Towards a Quantum based GA Search for an Optimal Artificial Neural Networks Architecture and Feature Selection to Model NOx Emissions: A Case Study

Mazen Azzam
Department of Chemical and Petroleum Engineering
American University of Beirut
Beirut, Lebanon

Joseph Zeaiter
Department of Chemical and Petroleum Engineering
American University of Beirut
Beirut, Lebanon
jz08@aub.edu.lb

Mariette Awad
Humans and Machines Lab
Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
ma162@aub.edu.lb

Abstract— **This paper describes the methodology used to design a NO_x predictive emissions monitoring system (PEMS) based on an artificial neural network (ANN). To find the optimal ANN architecture, a QGA-based search was performed over the set of possible model architectures, the activation function in each layer and the learning rate were architecture parameters taken into consideration. In addition, the QGA performed feature selection to remove non-significant input parameters that did not contribute significantly to the output. The objective function included penalties for network complexity and generalization error, among which a newly introduced penalty that makes use of the effective number of parameters provided by Bayesian Regularization. The developed framework was tested on data collected from a power plant consisting of twelve diesel engine-powered generators. It was found that the newly introduced penalty was enough to yield well-performing ANN's who demonstrated superior performance when compared to some traditional ML models, with up to a 75% reduction in the mean relative error when compared with a normal radial-basis network. In addition, the QGA was better at avoiding local optima, and on average converged in half the time required for the classical GA to converge.**

Keywords— Predictive Emissions Monitoring System, Quantum GA, Prediction, Continuous Monitoring, ML, Evolutionary Computation, Neural Networks, Quantum Computing

I. INTRODUCTION

Internal combustion engines and diesel engines in particular are popular nowadays due to their high thermal efficiency and low fuel consumption [1]. Strict environmental regulations impose limits on the allowed levels of emissions from these engines, forcing manufacturers to adopt constant pollution-monitoring systems. Hardware-based monitoring methods use direct measurements from sensors and analyzing equipment mounted on flue gas stacks. These traditional approaches suffer from a number of disadvantages, including the amount of time required to receive measurement data and the operational and capital costs associated with installing and maintaining such equipment. More sophisticated approaches make use of mathematical modelling to predict emissions against various engine-related operating conditions. These approaches constitute what is known as "Predictive Emissions Monitoring Systems (PEMS)" which have gained popularity in the past few decades due to constant advances in computing power and approval by relevant authorities in the US and Europe [2-4].

Smith [3] estimates that anywhere between $300,000 and $1 million of savings in operational costs across a 10-year lifespan can be achieved when hardware-based emissions monitoring methods are replaced by software-based PEMS. The savings can also reach initial capital costs as the need to purchase, install, and maintain new equipment is at least partially alleviated. On the other hand, NO_x emissions are known to have greatly detrimental effects on health and environment, as their high reactivity causes them to be present abundantly in atmospheric chemistry, and their high solubility in water allows them to diffuse to the different parts of the respiratory system [5]. In one case study, it was estimated that the marginal external health costs of NO_x emissions from the Belgian transportation sector after 2007 could reach around €4000 per ton of NO_x emissions [6]. Therefore, NO_x emissions from different sources is a main concern for policy-makers and authorities, and any policy regarding such emissions must include efficient detection and measurement of NO_x emissions. To that end, PEMS provide a promising tool for reliable monitoring of NO_x emissions. However, the model on which a given PEMS relies on must be optimized to ensure accurate measurements.

PEMS rely essentially on a model relating emissions with the relevant factors (inputs/features) that directly influence them. Thus, PEMS can fall into different categories depending on the origin of that model. First-principles-based PEMS rely on some sort of physical interpretation of the process in question; mainly thermodynamics, fluid mechanics and chemical kinetics. On the other hand, some PEMS employ a pure "black-box" approach to predict emissions, relying on historical data and proper mathematical methods to correlate between specified inputs and outputs [7]. This class of PEMS relies mostly on ML methods, of which artificial neural networks (ANN's) have especially gained popularity.

ANN-based PEMS have been widely popular in the industry, and have found applications in different classes of process equipment, including thermal power plants [8, 9], palm oil mills [4], incinerators [10, 11], boilers [12], gas turbines [13, 14], coal combustors [15] and general industrial

This work was financially supported by the Munib and Angela Masri Institute for Energy and Natural Resources.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

areas [16]. In addition to multi-layer perceptron (MLP), architectures such as radial-basis function neural networks [17], generalized regression networks [12], and fuzzy neural networks [9] can be found in the literature. Besides ANN's, other machine learning (ML) methods have been employed in the literature, the most common of which being support vector machines [10, 12, 15].

For diesel engines in particular, several attempts at modeling NO_x emissions can be found in the literature. First-principles models have been employed in [18-22] with high accuracy. However, due to the complexity of the combustion process and the physics-based model for internal combustion engines, ML-based modeling is becoming more attractive [23-25]. In particular, ANN's have been the most popular in modeling exhaust emissions and performance of diesel [26, 27], biodiesel [28, 29], n-heptane [30], and gasoline engines [23]. Recently, Lotfan et al. [31] used a one-layered ANN to model CO and NO_x emissions from a direct-injection dual fuel engine and employed it as an objective function for a genetic algorithm (GA) in order to minimize emissions. Bendu et. al. [32] modeled NO_x emissions from an ethanol-fueled engine using a generalized regression neural network (GRNN) and minimized emissions by virtue of a particle swarm optimization (PSO) algorithm. In addition to ANN's, support vector regression has been used well [24, 33], and has outperformed ANN's in some applications [34].

For ANN's, most of the studies on diesel engine modeling employed simple, one-layered feedforward architectures trained with backpropagation and produced satisfactory results. Other architecture types such as the radial basis function (RBF) network [35] and the GRNN mentioned earlier [32] can be found as well but with a more limited use. Previous studies relied mainly on trial-and-error and user experience to determine the optimal number of neurons in the hidden layer. In our previous work [36], we introduced the GA-based design of ANN's for PEMS applications, and we showed its superior performance over ANN's constructed by trial-and-error. In this research, we extend our work and employ a quantum genetic algorithm (QGA) in order to perform feature selection and architecture optimization applied to a real-world diesel engine power plant operating at the American University of Beirut (AUB).

The rest of the paper is structured such that Section II presents a brief overview of the QGA and describes details regarding the parameters of the computational experiments performed. We then describe the procedure in Section III before presenting and discussing the results in Sections IV and V. The paper concludes in Section VI.

II. Quantum Genetic Algorithm and Architecture Optimization

Fig. 1 shows the algorithm used for the optimal architecture design. The following two sections provide a brief overview of the QGA and the method used to encode ANN individuals.

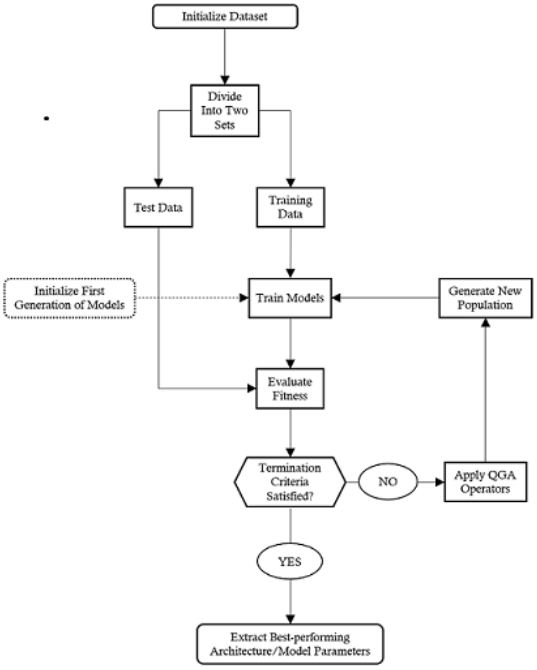


Fig. 1. General procedure used to design optimal ANN architecture using the QGA.

A. The Quantum Genetic Algorithm QGA

Our selection for QGA stems from its theoretical advantages. The growing body of research over the past decade on quantum computing and the possibility of building a machine that uses quantum-mechanical phenomena to perform operations on data has led to the rise of "quantum" algorithms that exploit the exponential speedup achievable on such machines. This exponential speedup is due to the quantum superposition property of electrons, which states that an electron exists in a superposition of two states (quantum state) until it is observed or measured. The act of measuring an electron causes a collapse in its state to one of the two basis states.

Many quantum algorithms have been developed to exploit the exponential speedup of quantum computers; one such algorithm is QGA as proposed in [44].

Quantum computing allows this variant of GA to explore a larger portion of the search space in less time. Furthermore, the mathematical properties of quantum mechanics, described by quantum probability theory, allow the GA to reach different parts of the space that would not have been reached using classical GA. Specifically, quantum states behave as waves instead of particles which leads to constructive and destructive interference. Assigning complex probabilities to the basis states leads to a non-commutative probability theory. The evolution of these probabilities is governed by Schrodinger's equation and includes Hamiltonian and unitary evolutionary operators. Measuring these quantum states leads to their collapse and introduces additional randomness into the system. All these factors result in a search space traversal by QGA that is different from classical GA's path.

This version of the GA has been applied over the past decade in a variety of optimization problems, such as economic dispatch modeling for a wind power system [45], neuro-fuzzy controller design [46], and more recently, task allocation in multi-robot systems [47] and multiprocessor task scheduling [48]; however, not in the context of PEMS.

Theoretically, the QGA presents an advantage over GA by requiring fewer chromosomes at every generation. The representation of every individual in the QGA as a string of qubits instead of classical bits allows for more exploration of the solution space, as from a quantum mechanics perspective, the entire population may be then considered as one chromosome in a state of superposition [44].

In a classical GA, individuals representing possible solutions are usually encoded in a binary string, with every bit or group of bits representing a certain feature of the individual. In QGA's, a classical bit is replaced by a "qubit - $|\varphi\rangle$" that represents a superposition of the two basis states – $|0\rangle$ and $|1\rangle$ – in a linear combination of the form:

$$|\varphi\rangle=\begin{pmatrix}\alpha\\ \beta\end{pmatrix}=\alpha\begin{pmatrix}1\\ 0\end{pmatrix}+\beta\begin{pmatrix}0\\ 1\end{pmatrix}=\alpha|0\rangle+\beta|1\rangle \quad (1)$$

α and β are complex numbers whose norms represent the probability of finding the qubit upon observation, either in the $|0\rangle$ or $|1\rangle$ state, respectively. Therefore, the following relationship holds:

$$|\alpha|^2+|\beta|^2=1 \quad (2)$$

As a result, a chromosome x_j in a QGA is an array of ordered pairs, (α_i,β_i) representing individual qubits which will in turn be mapped to classical bits that represent features of each individual:

$$x_j=\begin{pmatrix}\alpha_1 & \alpha_2 & \alpha_3 & \dots & \alpha_L\\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_L\end{pmatrix} \quad (3)$$

The steps taken by the QGA are similar to those of the classical GA, namely the population generation and the application of genetic operators to generate new, possibly improved individuals. Fig. 2 shows the main steps taken by a typical QGA.

Inputs: objective function, termination criteria

Outputs: optimal point

(1) Set $t \leftarrow 1$ and initialize a quantum population $Q(t)$
(2) **while** (termination criteria not reached)
  (1) Provide a measure of the quantum population, by collapsing it to its classical counterpart: $Q(t) \rightarrow P(t)$
  (2) Evaluate the fitness of every individual using the objective function
  (3) Apply QGA operators to generate new quantum population
  (4) $t \leftarrow t+1$
(3) end

Fig. 2. Main steps for a QGA.

Instead of the usual selection, crossover, and mutation operators employed in classical GA's, QGA's employ mainly what is known as "quantum-gates (Q-gates)" to update current population. Q-gates are unitary transformations applied on qubits using 2×2 unitary matrices. QGA's employ mainly three operators based on Q-gates to generate the new population:

1. *Qubit rotation gate*: this operator "rotates" a qubit in a quantum chromosome with an angle that is chosen such that the current qubit rotates toward the individual b that has the best fitness in the population:

$$\begin{pmatrix}\alpha_i^{t+1}\\ \beta_i^{t+1}\end{pmatrix}=\begin{pmatrix}\cos\delta\theta_i & -\sin\delta\theta_i\\ \sin\delta\theta_i & \cos\delta\theta_i\end{pmatrix}\begin{pmatrix}\alpha_i^{t}\\ \beta_i^{t}\end{pmatrix} \quad (4)$$

The rotation angle is given by:

$$\delta\theta_j=\mathrm{sgn}\left(\alpha_j,\beta_j\right)\Delta\theta_j \quad (5)$$

Where $\mathrm{sgn}\left(\alpha_j,\beta_j\right)$ and $\Delta\theta_j$ represent respectively the angle direction and rotation value, which can be determined using several different methods that are described in [44]. In this work, we adopt an adaptive strategy as suggested by [49]:

$$\Delta\theta_j=\theta\left(\theta\min_{\max}\frac{|f(x_j)-f(b)|}{\max(f(x_j),f(b))}\right)_{\min} \quad (6)$$

Where θ_{min} and θ_{max} are respectively minimum and maximum values set by the user for the rotation angle, and f is the objective/fitness function.

2. *Qubit mutation-inversion gate*: the mutation-inversion operator swaps, with a typically small probability, the amplitudes of a randomly chosen qubit in the chromosome. This is achieved by applying the so-called Pauli X gate:

$$\begin{pmatrix}\alpha_i^{t+1}\\ \beta_i^{t+1}\end{pmatrix}=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}\alpha_i^{t}\\ \beta_i^{t}\end{pmatrix}=\begin{pmatrix}\beta_i^{t}\\ \alpha_i^{t}\end{pmatrix} \quad (7)$$

3. *Qubit mutation-insertion gate*: the mutation-insertion operator swaps the positions of two randomly chosen qubits in the chromosome. This is usually applied with a small probability as well.

In addition to these operators, a crossover operator similar to that of the one used in classical GA can be used. Several variants of the QGA exist in the literature, and a detailed discussion of such variants can be found in [44].

B. Chromosome Encoding of ANN Individuals

An indirect binary encoding scheme was employed to represent ANN individuals. After the measurement – or collapse – of the qubits, a classical binary string is obtained which represents a chromosome that encodes information about the structure of the ANN and the selected features. The chromosome representing every ANN was thus divided into four groups (Fig. 3).

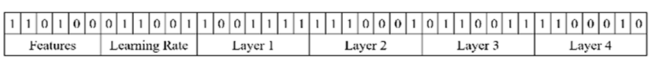
The first group is for feature selection and consists of six bits each corresponding to one of the inputs. A certain input or feature is considered only when its corresponding bit assumes a value of 1.

The second group corresponds to the learning rate and consists of six bits used to compute it as follows:

$$\mu=\sum_{k=1}^{6} b_k 2^{-k};\ b_k\in\{0,1\} \quad (8)$$

The third group represents the layers of the network and encodes both the number of neurons in every layer and the activation function. Every seven bits represent a certain layer, where the first four bits are used to compute the number of neurons and the last three bits evaluate an integer that is associated with an activation function.

Fig. 3 shows a detailed example of decoding the structure of an ANN from a binary string with the proposed encoding scheme.



- Selected Features: (1), (2), (4)
- Learning Rate: $0+1\times2^{-2}+1\times2^{-4}+0+0+1\times2^{-6}=0.3906$
- Layers:

  1: Neurons: $1\times2^3+1\times2^0-8$; Activation Function: $1\times2^2+1\times2^1+1\times2^0-6$ (soft max)

  2: Neurons: $1\times2^3+1\times2^2+1\times2^1=14$; Activation Function: $1\times2^0=1$ (tangent sigmoid)

  3: Neurons: $1\times2^2+1\times2^1-6$; Activation Function: $1\times2^1+1\times2^0-3$ (radial basis)

  4: Neurons: $1\times2^3+1\times2^2=12$; Activation Function: $1\times2^1=2$ (competitive)

Fig. 3. Example illustrating the indirect binary encoding scheme used in this study.

Table I lists the values considered for every architecture parameter undergoing optimization.

TABLE I. VALUES FOR EVERY CONSIDERED ARCHITECTURE PARAMETER

Architecture Parameter	Set of Considered Values
Number of hidden layers	{1,2,3,4}
Number of neurons in every layer	{1,2,3....15}
Activation functions	{logistic ("logsig"), hyperbolic tangent ("tansig"), competitive ("compet"), radial basis("radbas"), soft max("softmax"), hard limit ("hardlim"), inverse ("inv"), Elliot sigmoid ("elliotsig")}

C. Objective Function

The objective function employed in this work is similar to the one found in our previous work [36]. The function includes two main terms: (1) a complexity-penalizing term that is expressed by a function of the total number of parameters in the network and a ratio of the number of effective parameters to that of the total number of parameters and (2) a performance term that consisted of a weighted average of errors exhibited by the network over training and testing data. The objective function is given by:

$$F(M(1),...,M(4),\mu,SelectedFeatures) = \left[\exp(a_1\gamma^2) + a_2\left(1-\gamma_e/\gamma\right)\right]\left[\rho_1 E_{Test}+\rho_2 E_{Train}\right] \quad (9)$$

Where M(k) is the number of neurons in hidden layer *k*, μ is the learning rate, γ is the total number of parameters, γ_e is the effective number of parameters and E_{Test}, E_{Train} are errors exhibited over testing and training datasets respectively. The weighing factors a_1 and a_2 were chosen by trial and error to contribute appropriately to the output of the function. ρ_1 and ρ_2 were chosen from the range [0; 1] to obtain a weighted average of the testing and training errors. These errors were expressed as mean-squared-errors (MSE):

$$MSE=\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2 \quad (10)$$

Where y_i is the original target value, $\hat{y}_i$ is the output from the ANN, and N is the number of training instances for E_{Train} and testing instances for E_{Test}.

D. Improvements Over Previous Works

Although the methodology employed in this work has been used before, we propose the following modifications:

1. To improve the generalization ability of the individuals, early stopping with cross-validation was used in previous works [42, 43]. In the present work, the limited number of available measurements has resulted in a small dataset with a small number of instances available for validation and testing which may in turn lead to unreliable results in terms of performance evaluation. As such, we adopted Bayesian Regularization (BR) in ANN training, which ensures a lesser risk of overfitting by imposing a sufficiently large regularization penalty on the weight-updating rule and forcing unnecessary weights to converge to zero [39] – without the need for cross-validation. As in our previous work [36], the BR method has allowed for the consideration of a new network complexity penalty term, the number of effective parameters γ_e, which denotes the number of network weight elements that haven't been forced to converge to zero by regularization. The ratio of this number to the total number of parameters (weight elements) is an indicator of network complexity and its risk of overfitting. This term was adopted as a complexity penalty term in the objective function instead of the number of epochs needed for training that was adopted in [42].
2. The consideration of several architecture elements to be evolved has resulted in a large search space for the combinatorial problem. Therefore, it is expected that the performance vs. architecture surface would contain several local optima. To minimize the risk of the QGA getting trapped in a local optimum throughout the generations, we introduce a "perturbation" operator, which operates as follows: if, for a certain number of generations, the optimal solution does not change, a QGA rotation gate with a random angle is applied to certain individuals. The probability of a certain individual in a given generation to be submitted to the perturbation operator is inversely proportional to its fitness. Since the original QGA rotation gate rotates individuals in the direction of the best solution, the perturbation operator allows the algorithm to explore other parts of the performance surface and check for other possible optimal/near-optimal solutions in a more effective way than the traditional mutation-insertion and mutation-inversion operators.
3. In our previous work, [36], a raw exponential term penalizing the total number of parameters was adopted in the objective function, along with a weighing factor a_1 external to the exponential function. This has caused, in some cases, a computational problem with the evaluation of the exponential: large architectures may contain more than 500 weight elements, and as such, the evaluation of their objective function value with the raw exponential term may cause integer overflow. In this work, instead of adopting a raw exponential of the total number of parameters, we use the exponential of a non-linear function of the latter number, allowing for better control of the effect of this term on the objective function and avoiding integer overflow. The parabolic function, used in this work with a coefficient ranging a_1 from 10^{-8} to 10^{-5} (9), ensures that the exponential increases slowly with a small number of weight elements, and fast with large numbers, without causing integer overflow. One may eliminate the effect of the total number of parameters on the objective function by simply setting the coefficient a_1 to zero in Equation (9).

III. DATA ACQUISITION AND PRE-PROCESSING

The AUB campus is equipped with a backup power plant consisting of twelve spark-ignited 8-cylinder Caterpillar 3608 diesel engines serving different areas of the campus during power shedding which happens quite often in Beirut. At startup, the power load on each engine is brought steadily from 0 to 1200 kW over a period of 15 to 20 minutes. During this startup period, an electrochemical ENERAC-500 direct emissions analyzer was used to measure NO_x emissions from the exhaust of one of the engines. In addition to emissions,

the analyzer also recorded the ambient and exhaust temperatures, respectively. We also measured the voltage, current, and power factor provided by the engine as they fluctuate during startup; along with the variable energy load on the generator. Other parameters such as fuel flowrate, air intake rate and engine speed were constant. These measurements were conducted over four months from June to September 2017 and resulted in a dataset comprised of around 200 instances of NO_x emissions in ppm versus all aforementioned parameters. Of the 200 data points, around 70% were used for training and 30% for testing. Table II summarizes the details concerning the inputs used to predict NO_x emissions levels.

Input	Range
(1) Energy load	[0- 1200 kW]
(2) Supplied Voltage	[0- 3350 V]
(3) Supplied Current	[0- 180 A]
(4) Power Factor	[-0.99- 0.99]
(5) Ambient Temperature	[21- 27 °C]
(6) Stack Gas Temperature	[41- 282 °C]

TABLE II. INPUTS (FEATURES) FOR THE TRAINING DATASET

The computational experiments were performed on a machine employing a quad-core Intel i7-6700HQ clocked at 2.6 Hz with 16 GB of RAM. Codes were written in MATLAB R2017a.

IV. RESULTS FROM THE QGA RUNS

Since the purpose of ANN training is to achieve high generalization performance, more weight in the QGA's objective function was given to the error over the testing set, with ρ_1=0.99 and ρ_2=0.01 in Equation (9). As for the complexity-penalizing terms, several runs with different weighing factors a_1 and a_2 were performed in order to inspect the range of these terms and especially prevent the exponential term from dominating the objective function. Two termination criteria were imposed on the QGA:

(1) The algorithm is stalled for around 30 generations. If this mean is less than the function tolerance, which is set to 10^{-6}, the algorithm stops.
(2) Maximum number of generations: if the previous criterion is not met in 200 generations, the algorithm stops.

The performance of the networks obtained from the QGA runs was measured with the mean relative error (MRE) exhibited over both training and testing datasets. The overall architecture and the corresponding ratio of the number of effective to total parameters (weight elements) γ_e/γ were indicators of the complexity and the overfitting risk.

A. Scenario no. 1: γ_e/γ = exponential terms in objective function

After inspection, it was found that setting $a_1=10^{-5}$ and a_2=50 in Equation (9) would result in the two terms having close ranges. Table III displays the results of the runs.

B. Scenario no. 2: exponential term = 0 in objective function

Results from the runs are shown in Table IV. Despite the removal of the exponential term that directly penalizes for the total number of weight elements, simpler architectures than the ones observed in Scenario 1 were obtained in Scenario no. 2. Moreover, the average γ_e/γ was higher than the one observed in Scenario 1, with up to 89 % of the weight elements being effective. This indicates that the networks obtained in this set of runs were less prone to overfitting than the ones obtained in the previous set. This observation is also supported by the fact that better performance is observed by the networks of Scenario 2 over the testing data, with the MRE reaching as low as 4.41 %.

The runs in Scenario no. 2 indicate that the γ_e/γ term in the objective function is a more effective network complexity penalty that the total number of weight elements in the network. To test this hypothesis, we perform the set of runs in Scenario 3 described next.

TABLE III. RESULTS FOR SCENARIO NO.1[12]

Scenario no.1: $a_1=10^{-5}$ *and* $a_2=50$											
Run	**Architecture**	**Selected Features**						**Learning Rate**	γ_e/γ	**MRE (%)**	
		1	**2**	**3**	**4**	**5**	**6**			**Test**	**Train**
1	4 × 6 × 5 radbas × tansig × radbas	×		×		×	×	0.968	0.76	5.1	2.42
2	3 × 7 × 3 logsig × radbas × elliotsig	×		×		×	×	0.667	0.87	5.28	3.3
3	8 × 8 elliotsig × tansig	×		×		×	×	0.271	0.65	5.16	2.25
4	9 × 4 × 2 logsig × elliotsig × netinv			×	×	×	×	0.374	0.54	6.58	4.68
5	5 × 3 radbas × tansig	×	×	×	×	×	×	0.858	0.85	5.50	3.34
Average									**0.74**	**5.52**	**3.2**

C. Scenario no. 3: $\gamma_e/\gamma = 0$ in objective function

Results are shown in Table V. The result of restricting the complexity penalty to the total number of weight elements is almost unchanged. However, the obtained networks are more complex than the ones observed in Scenario no. 2, with most of the networks being composed of three layers with up to 15 neurons in the fifth run. In addition, the average ratio of effective to total number of parameters was 66 %, which is significantly lower than the values observed in Scenarios 1 and 2. Thus, we can infer that the γ_e/γ term is not only enough on its own as a complexity penalty, but it also leads to networks having better generalization performance coupled with lower complexity without the need for a direct penalty on the total number of parameters in the network.

D. Scenario no. 4: objective function with training and testing errors only

As a control, we ran a fourth scenario with an objective function that only considers the training and testing errors as a penalty. Results are shown in Table VI.

Despite results similar to those obtained in the previous scenarios in terms of performance of the networks and the MRE exhibited over the testing dataset, it is clear that the removal of the complexity penalties has led to very complex

[1] The notation for "Architecture" shows the hidden layers with the number of neurons and activation function in each layer. For example, the result from run (1) is a network with three hidden layers, 4 neurons in the first with a radial basis activation function, 6 in the second with the tangent sigmoid and 5 in the third with a radial basis activation function.

[2] For "Selected Features", the × implies that the feature has been selected. The integer attributed to each feature is shown in Table II.

networks with up to 15 neurons per hidden layer. Moreover, the average ratio of the number of effective parameters to the total number of parameters in the networks is considerably lower than the ratios obtained previously. This set of runs thus shows that the presence of the complexity penalties in the objective function is necessary if one is to obtain simple yet well-performing networks without the risk of overfitting.

TABLE IV. RESULTS FROM SCENARIO NO.2

Scenario no.2: $a_1 = 0$ *and* $a_2 = 50$										
Run	**Hidden Layers**	**Selected Features**						**Learning Rate**	γ_e/γ	**MRE (%)**	
		1	**2**	**3**	**4**	**5**	**6**			**Test**	**Train**
1	5 × 4 radbas × tansig	×	×	×	×	×	×	0.881	0.83	4.72	2.88
2	10 logsig	×		×		×	×	0.460	0.86	5.34	4.53
3	6 × 5 radbas × logsig	×		×		×	×	0.946	0.88	5.24	2.69
4	8 radbas	×		×		×	×	0.707	0.89	6.15	4.96
5	10 × 6 tansig × elliotsig	×		×		×	×	0.039	0.88	4.41	1.12
Average									**0.87**	**5.2**	**3.24**

TABLE V. RESULTS FROM SCENARIO NO.3

Scenario no.3: $a_1 = 10^{-5}$ *and* $a_2 = 0$										
Run	**Hidden Layers**	**Selected Features**						**Learning Rate**	γ_e/γ	**MRE (%)**	
		1	**2**	**3**	**4**	**5**	**6**			**Test**	**Train**
1	6 × 6 × 5 elliotsig × logsig × radbas	×	×	×	×	×	×	0.658	0.60	4.68	1.86
2	4 × 5 × 2 tansig × radbas × radbas	×		×	×	×	×	0.271	0.84	5.54	3.29
3	5 × 7 × 5 logsig × logsig × radbas	×	×	×		×	×	0.315	0.69	4.65	1.53
4	10 × 4 tansig × logsig	×		×		×	×	0.256	0.72	4.76	2.28
5	15 × 5 × 5 logsig × radbas × elliotsig	×	×	×	×	×	×	0.956	0.46	6.03	1.19
Average									**0.66**	**5.13**	**2.03**

E. Reflections on Features Selection

As for the selected features, most networks in the previously shown runs eliminated features (2) (supplied voltage) and (4) (power factor), which implies that these features did not contribute significantly to the output and the model could be simplified by considering features (1 – Energy Load), (3 – Supplied Current), (5 – Ambient Temperature) and (6 – Stack Gas Temperature) only.

TABLE VI. RESULTS FROM SCENARIO NO.4

Scenario no.4: $a_1 = a_2 = 0$										
Run	**Hidden Layers**	**Selected Features**						**Learning Rate**	γ_e/γ	**MRE (%)**	
		1	**2**	**3**	**4**	**5**	**6**			**Test**	**Train**
1	13 × 5 × 15 elliotsig × radbas × elliotsig	×		×	×	×	×	0.980	0.34	5.61	2.42
2	13 × 5 × 10 elliotsig × radbas × elliotsig	×		×	×	×	×	0.981	0.31	5.60	2.37
3	9 × 12 × 13 radbas × tansig × elliotsig	×	×	×	×	×	×	0.904	0.33	4.10	0.76
4	15 × 14 × 5 elliotsig × elliotsig × tansig	×	×	×	×	×	×	0.124	0.28	4.83	0.82
5	9 × 12 × 13 elliotsig × softmax × radbas	×	×	×	×	×	×	0.973	0.25	5.09	1.56
Average									**0.31**	**5.05**	**1.6**

V. COMPARISON WITH OTHER TECHNIQUES

A. Classical GA-Based Architecture Optimization

We use the same objective function weights to run architecture optimization with the classical GA. As for GA operators, we use the typical roulette wheel selection, single-point crossover, and mutation [50] throughout the runs. The same termination criteria described in Section 5 were imposed on the classical GA. Results are shown in Table VII.

TABLE VII. RESULTS FROM THE CLASSICAL GA-BASED RUNS

Classical GA Runs: $a_1 = 0$ *and* $a_2 = 50$										
Run	**Hidden Layers**	**Selected Features**						**Learning Rate**	γ_e/γ	**MRE (%)**	
		1	**2**	**3**	**4**	**5**	**6**			**Test**	**Train**
1	7 × 3 × 6 radbas × logsig × radbas	×				×	×	0.746	0.76	7.30	3.49
2	7 × 5 × 1 radbas × logsig × radbas			×		×	×	0.701	0.88	5.90	2.93
3	6 × 2 × 4 logsig × elliotsig × netinv	×	×				×	0.708	0.95	14.94	12.39
4	7 × 6 tansig × radbas	×		×	×	×	×	0.892	0.73	5.49	2.29
5	7 × 2 × 4 radbas × tansig × elliotsig	×		×	×	×	×	0.900	0.83	4.76	2.07
Average									**0.83**	**7.68**	**4.63**

Fig. 4 is a chart comparing the two algorithms using the relevant parameters. In terms of the quality of the obtained architectures, the QGA managed to yield networks with better generalization performance, as seen from the values of the mean relative error (MRE) over testing data – a reduction of 34%. For network complexity, the two algorithms yielded similar results with the average ratios of effective to total parameters being almost the same, although the architectures obtained by the QGA (Table IV) were simpler as they consisted of a maximum of two hidden layers, unlike the mostly three layered networks obtained by the classical GA (Table VII). This suggests that the basic GA frequently fell in the local optima of the performance vs. architecture surface, and that the QGA was more capable of averting these local optima. This was probably achieved through the QGA's perturbation operator and the qubit measurement process which allowed for more diversity in the individuals created at each generation.

As for the performance of the algorithm itself, the classical GA required, on average, less generations to converge than the QGA. However, the classical GA took almost twice as much time to converge as the QGA. This is probably related to the local optima problem described earlier, as the GA may have had a tendency to create more complex architectures throughout the generations, whose training took significantly more time than the simpler counterparts, and as such resulted in a larger total run time for the algorithm.

B. ML-based Models

We choose to run (5) (10×6-tansig×elliotsig) from Scenario 2 as the best-performing QGA model and we compare it to a shallow one-layered ANN frequently used in the literature, a support vector regression (SVR) model with a Gaussian kernel, and a typical radial basis function network (RBN).

We employed a typical hyperbolic tangent function and we varied the number of neurons in the one-layered ANN. Results are shown in Fig. 5. Thirteen neurons in the hidden layer resulted in the best performance for the traditional ANN model. Fig. 6 shows a comparison in the performance with the QGA-based model and other traditional ones. The QGA-ANN exhibited the best performance with a 40% reduction in the mean relative error over the shallow ANN and the SVR-

Gauss models, and almost a 75% reduction in the mean relative error compared with the normal RBN.

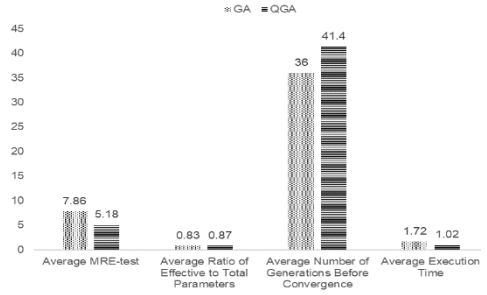


Fig. 4. Comparison of GA-based and QGA-based architecture optimization.

The QGA-ANN has been able to circumvent this issue by virtue of the complexity-penalizing terms present in the QGA's objective function, thus allowing the user to obtain a network with a good generalization ability. In addition, feature selection reduced the input space by two dimensions allowing the network to perform well despite the scarcity of training data points

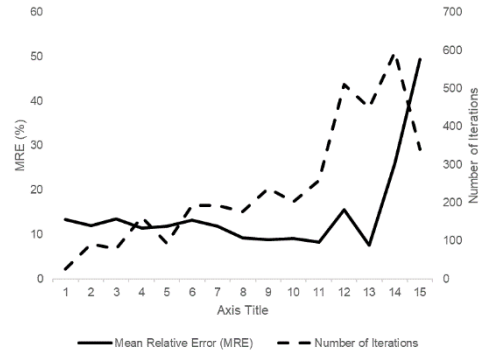


Fig. 5. Mean relative error over testing data and number of training iterations vs. number of neurons in the hidden layer for the shallow ANN

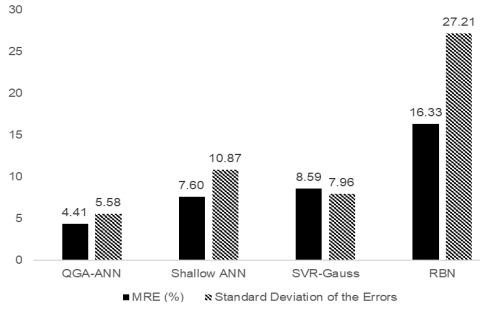


Fig. 6. Means and standard deviations of the relative errors exhibited by each of the considered models over testing data.

VI. Conclusion

We investigated the application of the QGA to the ANN architecture optimization problem and constructed a PEMS for a diesel engine used to power the campus of AUB. The QGA's objective function penalized for generalization performance through the MSE over testing data, and for network complexity using the total number of weight elements and the ratio of the number of the network's effective parameters to that of the total one. We found that the latter term was enough to yield highly performing yet simple networks compared to the case where only the total number of parameters was used as a complexity penalty.

VII. References

[1] J. Mohammadhassani, A. Dadvand, S. Khalilarya, and M. Solimanpur, "Prediction and reduction of diesel engine emissions using a combined ANN–ACO method," Applied Soft Computing, vol. 34, pp. 139–150, 2015.

[2] E. Vanderhaegen, M. Deneve, H. Laget, N. Faniel, and J. Mertens, "Predictive Emissions Monitoring Using a Continuously Updating Neural Network," in Volume 2: Combustion, Fuels and Emissions, Parts A and B, Glasgow, UK, 2010, pp. 769–775, doi: 10.1115/GT2010-22899.

[3] K. Smith and D. Cole, "Software vs. hardware approach to emissions monitoring," in 2015 61st IEEE Pulp and Paper Industry Conference (PPIC), Milwaukee, WI, USA, 2015, pp. 1–6, doi: 10.1109/PPIC.2015.7165868.

[4] A. L. Ahmad, I. A. Azid, A. R. Yusof, and K. N. Seetharamu, "Emission control in palm oil mills using artificial neural network and genetic algorithm," Computers & Chemical Engineering, vol. 28, no. 12, pp. 2709–2715, Nov. 2004, doi: 10.1016/j.compchemeng.2004.07.034.

[5] T. Boningari and P. G. Smirniotis, "Impact of nitrogen oxides on the environment and human health: Mn-based materials for the NO x abatement," Current Opinion in Chemical Engineering, vol. 13, pp. 133–141, Aug. 2016, doi: 10.1016/j.coche.2016.09.004.

[6] H. Michiels, I. Mayeres, L. Int Panis, L. De Nocker, F. Deutsch, and W. Lefebvre, "PM2.5 and NOx from traffic: Human health impacts, external costs and policy implications from the Belgian perspective," Transportation Research Part D: Transport and Environment, vol. 17, no. 8, pp. 569–577, Dec. 2012, doi: 10.1016/j.trd.2012.07.001.

[7] O. Skreiberg, J. E. Hustad, E. I. Garnaes, P. Reinermann, and T. Bjorge, "COMPARING EMPIRICAL, STATISTICAL, AND NEURAL NETWORK MODELS CALCULATING NOx EMISSIONS FROM GAS TURBINES," Inter J Ener Clean Env, vol. 8, no. 3, pp. 221–237, 2007, doi: 10.1615/InterJEnerCleanEnv.v8.i3.30.

[8] G. Ferretti and L. Piroddi, "Estimation of NOx Emissions in Thermal Power Plants Using Neural Networks," Journal of Engineering for Gas Turbines and Power, vol. 123, no. 2, pp. 465–471, Apr. 2001, doi: 10.1115/1.1367339.

[9] E. Ikonen, K. Najim, and U. Kortela, "Neuro-fuzzy modelling of power plant flue-gas emissions," Engineering Applications of Artificial Intelligence, vol. 13, no. 6, pp. 705–717, Dec. 2000, doi: 10.1016/S0952-1976(00)00054-3.

[10] E. Pathmanathan, R. Ibrahim, and V. S. Asirvadam, "CO2 emission model development employing particle swarm optimized — Least squared SVR (PSO-LSSVR) hybrid algorithm," in 2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012), Kuala Lumpur, Malaysia, 2012, pp. 137–142, doi: 10.1109/ICIAS.2012.6306175.

[11] S. M. Zain and Kien Kek Chua, "Development of a neural network Predictive Emission Monitoring System for flue gas measurement," in 2011 IEEE 7th International Colloquium on Signal Processing and its Applications, Penang, Malaysia, 2011, pp. 314–317, doi: 10.1109/CSPA.2011.5759894.

[12] H. Zhou, J. Pei Zhao, L. Gang Zheng, C. Lin Wang, and K. Fa Cen, "Modeling NOx emissions from coal-fired utility boilers using support vector regression with ant colony optimization," Engineering Applications of Artificial Intelligence, vol. 25, no. 1, pp. 147–158, Feb. 2012, doi: 10.1016/j.engappai.2011.08.005.

[13] K. K. Botros and M. Cheung, "Neural Network Based Predictive Emission Monitoring Module for a GE LM2500 Gas Turbine," in 2010 8th International Pipeline Conference, Volume 3, Calgary, Alberta, Canada, 2010, pp. 77–87, doi: 10.1115/IPC2010-31016.

[14] W. G. Lamont, M. Roa, and R. P. Lucht, "Application of Artificial Neural Networks for the Prediction of Pollutant Emissions and Outlet Temperature in a Fuel-Staged Gas Turbine Combustion Rig," in Volume 4A: Combustion, Fuels and Emissions, Düsseldorf, Germany, 2014, p. V04AT04A002, doi: 10.1115/GT2014-25030.

[15] Ligang Zheng, Shuijun Yu, and Minggao Yu, "Prediction of nitrogen oxides from coal combustion by using response surface methodology," in 2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring, Hunan, 2012, pp. 508–511, doi: 10.1109/CDCIEM.2012.126.

[16] R. M. Durão, M. T. Mendes, and M. João Pereira, "Forecasting O 3 levels in industrial area surroundings up to 24 h in advance, combining classification trees and MLP models," Atmospheric Pollution

Research, vol. 7, no. 6, pp. 961–970, Nov. 2016, doi: 10.1016/j.apr.2016.05.008.

[17] W. K. Yap and V. Karri, “Emissions predictive modelling by investigating various neural network models,” Expert Systems with Applications, vol. 39, no. 3, pp. 2421–2426, Feb. 2012, doi: 10.1016/j.eswa.2011.08.091.

[18] F. Maroteaux and C. Saad, “Combined mean value engine model and crank angle resolved in-cylinder modeling with NOx emissions model for real-time Diesel engine simulations at high engine speed,” Energy, vol. 88, pp. 515–527, Aug. 2015, doi: 10.1016/j.energy.2015.05.072.

[19] R. Finesso and E. Spessa, “A real time zero-dimensional diagnostic model for the calculation of in-cylinder temperatures, HRR and nitrogen oxides in diesel engines,” Energy Conversion and Management, vol. 79, pp. 498–510, Mar. 2014, doi: 10.1016/j.enconman.2013.12.045.

[20] S. d’Ambrosio, R. Finesso, L. Fu, A. Mittica, and E. Spessa, “A control-oriented real-time semi-empirical model for the prediction of NOx emissions in diesel engines,” Applied Energy, vol. 130, pp. 265–279, Oct. 2014, doi: 10.1016/j.apenergy.2014.05.046.

[21] J. Asprion, O. Chinellato, and L. Guzzella, “Optimisation-oriented modelling of the NOx emissions of a Diesel engine,” Energy Conversion and Management, vol. 75, pp. 61–73, Nov. 2013, doi: 10.1016/j.enconman.2013.05.039.

[22] J. Asprion, O. Chinellato, and L. Guzzella, “A fast and accurate physics-based model for the NOx emissions of Diesel engines,” Applied Energy, vol. 103, pp. 221–233, Mar. 2013, doi: 10.1016/j.apenergy.2012.09.038.

[23] C. Sayin, H. M. Ertunc, M. Hosoz, I. Kilicaslan, and M. Canakci, “Performance and exhaust emissions of a gasoline engine using artificial neural network,” Applied Thermal Engineering, vol. 27, no. 1, pp. 46–54, Jan. 2007, doi: 10.1016/j.applthermaleng.2006.05.016.

[24] K. I. Wong, P. K. Wong, C. S. Cheung, and C. M. Vong, “Modelling of diesel engine performance using advanced ML methods under scarce and exponential data set,” Applied Soft Computing, vol. 13, no. 11, pp. 4428–4441, Nov. 2013, doi: 10.1016/j.asoc.2013.06.006.

[25] N. Shrivastava and Z. Mohd. Khan, “Application of Soft Computing in the Field of Internal Combustion Engines: A Review,” Arch Computat Methods Eng, vol. 25, no. 3, pp. 707–726, Jul. 2018, doi: 10.1007/s11831-017-9212-9.

[26] E. Arcaklioğlu and İ. Çelıkten, “A diesel engine’s performance and exhaust emissions,” Applied Energy, vol. 80, no. 1, pp. 11–22, Jan. 2005, doi: 10.1016/j.apenergy.2004.03.004.

[27] T. F. Yusaf, D. R. Buttsworth, K. H. Saleh, and B. F. Yousif, “CNG-diesel engine performance and exhaust emission analysis with the aid of artificial neural network,” Applied Energy, vol. 87, no. 5, pp. 1661–1669, May 2010, doi: 10.1016/j.apenergy.2009.10.009.

[28] S. Dharma et al., “Experimental study and prediction of the performance and exhaust emissions of mixed Jatropha curcas-Ceiba pentandra biodiesel blends in diesel engine using artificial neural networks,” Journal of Cleaner Production, vol. 164, pp. 618–633, Oct. 2017, doi: 10.1016/j.jclepro.2017.06.065.

[29] S. Javed, Y. V. V. Satyanarayana Murthy, R. U. Baig, and D. Prasada Rao, “Development of ANN model for prediction of performance and emission characteristics of hydrogen dual fueled diesel engine with Jatropha Methyl Ester biodiesel blends,” Journal of Natural Gas Science and Engineering, vol. 26, pp. 549–557, Sep. 2015, doi: 10.1016/j.jngse.2015.06.041.

[30] H. Taghavifar, H. Taghavifar, A. Mardani, A. Mohebbi, S. Khalilarya, and S. Jafarmadar, “Appraisal of artificial neural networks to the emission analysis and prediction of CO2, soot, and NOx of n-heptane fueled engine,” Journal of Cleaner Production, vol. 112, pp. 1729–1739, Jan. 2016, doi: 10.1016/j.jclepro.2015.03.035.

[31] S. Lotfan, R. A. Ghiasi, M. Fallah, and M. H. Sadeghi, “ANN-based modeling and reducing dual-fuel engine’s challenging emissions by multi-objective evolutionary algorithm NSGA-II,” Applied Energy, vol. 175, pp. 91–99, Aug. 2016, doi: 10.1016/j.apenergy.2016.04.099.

[32] H. Bendu, B. B. V. L. Deepak, and S. Murugan, “Multi-objective optimization of ethanol fuelled HCCI engine performance using hybrid GRNN–PSO,” Applied Energy, vol. 187, pp. 601–611, Feb. 2017, doi: 10.1016/j.apenergy.2016.11.072.

[33] B. Liu, J. Hu, F. Yan, R. F. Turkson, and F. Lin, “A novel optimal support vector machine ensemble model for NOX emissions prediction of a diesel engine,” Measurement, vol. 92, pp. 183–192, Oct. 2016, doi: 10.1016/j.measurement.2016.06.015.

[34] X. Niu, C. Yang, H. Wang, and Y. Wang, “Investigation of ANN and SVM based on limited samples for performance and emissions prediction of a CRDI-assisted marine diesel engine,” Applied Thermal Engineering, vol. 111, pp. 1353–1364, Jan. 2017, doi: 10.1016/j.applthermaleng.2016.10.042.

[35] Z. Liu and S. Fei, "Study of CNG/diesel dual fuel engine's emissions by means of RBF neural network", Journal of Zhejiang University SCIENCE, vol. 5, no. 8, pp. 960-965, 2004. Available: 10.1631/jzus.2004.0960.

[36] M. Azzam, M. Awad, and J. Zeaiter, “Application of evolutionary neural networks and support vector machines to model NOx emissions from gas turbines,” Journal of Environmental Chemical Engineering, vol. 6, no. 1, pp. 1044–1052, Feb. 2018, doi: 10.1016/j.jece.2018.01.020.

[37] V. K. Ojha, A. Abraham, and V. Snášel, “Metaheuristic design of feedforward neural networks: A review of two decades of research,” Engineering Applications of Artificial Intelligence, vol. 60, pp. 97–116, Apr. 2017, doi: 10.1016/j.engappai.2017.01.013.

[38] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesus, Neural network design, 2nd edition. Wrocław: Amazon Fulfillment Poland Sp. z o.o.

[39] S. Curteanu and H. Cartwright, “Neural networks applied in chemistry. I. Determination of the optimal topology of multilayer perceptron neural networks: The optimal topology of neural networks,” J. Chemometrics, vol. 25, no. 10, pp. 527–549, Oct. 2011, doi: 10.1002/cem.1401.

[40] M. Awad and R. Khanna, Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. Berkeley, CA: Apress, 2015.

[41] H. Niska, T. Hiltunen, A. Karppinen, J. Ruuskanen, and M. Kolehmainen, “Evolving the neural network model for forecasting air pollution time series,” Engineering Applications of Artificial Intelligence, vol. 17, no. 2, pp. 159–167, Mar. 2004, doi: 10.1016/j.engappai.2004.02.002.

[42] A. R. Carvalho, F. M. Ramos, and A. A. Chaves, “Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem,” Neural Comput & Applic, vol. 20, no. 8, pp. 1273–1284, Nov. 2011, doi: 10.1007/s00521-010-0504-3.

[43] P. G. Benardos and G.-C. Vosniakos, “Optimizing feedforward artificial neural network architecture,” Engineering Applications of Artificial Intelligence, vol. 20, no. 3, pp. 365–382, Apr. 2007, doi: 10.1016/j.engappai.2006.06.005.

[44] R. Lahoz-Beltra, “Quantum Genetic Algorithms for Computer Scientists,” Computers, vol. 5, no. 4, p. 24, Oct. 2016, doi: 10.3390/computers5040024.

[45] J.-C. Lee, W.-M. Lin, G.-C. Liao, and T.-P. Tsao, “Quantum genetic algorithm for dynamic economic dispatch with valve-point effects and including wind power system,” International Journal of Electrical Power & Energy Systems, vol. 33, no. 2, pp. 189–197, Feb. 2011, doi: 10.1016/j.ijepes.2010.08.014.

[46] P. C. Li, K. P. Song, and F. H. Shang, “Double chains quantum genetic algorithm with application to neuro-fuzzy controller design,” Advances in Engineering Software, vol. 42, no. 10, pp. 875–886, Oct. 2011, doi: 10.1016/j.advengsoft.2011.06.006.

[47] F. Zitouni and R. Maamri, “Cooperative Learning-Agents for Task Allocation Problem,” in Interactive Mobile Communication Technologies and Learning, vol. 725, M. E. Auer and T. Tsiatsos, Eds. Cham: Springer International Publishing, 2018, pp. 952–968.

[48] R. Bangroo, N. Kumar, and R. Sharma, “A Model for Multi-processor Task Scheduling Problem Using Quantum Genetic Algorithm,” in Hybrid Intelligent Systems, vol. 734, A. Abraham, P. Kr. Muhuri, A. K. Muda, and N. Gandhi, Eds. Cham: Springer International Publishing, 2018, pp. 126–135.

[49] B. Han, J. Jiang, Y. Gao, and J. Ma, “A Quantum Genetic Algorithm to Solve the Problem of Multivariate,” in Information Computing and Applications, vol. 243, C. Liu, J. Chang, and A. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 308–314.

[50] M. Mitchell, An introduction to genetic algorithms, 7. print. Cambridge, Mass., 2001.