, MA, USA, 2001.
45. Hedar, A.R.; Fukushima, M. Directed Evolutionary Programming: Towards an Improved Performance of Evolutionary Programming. In Proceedings of the IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 1521–1528, doi:10.1109/cec.2006.1688489. [CrossRef]
46. Leung, Y.W.; Wang, Y. An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Trans. Evol. Comput.* **2001**, *5*, 41–53. [CrossRef]
47. Tsai, J.T.; Liu, T.K.; Chou, J.H. Hybrid Taguchi-genetic algorithm for global numerical optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 365–377. [CrossRef]
48. Wang, L.; Xu, R.M.; Yan, B. Accurate small-signal model extraction for pHEMT on GaAs. *Int. J. Infrared Millim. Waves* **2007**, *28*, 1133–1141. [CrossRef]
49. Koumousis, V.K.; Katsaras, C.P. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Trans. Evol. Comput.* **2006**, *10*, 19–28. [CrossRef]
50. Lim, D.; Ong, Y.S.; Jin, Y.; Sendhoff, B. Trusted evolutionary algorithm. In Proceedings of the IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 149–156.
51. Zhou, Z.; Ong, Y.S.; Nair, P.B.; Keane, A.J.; Lum, K.Y. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* **2007**, *37*, 66–76. [CrossRef]
52. Safe, M.; Carballido, J.; Ponzoni, I.; Brignole, N. On stopping criteria for genetic algorithms. In *Advances in Artificial Intelligence—SBIA 2004*; Springer: Berlin, Germany, 2004; pp. 405–413.
53. Kaelo, P.; Ali, M.M. Integrated crossover rules in real coded genetic algorithms. *Eur. J. Oper. Res.* **2007**, *176*, 60–76. [CrossRef]
54. Tsoulos, I.G. Modifications of real code genetic algorithm for global optimization. *Appl. Math. Comput.* **2008**, *203*, 598–607. [CrossRef]
55. Engelbrecht, A.P. *Computational Intelligence: An Introduction*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
56. Konar, A. *Computational Intelligence: Principles, Techniques and Applications*; Springer Science & Business Media: Berlin, Germany, 2006.
57. Ding, S.; Li, H.; Su, C.; Yu, J.; Jin, F. Evolutionary artificial neural networks: A review. *Artif. Intell. Rev.* **2013**, *39*, 251–260. [CrossRef]

58. Such, F.P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K.O.; Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv* **2017**, arXiv:1712.06567.
59. Cheng, L.; Hou, Z.G.; Lin, Y.; Tan, M.; Zhang, W.C.; Wu, F.X. Recurrent neural network for non-smooth convex optimization problems with application to the identification of genetic regulatory networks. *IEEE Trans. Neural Netw.* **2011**, *22*, 714–726. [CrossRef]
60. Schweidtmann, A.M.; Mitsos, A. Deterministic global optimization with artificial neural networks embedded. *J. Optim. Theory Appl.* **2019**, *180*, 925–948. [CrossRef]
61. Gunst, R.F. *Regression Analysis and Its Application: A Data-Oriented Approach*; CRC Press: Boca Raton, FL, USA, 2018.
62. Dokeroglu, T.; Sevinc, E.; Kucukyilmaz, T.; Cosar, A. A survey on new generation metaheuristic algorithms. *Comput. Ind. Eng.* **2019**, *137*, 106040, doi:10.1016/j.cie.2019.106040. [CrossRef]
63. Liu, Z.; Liu, Y.; Xiong, L. Robust Linear Neural Network for Constrained Quadratic Optimization. *Discrete Dyn. Nat. Soc.* **2017**, *2017*, doi:10.1155/2017/5073640. [CrossRef]
64. Ghasabi-Oskoei, H.; Mahdavi-Amiri, N. An efficient simplified neural network for solving linear and quadratic programming problems. *Appl. Math. Comput.* **2006**, *175*, 452–464, doi:10.1016/j.amc.2005.07.025. [CrossRef]
65. Feng, J.; Qin, S.; Shi, F.; Zhao, X. A recurrent neural network with finite-time convergence for convex quadratic bilevel programming problems. *Neural Comput. Appl.* **2018**, *30*, 3399–3408, doi:10.1007/s00521-017-2926-7. [CrossRef]
66. Baker, J.E. Adaptive selection methods for genetic algorithms. In Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications, Pittsburgh, PA, USA, 24–26 July 1985; pp. 101–111.
67. Herrera, F.; Lozano, M.; Verdegay, J.L. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artif. Intell. Rev.* **1998**, *12*, 265–319. [CrossRef]
68. Nelder, J.A.; Mead, R. A simplex method for function minimization. *Comput. J.* **1965**, *7*, 308–313. [CrossRef]
69. Beyer, H.G.; Schwefel, H.P. Fast evolution strategies: A comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52, doi:10.2146/130117. [CrossRef]
70. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*; Springer Science & Business Media: Berlin, Germany, 2003.
71. Vavasis, S.A. Complexity issues in global optimization: A survey. In *Handbook of Global Optimization*; Springer: Berlin, Germany, 1995; pp. 27–41.
72. Talbi, E.G. *Metaheuristics: From Design to Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2009; Volume 74.
73. Yang, X.S. Metaheuristic optimization: Algorithm analysis and open problems. In Proceedings of the International Symposium on Experimental Algorithms, Crete, Greece, 5–7 May 2011; pp. 21–32.
74. Liang, J.; Suganthan, P.; Deb, K. Novel composition test functions for numerical global optimization. In Proceedings of the 2005 IEEE Swarm Intelligence Symposium, Pasadena, CA, USA, 8–10 June 2005; pp. 68–75.
75. Suganthan, P.N.; Hansen, N.; Liang, J.J.; Deb, K.; Chen, Y.P.; Auger, A.; Tiwari, S. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Available online: http://www.cmap.polytechnique.fr/~nikolaus.hansen/Tech-Report-May-30-05.pdf (accessed on 14 January 2020).
76. Hansen, N.; Müller, S.D.; Koumoutsakos, P. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evol. Comput.* **2003**, *11*, 1–18, doi:10.1162/106365603321828970. [CrossRef]
77. García, S.; Fernández, A.; Luengo, J.; Herrera, F. A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability. *Soft Comput.* **2009**, *13*, 959. [CrossRef]
78. Sheskin, D.J. *Handbook of Parametric and Nonparametric Statistical Procedures*; CRC Press: Boca Raton, FL, USA, 2003.
79. Zar, J.H. *Biostatistical Analysis*; Pearson Higher Education: London, UK, 2013.
80. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]

81. García-Martínez, S.; Molina, D.; Lozano, M.; Herrera, F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics* **2009**, *15*, 617–644. [CrossRef]
82. Xue, Y.; Jiang, J.; Zhao, B.; Ma, T. A self-adaptive artificial bee colony algorithm based on global best for global optimization. *Soft Comput.* **2018**, *22*, 2935–2952. [CrossRef]
83. Gao, W.; Liu, S.; Huang, L. A global best artificial bee colony algorithm for global optimization. *J. Comput. Appl. Math.* **2012**, *236*, 2741–2753, doi:10.1016/j.cam.2012.01.013. [CrossRef]