Solving Satisfiability Logic Programming Using Radial Basis Function Neural Networks

Nawaf Hamadneh

College of Science and Theoretical Studies, Saudi Electronic University, Riyadh 11673, Saudi Arabia, nhamadneh@seu.edu.sa

Saratha Sathasivam

School of Mathematical Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia, saratha@usm.my

Abstract

We proposed a new technique to solve QBF based on Radial basis function neural networks (RBFNNs) and Prey-Predator algorithm (PPA). Prey-Predator algorithm (PPA) is a neural learning algorithm used to determine the parameters of the networks. We built the neural networks to represent the logic programming in Conjunctive Normal Form (CNF), which has at most three variables in each clause (3-CNF). Then, these neural networks are developed to be recurrent neural networks to deal with universal variables in QBF problems. The neural networks models can be applied to solve a wide range of practical applications of Satisfiability logic programming, such as NP-complete decision problem, and computer network design.

Keywords: logic programming; Satisfiability; Radial basis function neural network, Prey-Predator algorithm

1. Introduction

Logic programming is used to describe relations, and have both a declarative and an operational meaning [1,2]. The general objective of artificial neural network is stored experiential knowledge and makes it available for use, similar to the brain. It acquires knowledge through the learning process, and use the power of communication between artificial neurons to store knowledge. The advantages and the importance of the RBFNNs come from that they have a simple topological structure , faster learning algorithms, high approximation capability, and have been widely applied in many science and engineering fields [3-5].

Prey-Predator algorithm (PPA) is a new metaheuristic algorithm, introduced by Tilahun and Ong (2012) [6]. PPA is developed for optimization problems such as particle swarm optimization algorithm [5,7,8]. The algorithm is a more general algorithm where some of well-known algorithms, including simulated annealing, particle swarm optimization, firefly algorithm and evolutionary strategy, become a special case [7,9] . It is inspired by how the predators run after their prey and how the preys try to survive in the ecosystem.

A proportional logic programming is a set of axioms, clauses, or rules which in turn consist of Boolean variables (literals), i.e. atoms and negated atoms only (negation is denoted by $\neg$) [2,10,11]. Quantified Boolean formula or QBF) is an extension of propositional logic programming in Conjunctive Normal Form (CNF) [1,11,12]. QBF is a proportional logic programming with existential quantification and universal quantification. The algorithms which are used to check the stability of the logic programming are called Satisfiability or SAT solvers [13]. In practice, SAT is fundamental in solving many problems, such as, circuit design, and computer network design. The simplest SAT solver is called the truth-table method. It constructs the full table, which will have 2^n rows when P has n variables, and report whether the final column, representing P has value 1 in any row. Davis Putnam Logemann Loveland (DPLL) algorithm [14] works on the principle of assigning variables, simplifying the formula to account for that assignment and then recursively solving the simplified formula. Zhang and Malik have developed a new method for evaluating QBF [1]. This method is based on the DPLL algorithm. Their experiments show that conflict driven learning strategies, when adapted in a QBF solver, can speed up the search process vastly. But still the most used strategy for SAT solvers.

In this paper, we presented a novel preprocessing technique for QBF based on RBFNNs Prey-Predator algorithm (PPA-RBFNNs), with commercial software MATLAB®. It is the extension of that conference paper with additional of discussion [15]. PPA is selected on the basis that it is a new generalized algorithm and gives promising results when tested on benchmark problems as well as modeled transportation and engineering problems [6]. The outline of the paper is as follows. In section 2 a brief introduction on radial basis function neural network will be discussed followed by pray predator algorithm in section 3. In section 4, Logic programming, QBF solvers and first order logic programming are discussed. In section 5, 3-QBF solvers based on RBFNNs and PPA is proposed with details. At the end there will be a conclusion in section 6.

1. Radial basis function neural networks

A RBFNN in its form consists of three layers, namely, input layer, hidden layer, and the output layer. Each hidden neuron implements a nonlinear activation function and each output neuron implements a weighted sum of hidden neuron outputs. The activation functions in the hidden layer, such as Gaussian function, are also called radial basis function. RBFNN has typically three layers [16,17]: namely, an input layer, a hidden layer [18] and a linear output layer, as shown in Fig. 1. The hidden layer neurons receive the input information, followed by certain decomposition, extraction, and transformation steps to generate the output information. RBFNNs with good specification should have less hidden units and high prediction accuracy. So, the effective number of the hidden neurons is a relatively small number.

Pursuant to that, each hidden neuron is allocated to respond to each of sub-spaces of the input regions, formed by the clusters of training samples [19-21]. The advantages and the importance of the RBFNNs comes from that they have a simple topological structure and their ability to reveal how learning proceeds in an explicit manner.

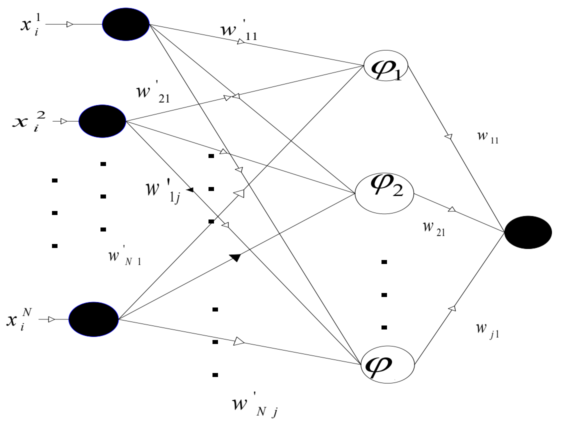

Fig. 1: Structure of a radial basis function network

The output neuron L represents a map F that satisfies the interpolation condition

$$F_L(x_i) = d_{iL},\ \mathrm{i} = 1, \dots, R \quad (1)$$

Where:

- $\{x_i \in \Re^N, i = 1, \dots, R\}$ is the input data set,
- $\{d_{iL} \in \Re,\ i = 1, \dots, R\}$ is a corresponding output target set in the output neuron L.
- $\{F_L(x_i) : \Re^N \to \Re,\ i = 1, \dots, R\}$ is a corresponding output actual set in the output neuron L.
- The interpolating function F_L has to pass through all the training data points. Accordingly, the radial basis function neural network technique has the following form
- $$F_L(x) = \sum_{m=1}^{J} w_{mL}\varphi_m(x) \quad (2)$$

Where:

- $F_L(x)$ is the actual output value of the output neuron L which corresponds to the input value $x \in \Re^N$.
- w_{mL} is the output weight between the hidden neuron m and the output neuron L.
- φ_m is the activation function in hidden neuron m.
- J is the number of hidden neurons.

The following equation gives the activation function (Gaussian function) which is used in RBFNN [18].

$$\varphi_m(x) = e^{-\frac{(\sum_{h=1}^{N} w'_{hm} * x^h - c_m)^2}{2\sigma_m^2}} \quad (3)$$

Where:

- φ_m is the radial basis function in hidden neuron m .
- c_m and σ_m^2 are the center and the width of the hidden neuron m respectively**.**
- $x = (x^1, x^2, \ldots, x^N)$ is a value entered in the input layer.
- w'_{hm} is the constant input weight between the input neuron h and the hidden neuron m .

The training sets in RBFNN is labeled pairs $\{(x_i, d_i),\ i = 1, \ldots, R\}$, where R is the number of data sets [16,18,22]. In other words, it represents associations for a given finite set of input–output data. As mention above, d_i is the target output value which corresponds to the input data x_i . The actual output value is represented by the map F_L that is encoded with the binary values $\{0,1\}$. 0 and 1 are used to emphasize "false" and "true" respectively. The difference between the actual values and the output target values can be obtained by using a differentiable function, such as the sum of squared error function [23].

2. Prey-Predator Algorithm

Two important areas in neural networks are optimization and validation. On the optimization aspect, the efforts are directed towards building networks that are efficient and fast. On the validation aspect, the networks need to be functionally correct.

Prey-Predator algorithm (PPA) is a new metaheuristic algorithm introduced by Tilahun and Ong [5,6,24] for optimization problems. It is inspired by the interaction between a predator and preys of animals in the ecosystem. Randomly generated solutions will be assigned as a predator and preys depending on their performance on the objective function. A solution with least performance will be assigned as a predator and the others preys. A prey with better performance of the objective function will be called best prey. After the assignment of predator and preys, the preys will run away from the predator and follow preys with better performance. The predator does the exploration by running randomly and chasing the prey with least performance. The best prey in the other hand does only a local search for exploitation purpose.

PPA is a more general algorithm which will coverage to other algorithms under different values for the algorithm parameters. It has also been tested on different problems including public bus timetabling and gives promising results [6]. PPA is effective and suitable for use in many areas of science and engineering [5,25]

3. Logic programming and the Satisfiability

4.1 Logic programming

Logic programming seems to use like the relational database, and certainly has several properties because the knowledge about it is easy to change [1,2,12]. Propositional logic programming is built up from propositional variables (Boolean variables) through the use of the Boolean connectives ($\wedge, \vee, \leftarrow, \neg$), where the Propositional variables can be assigned by values: True or False. A proportional logic program consists of a set of logic clauses each one of the forms:

$$\overset{k}{\underset{i=1}{\vee}} \mathrm{A}_i \leftarrow \overset{r}{\underset{j=1}{\wedge}} \neg B_j \overset{n}{\underset{j=r+1}{\wedge}} \mathrm{B}_j \qquad (4)$$

where

$k, r, n \in \mathbf{Z}^+$. $\forall A_i$ and $\forall B_j$ are atoms. The arrow may be read "if", the symbols $\vee$ and $\wedge$ read "or "and" and" respectively.

Clauses can be either represented in Disjunctive (logic **or**) Normal Form (DNF) or Conjunctive (logic **and**) Normal Form (CNF), which is widely being used to represent clauses. A clause is in CNF if it is in form $\overset{n}{\underset{i=1}{\vee}} A_i, n \in \mathbf{Z}^+$, where A_i a literal. So, a logic programming is in DNF if and only if has the form $P = \overset{n}{\underset{i=1}{\wedge}} F_i, n \in \mathbf{Z}^+$, where F_i is a clause in CNF.

A logic programming is in the 3-CNF if each clause consists of three literals as maximum. 3-satisfiability problem or 3-SAT problem is a mapping problem from a logic programming in 3-conjunctive normal form (3-CNF) to "truth values" (1 and 0), which refer to true and false respectively. A logic programming P is satisfiable, if and only if there is a substitution of "truth values" for its literals that makes it true.

Quantified Boolean formula (QBF) is a proportional logic programming with existential quantification and universal quantification. However, propositional logic programming is equivalent to QBF with only existentially quantified variables. QBF can be used to solve many practical problems ranging from artificial intelligent (AI) planning. QBF [26,27] has the form: $\overset{\leftarrow}{q} F$, where F is a propositional logic programming expressed in CNF, and $\overset{\leftarrow}{q}$ is a sequence of universal quantifier ($\forall$) and existential Quantifier ($\exists$). For example, $\forall x \exists y \ Like(x, y)$ means ʻʼ for all x, there exist at least one y like x". We can replace the positions of the same type of the quantifiers without affecting the truth value, while the positions of different types of quantifiers cannot be switched. For example $\forall x \ \exists y \ greater(x, y)$ is not equivalent to $\exists y \ \forall x \ greater(x, y)$. Then $\forall x \exists y \ greater(x, y)$ reads "for every number x, there is a number y that is greater than x", which is true, while $\exists y \ \forall x \ greater(x, y)$ reads "there is a number **y** that is greater than any number", which is not true.

First-Order Logic extends Propositional Logic with predicates, functions, and quantifiers [11]. Predicates have a value of true or false, where it can take arguments, which are terms. On the other hand, the functions are given values in the logic programming. The terms evaluate to values other than truth values such as integers, real numbers, and functions applied to variables and constants. For example, the sentence (Anyone succeed in computer science exams and winning the lottery is happy) is represented in First-Order Logic (FOL), as

$$\forall x \ Succeed(Computers, x) \wedge Win(Lottery, x) \rightarrow Happy(x) \quad (5)$$

4.2 Satisfiability

A propositional logic programming is said to be satisfiable if there is an assignment of truth values to its literals in a way that makes the programming true. Satisfiability or SAT problem is an example of NP-complete problems. NP-complete problems are decision problems that have only two outputs values, which are 1 and 0, which refer to true and false respectively. SAT can be used to solve many practical problems. The logic programming below represents the general form of 3-SAT Boolean formula that consists of (N) clause; each one consists of three literals. Note that, each literal is either an atom or negated atom.

$$\begin{aligned} P &= (A_{11} \vee A_{12} \vee A_{13}) \wedge \\ &(A_{21} \vee A_{22} \vee A_{23}) \wedge \\ &\vdots \\ &(A_{N1} \vee A_{N2} \vee A_{N3}) \end{aligned} \quad (6)$$

For example, the following below is an unsatisfiable *3-SAT* logic programming with 12 literals, 5 clauses, and 3 variables. This logic programming can be seen to be unsatisfiable by inspection: the first and second clause requires (respectively) that at least one variable is true and at least one variable is false, whereas the last three clauses (together) require that all variables have the same truth value.

$$P = (\neg A \vee \neg B \vee \neg C) \wedge (A \vee B \vee C) \wedge (A \vee \neg B) \wedge (C \vee \neg A) \quad (7)$$

QBF solvers are used to answer the question of whether or not $\overset{\leftarrow}{q} F$ is true or false. The satisfiability problem (*SAT*) is *QBF* solvers under the restriction all variables are existential. *QBF* solver is slower than SAT because; the search must solve both settings of every universal variable. A formula $F[T/A]$ is obtained by setting variable A to the value 1 (True) in F. A formula $\exists A F$ is true if and only if at least one of $F[T/A]$ and $F[\perp/A]$ is true. Similarly, $F[\perp/A]$ is obtained by setting variable A to the value 0 (False) in F. And also, a formula $\forall A P$ is true if and only if $F[T/A]$ and $F[\perp/A]$ are true. To determine $F[T/A]$ in CNF, we have two steps

- Removing all the clauses in logic programming F that contain *A*.
- Deleting $\neg A$ from all the clauses that contain $\neg A$.

The simplest QBF solver is called semantic tree method [28]. This method is a very well-known method in logic. The method is an automatic method of semantic analysis, which consists of determining the logical values of sub formulas of a given formula. The evaluation of the following QBF, within the semantic tree method, is illustrated in Fig. 2. $P = \exists A_1 \exists A_2 \forall A_3 ((A_2 \leftarrow A_1) \wedge (A_2 \leftarrow A_3))$ is satisfiable if and only if both of $F[T/A]$ and $F[\perp/A]$ are true.

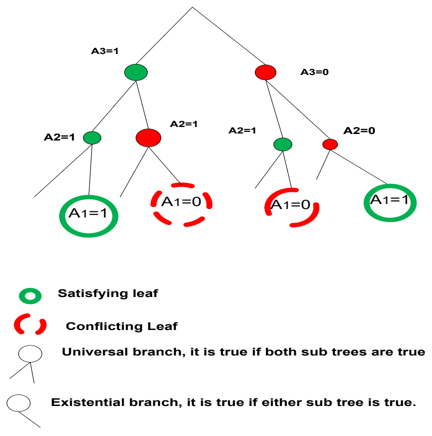
A3=1
A3=0
A2=1
A2=1
A2=1
A2=0
A1=1
A1=0
A1=0
A1=1
Satisfying leaf
Conflicting Leaf
Universal branch, it is true if both sub trees are true
Existential branch, it is true if either sub tree is true.

Fig. 2: A semantic tree proof of the stability of

$P = \exists A_1 \exists A_2 \forall A_3 ((A_2 \leftarrow A_1) \wedge (A_2 \leftarrow A_3))$.

In AI, a proof must always be built from a fixed set of inference rules. DPLL algorithm has been used widely as a complete (it finds a solution if one exists; otherwise correctly says that no solution exists) and efficient procedure to solve SAT [14]. The basic idea of the SAT-DPLL algorithm is:

i. Given a logic programming P in CNF.

ii. Assign a truth value for a variable.

iii. Find the set of all unit clauses created from step ii. A unite clause is a clause which has exactly one literal, which is still unassigned. Note that the literals assign the needed value.

iv. Iteratively retry step ii until there is no change (found transitive closure) If the current assignment cannot yield true for all clauses - fold back from recursion and retry a different assignment else - "guess" another variable (recursively invoke and return to 1.

4. 3-QBF solvers based on RBFNNs and PPA

Deciding the stability of QBF problem is an extension of the Boolean Satisfiability problem. So, Boolean Satisfiability problem (SAT) is QBF under the restriction all variables are existential. QBF problem is in practice much harder to solve than SAT. In this paper, we present a new algorithm used to check the stability of logic programming. Firstly, we used it as a SAT solver, and then we used it as a QBF solver. The new algorithm is based on radial basis function neural networks. The following is the outline of the new SAT algorithm, which is based on RBFNNs (SAT-RBFNNs). After that, we developed it to be 3SAT-RBFNNs. This is done by reduction the logic programming, for example, the reduction of a formula F by a literal A is denoted by $F[T/A]$ or $F[\perp/A]$. To determine $F[T/A]$ in CNF, we have two steps; removing all the clauses in logic programming F that contain A, and also, deleting $\neg A$ from all the clauses in F. After that, we extended the algorithm to use as a QBF solver. In this case, we used recurrent radial basis function neural networks (RRBFNNs) to deal with universal variables. The steps below are the outline of SAT-RBFNNs algorithm.

i) Given a logic programming in CNF.

ii) Calculate the training data for each clause in the logic programming [29]. We used the following equation to determine the training data for each clause.

$$F_Z(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m) = \sum_{i=1}^{n} A_i - \sum_{i=1}^{m} B_i \qquad (8)$$

where $\{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m\}$ are all literals in clause Z and $\{A_1, A_2, ..., A_n\}$ is the set of atoms, while $\{B_1, B_2, ..., B_m\}$ is the set of negated atoms. F_Z is the actual output data which corresponding to $\{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m\}$. We have replaced $\perp$ and T by 0 and 1 respectively, to emphasize "false" and "true".

iii) Building a RBFNN initial structure represents the logic programming. We identify an input neuron for each variable, and output neuron for each clause.

iv) Obtain parameters of the RBFNN by using a neural learning algorithm, such as PPA.

The 3SAT-RBFNNs can be optimized as follows

i) Given a logic programming in CNF.

ii) Convert the logic programming into 3-CNF. The reduction of a formula F by a literal A is denoted by $F[T / A]$ or $F[\perp / A]$.

iii) Building a RBFNN initial structure represents the logic programming. We identify an input neuron for each variable, and output neuron for each clause.

iv) Since the logic programming is in 3-CNF, the values of the parameters in the RBFNN can be obtained from Table 1.

The semantics of a QBF can be defined recursively in the following way:

- If F is the empty set of clauses then $\overleftarrow{q} F$ is true.
- If F contains an empty clause then $\overleftarrow{q} F$ is false.
- A formula $\forall A F$ is true if and only if $F[T / A] \wedge F[\perp / A]$ are true.
- A formula $\exists A F$ is true if and only if $F[T / A] \vee F[\perp / A]$ is true.

The main difference between SAT-solver and QBF-solver is with the solutions of QBF in order to verify both settings of each universally quantified variable. So, to use SAT-solver to solve the QBF problem, we reduce a formula by universal variables. In this case, the algorithm is called QBF-RBFNNs.

The following is the outline of the new 3QBF-RBFNNs algorithm:

i) Given a logic programming in CNF.

ii) Reduction of a formula F by universal literals.

iii) Convert the logic programming into 3-CNF.

iv) Building a RBFNN initial structure represents the logic programming. We identify an input neuron for each variable, and output neuron for each clause.

v) Since the logic programming is in 3-CNF, the values of the parameters in the RBFNN can be obtained from the Table 1.

Table 1: the parameters of the RBFNNs which represent 3-CNF, by using PPA

3CNF clause	centers: C_1, C_2, C_3	widths $\sigma_1^2, \sigma_2^2, \sigma_3^2$	weight W_1, W_2, W_3	SSE
$A_1 \vee A_2 \vee A_3$	1.0000, 1.9647, 2.9493	0.0276, 0.8861, 0.6650	0.5487, 0.6612, 0.6267	0.0017
$A_1 \vee A_2 \vee \neg A_3$	0.0599, 0.9597, 1.9554	0.0757, 0.9173, 0.6634	0.5644, 0.6508, 0.6301	0.0020
$A_1 \vee \neg A_2 \vee \neg A_3$	0.0607, 0.9600, 1.9557	0.0770, 0.9181, 0.6631	0.5640, 0.6505, 0.6296	0.0020
$\neg A_1 \vee \neg A_2 \vee \neg A_3$	-2.0014, -1.1313, 0.0399	0.0332, 0.9592, 0.6678	0.5051, 0.6669, 0.6654	0.0035

To see the evaluation of

$$F = \exists A_1 \exists A_2 \exists A_3 \forall A_4 ((\neg A_1 \vee A_4 \vee \neg A_2) \wedge (A_2 \vee \neg A_3) \wedge (A_2 \vee A_3) \wedge (A_1 \vee \neg A_4 \vee \neg A_2) \quad (9)$$

We have using the 3QBF-RBFNNs, we follow the following steps.

i) As mentioned, the logic programming should be in CNF. So, the first step is already done, because the formula is in 3-CNF, where each clause have at most 3 literals.

ii) Reduction of a formula F by universal literals, where we have only one universal variable, which is A_4. Accordingly, we determine $F[T / A_4] \wedge F[\perp / A_4]$, because the formula F is true iff $F[T / A_4] \wedge F[\perp / A_4]$ is true. As mentioned, $F[T / A_4]$ is equivalent to F with deleting the clauses which contain A_4, and also deleting $\neg A_4$ from all clauses in F.

5. Conclusions

We presented a new QBF-solver based on RRNFNNs. The new technique is also used as a new SAT-solver. The main importance of this technique is open a new field in computing the stability within the artificial neural networks. This technique is also suitable to use in the first order logic programming. In this study, the key components are RBFNNs, PPA, and 3-CNF. We determine the RBFNNs parameters which represent the 3-SAT, by using PPA. A large number of applications can be represented by the new methods such applications of SAT, planning problems, NP-complete problems and electronic circuits.

Referances

[1]. Zhang L, Malik S. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation; 2006. Springer. pp. 313-318.

[2]. Kowalski R (1979) Algorithm= logic+ control. Communications of the ACM 22: 424-436.

[3]. Wen S, Huang T, Zeng Z, Chen Y, Li P (2015) Circuit design and exponential stabilization of memristive neural networks. Neural Networks 63: pp. 48-56.

[4]. Abdurahman A, Jiang H, Teng Z (2015) Finite-time synchronization for memristor-based neural networks with time-varying delays. Neural Networks 69: pp.20-28.

[5]. Nawaf Hamadneh SLT, Saratha Sathasivam and Ong Hong Choon (2013) Prey-Predator Algorithm as a New Optimization Technique Using in Radial Basis Function Neural Networks. Research Journal of Applied Sciences 8: pp.383-387.

[6]. Tilahun SL (2013) Prey-Predator Algorithm (PPA): A new metaheuristic optimization algorithm Penang, Malaysia: Universiti Sains Malaysia.

[7]. Tilahun SL, Ong HC (2015) Prey-predator algorithm: a new metaheuristic algorithm for optimization problems. International Journal of Information Technology & Decision Making 14: pp.1331-1352.

[8]. Hamadneh N, Khan WA, Sathasivam S, Ong HC (2013) Design optimization of pin fin geometry using particle swarm optimization algorithm. PloS one 8: e66080.

[9]. Tilahun SL, Ngnotchouye JMT, Hamadneh NN (2017) Continuous versions of firefly algorithm: a review. Artificial Intelligence Review: pp.1-48.

[10]. Kowalski R, Logic UoEDoC (1973) Predicate logic as programming language: Edinburgh University.

[11]. Bradley AR, Manna Z (2007) The calculus of computation: decision procedures with applications to verification: Springer.

[12]. Hamadneh N (2013) Logic Programming in Radial Basis Function Neural Networks: Universiti Sains Malaysia.

[13]. Zhang L, Madigan CF, Moskewicz MH, Malik S. Efficient conflict driven learning in a boolean satisfiability solver; 2001. IEEE Press. pp. 279-285.

[14]. Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. Communications of the ACM 5: pp. 394-397.

[15]. Hamadneh N, Sathasivam S, Tilahun SL, Choon OH. Satisfiability of logic programming based on radial basis function neural networks; 2014. pp. 547-550.

[16]. Moody J, Darken J (1989) Fast learning in networks of locally tuned processing units. Neural Computation 1: pp. 281 - 294.

[17]. Hamadneh N, Sathasivam S, Choon OH (2012) Higher order logic programming in radial basis function neural network. Appl Math Sci 6: pp. 115-127.

[18]. Sathasivam S, Hamadneh N, Choon OH (2011) Comparing Neural Networks: Hopfield Network and RBF Network. Applied Mathematical Sciences 5: pp. 3439 - 3452.

[19]. Taghi M, Vakil-Baghmisheh, Pavesic N (2004) Training RBFnetworks with selective backpropagation. Neurocomputing 62: pp. 39 – 36.

[20]. Noman S, Shamsuddin SM, Hassanien AE (2009) Hybrid Learning Enhancement of RBF Network with Particle Swarm Optimization: Springer-Verlag Berlin Heidelberg. pp. 381-397

[21]. Xiaobin L (2009) RBF Neural Network Optimized by Particle Swarm Optimization for Forecasting Urban Traffic Flow. Third International Symposium on Intelligent Information Technology Application: IEEE. pp. 124-127.

[22]. Lowe D (1989) Adaptive radial basis function nonlinearities, and the problem of generalization. First IEE International Conference on artificial neural networks. London. pp. 171 -175.

[23]. Schwenker F, Kestler HA, Palm G (2001) Three learning phases for radial-basis-function networks. Neural Networks 14: pp. 439-458.

[24]. Khan WA, Hamadneh NN, Tilahun SL, Ngnotchouye JMT (2016) A Review and Comparative Study of Firefly Algorithm and its Modified Versions. Optimization Algorithms-Methods and Applications. Rijeka: InTech. pp. Ch.13.

[25]. Abouel-Kasem A, Hassab-Allah I, Nemat-Alla M (2014) Analysis and Design of Viscoelastic Adhesively Bonded Tubular Joint. Journal of Engineering and Applied Sciences 1: pp. 13-23.

[26]. Yu Y, Malik S. Validating the result of a Quantified Boolean Formula (QBF) solver: theory and practice; 2005. ACM. pp. 1047-1051.

[27]. Samulowitz H (2008) Solving Quantified Boolean Formulas: University of Toronto.

[28]. Nilsson NJ (1986) Probabilistic logic. Artificial intelligence 28: pp. 71-87.

[29]. Hamadneh N, Sathasivam S, Tilahun SL, Choon OH (2012) Learning Logic Programming in Radial Basis Function Network via Genetic Algorithm. Journal of Applied Sciences 12: pp. 840-847.