View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Leeds Beckett Repository

A Constraint-based Genetic Algorithm for Optimizing Neural Network Architectures for Detection of Loss of Coolant Accidents of Nuclear Power Plants

David Tian[a], Jiamei Deng[*,a], Gopika Vinod[b], T.V. Santhosh[b], Hissam Tawfik[a]

[a]*School of Computing, Creative Technologies and Engineering, Leeds Beckett University, Leeds, UK, LS1 3HE*
[b]*Reactor Safety Division, Bhabha Atomic Research Centre, Mumbai - 400 085 India*

Abstract

Monitoring the safety of nuclear power plants (NPPs) is a crucial task in nuclear energy industry. One type of severe accidents of a NPP is the loss of coolant accident (LOCA) which can be caused by a large break in the inlet headers (IHs) of the primary heat transport (PHT) of a nuclear reactor. Nowadays, neural networks with 2-hidden layers trained on nuclear simulation transient datasets have been used as a dominant methodology to detect LOCA. However, the performance of the neural networks critically depend on their architectures and it is too costly to find an optimal network architecture by exhaustively training all the architectures. This paper proposes a constraint-based genetic algorithm (GA) to find optimal 2-hidden layer network architectures for detecting LOCA of a NPP. The GA uses a proposed constraint satisfaction algorithm called random walk heuristic to create an initial population of neural network architectures of high performance. At each generation, the GA population is split into a sub-population of inputs and a sub-population of 2-hidden layer architectures to breed offspring from each sub-population independently in order to generate a wide variety of network architectures. During the breeding of 2-hidden layer architectures, a constraint-based nearest neighbour search algorithm is proposed to find the nearest neighbours of the offspring population generated by mutation. The performance of the GA in detecting LOCA was evaluated using a break size dataset generated by the RELAP-3D nuclear simulator and the skillcraft dataset from the UCI machine learning repository. The performance of the GA was compared with that of a random search, that of an exhaustive search and that of a RBF kernel support vector regression (SVR). The results showed that for LOCA detection, the GA-optimised network outperformed all the other approaches in terms of generalization performance. For the skillcraft dataset, the GA-optimised network has a similar performance to the RBF kernel SVR and outperformed the other approaches.

Key words: multilayer perceptron, genetic algorithms, constraint satisfaction problems, random search, exhaustive search, loss of coolant accidents of NPP

1. Introduction

Nuclear power plants (NPP) life management is concerned with monitoring the safety, conditions of the components of a NPP and the maintenance of the NPP in order to extend its lifetime. It is crucial to regularly monitor the safety of the components of a NPP to detect as early as possible any serious anomalies which would potentially cause accidents. When an accident is predicted to occur or occurring, the plant operator must take necessary actions as quickly as possible to safeguard the NPP, which involves complex judgements, making trade-offs between demands and requires a lot of expertise to make critical decisions. It is commonly believed that timely and correct decisions in these situations could either prevent an event from developing into a severe accident or mitigate the undesired

*Corresponding author
Email addresses: d.tian@leedsbeckett.ac.uk; dtian09@gmail.com (David Tian), jiameideng@hotmail.com (Jiamei Deng), vgopika@barc.gov.in (Gopika Vinod), santutv@barc.gov.in (T.V. Santhosh), H.Tawfik@leedsbeckett.ac.uk (Hissam Tawfik)

consequences of an accident. As nuclear power plants become more advanced, their safety monitoring approaches in turn advance considerably. Current approaches include nuclear reactor simulators, safety margin analysis [34, 2], Probabilistic Safety Assessment (PSA) [34, 1] and artificial intelligence (AI) methods [36, 41, 13, 27]. Nuclear reactor simulators such as RELAP5-3D [49] are used to simulate the dynamics of a NPP in accidental scenarios and to generate transient datasets of reactors for training neural networks to detect severe accidents. Safety margin analysis analyses the values of the safety parameters of a reactor and triggers an alert to the plant operator if the safety margin falls below a minimum safety margin. PSA computes the probability of occurrence of accidents based on the probabilities of the component failures for causing accidents.

AI approaches such as robotics [17, 24], neural networks [36, 41, 13, 27] and fuzzy systems [59, 12] have gained considerable attention in detecting failure and accidents of nuclear systems over the past decade. Continual inspection of critical components such as the primary heat transport (PHT) is crucial to maintaining the safety of the NPP. However, human inspection is dangerous and difficult due to the hazardous environment and geometric restraints of the components [17, 24]. Inspection robotics has been used as an alternative to human inspectors to determine early warning of component failure and prevent possible nuclear accidents. A snake-armed inspection robot attached with high resolution camera has been used to examine the conditions of the PHT pipes of a reactor under hazardous environments which are too dangerous for human inspectors [17]. Successful predictive models have been developed by training and testing neural networks and fuzzy systems on transient datasets generated using RELAP5-3D and MAAP5 simulators. Na MG et al [36] generated transient data of inlet headers (IH) using MAAP4 code and trained neural networks on the transient data to detect loss of coolant accidents (LOCA) in an advanced power reactor 1400 (APR1400). Zio E et al [59] applied fuzzy similarity analysis approaches to detecting the failure modes of nuclear systems. Wang WQ et al [38] developed a neuro-fuzzy system to predict the fuel rod gas pressure based on cladding inner surface temperatures in a loss-of-coolant accident simulation. Baraldi P et al [12] proposed an ensemble of fuzzy c-mean classifiers to identify the faults in the feed water system of a boiling water reactor. Wei X et al [57] developed self-organizing RBF networks to predict fuel rod failure of nuclear reactors. Souza [48] developed a RBF network capable of online identifying the accidental dropping of the control rod at the reactor core of a pressurised water reactor. Secchi P et al [42] developed bootstrapped neural networks to estimate the safety margin on the maximum fuel cladding temperature reached during a header blockage accidental scenario of a nuclear reactor. Back J et al [10] developed a cascaded fuzzy neural network using simulation data of the optimized power reactor 1000 to predict the power peaking factor in the reactor core in order to prevent nuclear fuel melting accidents. Guimares A and Lapa C [26] used an adaptive neural fuzzy inference system (ANFIS) to detect the cladding failure in fuel rods of a nuclear power plant.

A multilayer perceptron (MLP) is a type of feedforward neural networks [15, 16] and consists of a layer of inputs, a number of hidden layers of neurons (activation functions) and a layer of output neurons. The performance of a neural network critically depends on its architecture in that a too simple architecture would underfit to the training set and an over complicated architecture would overfit the training set [15, 16]. In both cases, the neural network would have a poor generalization performance on unseen data. Evolutionary algorithms such as genetic algorithms (GA) [11, 20, 32, 51, 35, 39] have been developed to optimize the architectures of neural networks in different applications such as manufacturing and image recognition [11, 20, 46]. The results show that GA outperforms random search and trial and error methods because GA finds the best solutions using the evolutionary process which finds solutions with better quality than the ones of the past generations; whereas, random search finds a best solution among some randomly-generated solutions; trial and error methods find a best solution among a number of solutions generated using different parameter settings set by the user.

Constraint satisfaction (CS) [8, 23, 40] is a well-established field in AI. For many real-world problems, the users' domain knowledge are available. The problems can be modeled using constraints and solved using CS algorithms. A constraint satisfaction problem (CSP) consists of a set of variables each of which is associated with a domain containing valid values of each variable and a set of constraints over the variables [23]. A solution to a CSP is a simultaneous assignment of a value to each of the variables while satisfying all the constraints over the variables. CS has been successfully applied to different problems such as tasks scheduling and resource optimization. A constraint satisfaction optimization problem (CSOP) is a CSP with an objective function so that an optimal solution has the maximum or the minimum objective value. CS has been effectively used to build and train models in machine learning. Support vector machines (SVMs) [18, 37, 44] are kernel machines which transform the training patterns to high dimensional feature spaces using kernel functions and construct maximum margin hyperplanes for pattern

classification. SVMs are formulated as quadratic programming problems (a type of CSOPs) and are trained using constrained optimization algorithms such as the sequential minimal optimization algorithm (SMO). Cussens [21, 22] and Bartlett [14] modeled the problem of optimizing the structures of Bayesian networks as an integer program (a type of CSOP) with acyclic constraints and solved the optimization model using the cutting plane method.

Motivated by the works in using CS to build and train models in machine learning, this work proposes a constraint-based GA to find an optimised 2-hidden layer network architecture for detecting LOCA of a nuclear reactor. The initial population of the GA consists of neural network architectures. With the availability of domain knowledge on the architectures of neural networks of high performance, a neural network architecture in the initial population can be formulated as a CSP and created by solving the CSP using a proposed random walk heuristic. At each generation of the GA, the current population is split into 2 sub-populations: a sub-population of inputs and a sub-population of 2-hidden layer architectures where a 2-hidden layer architecture consists of 2 hidden layers and no inputs. Then, each sub-population undergoes a breeding process which consists of 1-point crossover, 2-point crossover and mutation. Moreover, during breeding the offspring of the 2-hidden layer architectures, a constraint-based nearest neighbour search algorithm is proposed to find the nearest neighbours of the offspring generated by mutation using CSOP techniques. This paper is organized as follows: Section 2 reviews the related work in optimizing the architectures of neural networks; Section 3 describes the loss of coolant accidents in NPPs and the generation of a break size dataset; Section 4 describes the proposed constraint-based GA and analyzes its computational complexity; Section 5 evaluates the performance of the GA using a break size dataset and the skillcraft dataset from the UCI machine learning repository; Section 6 discusses the results; conclusions and future work are presented in Section 7.

2. Related Work

An optimal neural network architecture can be obtained by exhaustive search of a very wide range of neural network architectures. However, this is often too computationally expensive to perform in practice. Trial-and-error methods [20, 39, 47] and random search methods [20, 39, 47] are common approaches for finding neural network architectures of high performances. In trial and error methods, a number of architectures are created empirically and compared with one another for their performances in order to select an optimal architecture. Random search generates random network architectures and compare their performances to find an optimal architecture. In recent years, metaheuristics such as GA [39, 46, 43, 47, 52], simulated annealing (SA) [43, 47], particle swarm optimization (PSO) [53] and tabu search [6, 58] have been used to optimize the architectures of neural networks in different applications. Metaheuristics explore a sub-space of the whole search space of all the neural network architectures to find optimal network architectures and are faster than exhaustive search. Metaheuristics often have better performances than the trial and error method. Ciancio et al [20] compared the performances of the 4 algorithms: GA, tabu search, Taguchi and decision trees in optimizing the architectures of neural networks for 3 manufacturing problems: the extrusion process, the rolling process and the shearing process. For each problem, the 4 algorithm optimize the following features of an architecture: the number of hidden layers, the number of neurons of each hidden layer, the types of the activation functions and the type of the training algorithm. The number of inputs and that of the outputs of an architecture are the fixed parameters. Each architecture has 3 inputs and 2 outputs. The GA has the best performances for the extrusion and the shearing processes [20]. Tabu search has similar performances to the GA for the 3 problems and the best performance for the rolling process. Aladag et al [6] proposed a tabu search to optimize the number of inputs and the number of neurons of the hidden layer of 1-hidden layer architectures. The performance of the tabu search was compared with that of the trial and error method on 5 time series datasets: the number of outpatient visits, the number tourist visits in Turkey, the beer consumption in Austria, the air pollution records in Ankara and the electricity consumption in Turkey. The tabu search found better network architectures than the ones found by the trial and error method. Soares et al [47] proposed a GA and a simulated annealing (SA) based approaches to optimize the structure of a neural network ensemble which is a combination of a number of single hidden layer neural networks. The GA and the SA find an optimal combination of neural network models and a combination strategy among a set of network models and a set of combination strategies. The GA and the SA had similar performances on 2 widely-used datasets: Friedman and Boston Housing. The GA and the SA had better performances than the individual network models, the ensemble of all the individual models and some well-known ensemble approaches including Bagging and AdaBoost on the 2 datasets. Rocha et al [39] proposed a random neural network search approach called Heuristic Neural Network (HNN), a GA neural network approach called Topology-optimization Evolutionary Neural Network

(TENN) and the Simultaneous Evolutionary Neural Network approach (SENN) which optimizes weights and architectures of neural networks simultaneously. The HNN ensemble (HNNE), the TENN ensemble (TENNE) and the SENN ensemble (SENNE) were also proposed. HNN, TENNE and SENN optimise the architectures of single hidden layer MLPs. These methods were compared using 8 classification datasets and 8 regression datasets from the UCI machine learning repository. For both the classification and regression datasets, TENNE and SENNE outperformed the single neural network approaches HNN, TENN and SENN on most datasets. Moreover, TENN and SENN outperformed the HNN on most datasets. Valdez et al [53] proposed a hybrid algorithm for optimizing the architectures of modular neural networks. The hybrid algorithm combines Particle Swarm Optimization (PSO) and GA using fuzzy logic. The hybrid approach outperformed GA, PSO and a trial-and-error approach on a human facial image database. Melin et al [35] proposed a multi-objective genetic algorithm for optimizing modular granular neural network architectures. The proposed approach outperformed a number of other approaches for designing modular granular neural network architectures on benchmark face and ear datasets. Signal processing algorithms have been successfully used to optimize the architectures of convolutional neural networks [7, 25]. In control systems, Wang et al [54] proposed a quantized sampled-data neural network controller and Wei et al [55] proposed a semi-Markovian jump neural network controller.

Santhosh et al [41] designed a neural network using a trial and error method to detect the size of a break, its location on the PHT of a nuclear reactor and the availability of the emergency core cooling system (ECCS). Santhosh's neural network consists of 37 inputs, 2 hidden layers and 3 output neurons which output break size, the location of the break and the availability of the ECCS. Based on our prior knowledge, neural networks of high performances normally have between 5 to 40 neurons in each hidden layer. It would be too computationally expensive to train every possible 2-hidden layer neural network architecture with between 2 to 36 inputs and between 5 to 40 neurons in each hidden layer in order to find an optimal 2-hidden layer network architecture because the number of the 2-hidden layer network architectures is extremely large i.e. 1.7812e+14 $((2^{37} - 3) \times 36 \times 36)$ where the architectures with different subsets of inputs are counted as different architectures as well as the architectures with different numbers of inputs and/or different number of neurons in the hidden layers. For example, a neural network NN1 consists of the first 4 inputs of a training set, a hidden layer of 10 neurons and 1 output neuron. Another neural network NN2 consists of the 1st, 3rd, 5th and 6th inputs of the training set, a hidden layer of 10 neurons and 1 output neuron. The architectures of NN1 and NN2 are different although they both have 4 inputs. This work proposes a constraint-based GA to find optimal 2-hidden layer neural network architectures with between 2 to 36 inputs and between 5 to 40 neurons in each hidden layer for detecting the break size of the PHT of the nuclear reactor.

3. Loss of Coolant Accidents in Nuclear Power Plants and the Generation of a Break Size Dataset

This work used RELAP5-3D to simulate the dynamics of the parameters of a nuclear reactor in LOCA scenarios and generate a break size dataset for training neural networks to detect the break sizes of IHs during LOCA scenarios. A LOCA is caused by a large break of the IHs of the primary heat transport system (PHT) (Figure 1 [31]) of a nuclear reactor as follows. When large breaks of IHs of the PHT occur, the system depressurizes rapidly which causes coolant voiding into the reactor core. This coolant voiding into the core causes positive reactivity addition and consequent power rise. Then, the emergency core cooling system automatically shuts down the reactor to keep the NPP safe. During an occurrence of a break, transient data such as the temperature and pressure of the IHs can be collected during a short time period to detect the sizes of the break using neural networks. The break size is defined as the percentage of the cross-sectional area of an IH [41]. The break size is between 0% (no break) and 200% i.e. double cross-sectional areas of an IH (a complete rupture of the IH). It is infeasible to generate all possible break sizes. In this study, a transient dataset consisting of the 10 break sizes 0%, 20%, 40%, 50%, 60%, 75%, 100%, 120%, 160% and 200% was generated using RELAP5-3D. The break sizes of 100% or greater are considered as large breaks. For each break size, the 37 signals used in [41] were collected at various parts of the PHT over 60 seconds using RELAP5-3D under the assumption that this time duration is sufficient to identify large break LOCA in IH. For each break size, the signals were measured at 541 time instants within a 60s duration. Therefore, each break size class of the dataset consists of 541 instances (observations) and 37 features (signals). The dataset is a 5410×38 matrix with the last column representing the break size (the output).

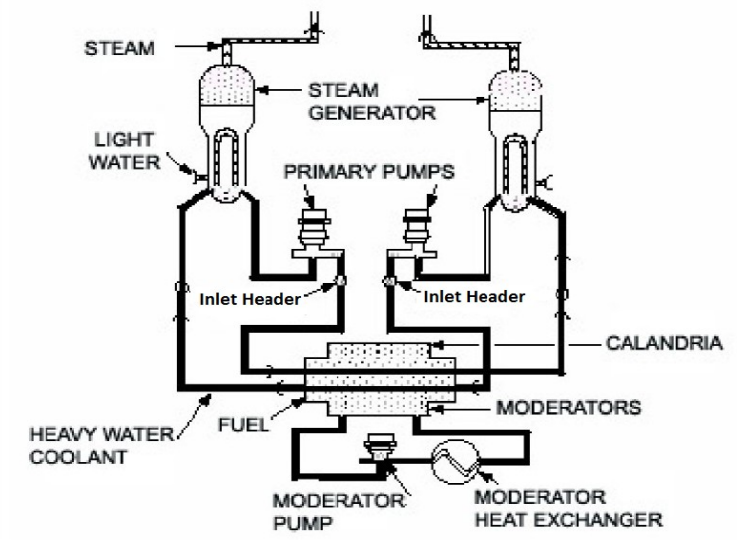

Figure 1: The PHT of a nuclear reactor

4. The Proposed Genetic Algorithm

Genetic algorithms are optimization techniques that mimic the evolution process where the fittest individuals have the greatest possibilities of surviving and reproducing offspring. In a GA, A fitness function is defined to measure the quality of each individual of a population. An individual with the maximum fitness function value is an optimal individual and returned by the GA. In this work, a chromosome represents a 2-hidden layer MLP architecture with a subset of all the 37 inputs and 1 output. The tanh sigmoid activation function is chosen as a hidden neuron. The linear function is chosen as the output neuron. Given a training set of D inputs, the 1st D bits of a chromosome represent the inputs of the network architecture with 1 indicating the inclusion of an input to the architecture and 0 the exclusion of an input from it. The next 6 bits of the chromosome represent the number of neurons in the 1st hidden layer. The remaining 6 bits of the chromosome represent the number of the neurons in the 2nd hidden layer (Figure 2). To obtain the number of neurons of a hidden layer, the corresponding 6-bit binary number is decoded into a decimal integer in the range [1,63].

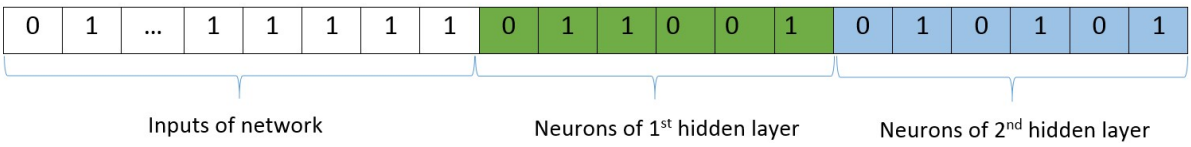

Figure 2: A chromosome representing a network architecture

The pseudocode and the flowchart of the proposed GA are illustrated in Figures 3 and 4, where N is the size of the population; M is the size of the parents population ($M \leq N$); min_inputs is the minimum number of inputs of a network architecture; max_inputs is the maximum number of inputs of a network architecture; min_neu is the minimum number of neurons of a hidden layer; max_neu is the maximum number of neurons of a hidden layer; min_ws is the minimum number of weights of a network architecture and max_ws is the maximum number of weights of a network architecture. The fitness function of the GA is described in Section 4.1. The steps 1, 4 and 5 are described in detail in Sections 4.2, 4.3 and 4.4.

Algorithm 1: Constraint-based genetic algorithm
Input: a dataset e.g. a break size dataset
Output: an optimised neural network architecture

1. Create an initial population of N 2-hidden layer network architectures each of which has between min_inputs and max_inputs inputs, between min_neu and max_neu neurons in each hidden layer and between min_ws and max_ws weights using constraint satisfaction. Set the generation number I to 1.
   Set the current population to the initial population.
2. Select the M fittest individuals from the current population using stochastic universal sampling (SUS) [19].
   In SUS, the fitter the individuals, the more chance they are chosen. The selected individuals
   form the population P of parents.
3. Split P into a sub-population P1 of feature subsets and a sub-population P2 of 2-hidden-layer architectures.
   P1 consists of the 1st D columns of P and all the rows of P. P2 consists of the remaining 12 columns of P and all the rows of P.
4. Breed an offspring population OP1 from P1 using 1-point crossover, 2-point crossover and mutation.
5. Breed an offspring population OP2 from P2 using 1-point crossover, 2-point crossover, mutation and
   a proposed constraint-based nearest neighbour search.
6. Create a population P3 of network architectures by joining OP1 with OP2 along their columns so that each row of P3 is a (D+12)-bit chromosome representing a network architecture. Any architectures with a hidden layer of less than 5 neurons or greater than 40 neurons are removed from P3.
7. Update the current population as follows. Keep a fraction R (e.g. R=0.1) of the fittest individuals in the current population and replace the other individuals in the current population with the fittest (1-R) offspring of P3.
   This implements elitism with an elitism rate R.
8. If I is equal to the maximum number of generations N, go to Step 9. Otherwise, increment I by 1 and go to Step 2.
9. Output the optimal network architecture.

Figure 3: Pseudocode of the contraint-based genetic algorithm

4.1. The Fitness Function

The fitness of a network architecture is defined based on the performance of that network architecture in detecting LOCA. The following performance measures are used for the break size dataset:

- Root Mean Square Error (RMSE) of a break size k:

$$RMSE_k = \sqrt{MSE_k} \tag{1}$$

and

$$MSE_k = \frac{\sum_{i=1}^{M}(O_i - T_i)^2}{M} \tag{2}$$

where $k \in \{0\%, 20\%, 40\%, 50\%, 60\%, 75\%, 100\%, 120\%, 160\%, 200\%\}$; M is the number of the patterns of break size k in the test set; i is the i^{th} pattern of break size k in the test set; O_i is the output of the network for the i^{th} pattern; T_i is the break size target of the i^{th} pattern.

- $$\text{Mean RMSE} = \frac{\sum_k RMSE_k}{N} \tag{3}$$

where N is the number of the different break sizes in the test set and N=10 (10 break sizes).

- $$\text{Standard deviation of RMSEs} = \sqrt{\frac{\sum_k (RMSE_k - \overline{RMSE})^2}{N-1}} \tag{4}$$

where $\overline{RMSE}$ is the mean RMSE.

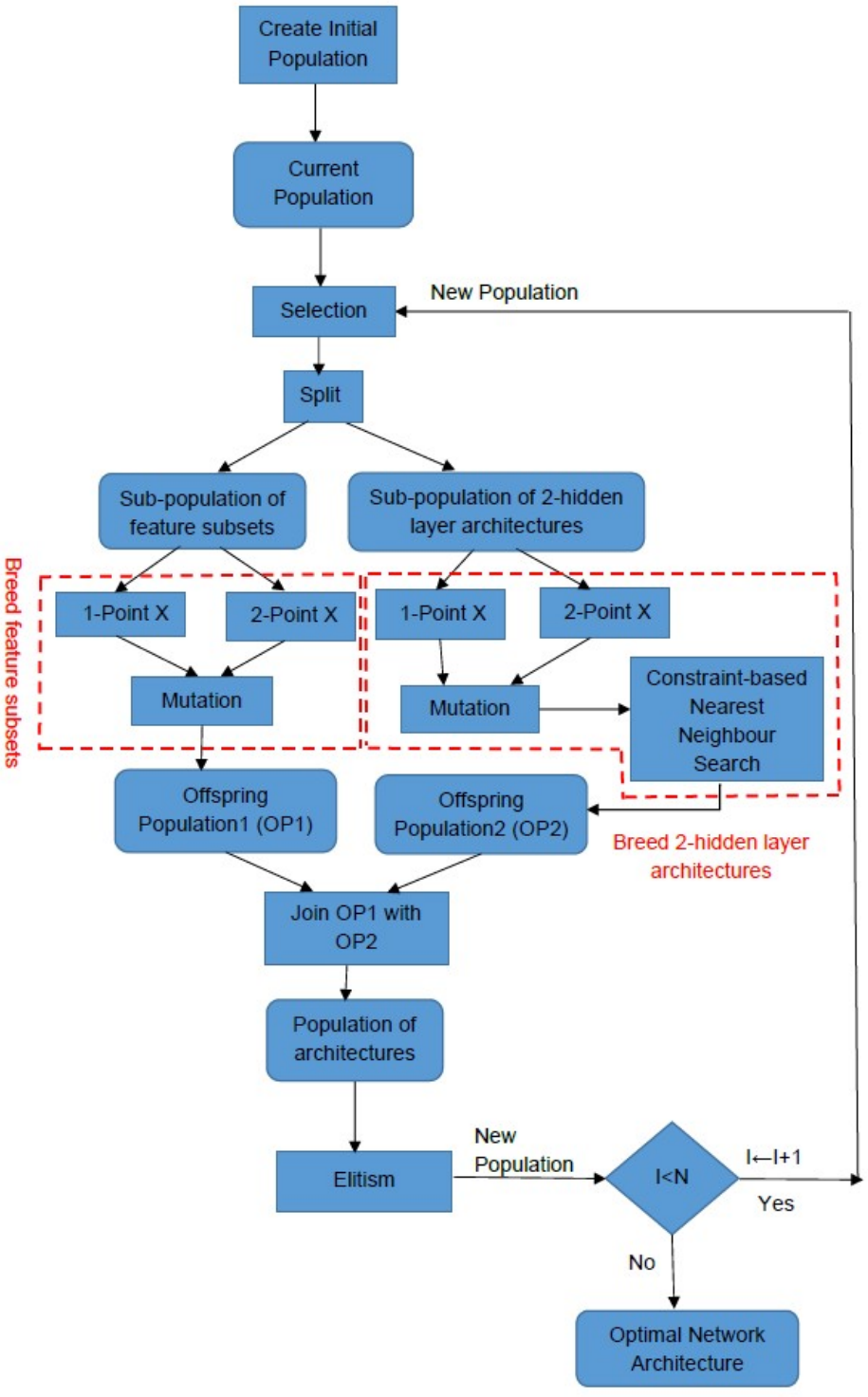

Figure 4: Flowchart of the proposed genetic algorithm: 1-point X (1-point crossover); 2-point X (2-point crossover)

The RMSE of a break size measures the performance of a network in detecting that specific break size. The mean RMES measures the average performance of a network in detecting break sizes. The standard deviation of RMSEs measures the stability/variation of the performance of a network in detecting different break sizes. The optimal network has the minimum mean RMSE.

The fitness function is defined based on the mean RMSE. Firstly, the network architectures of a population are ranked in descending order of their mean RMSEs. Then, the fitness of the network architecture at the position Pos in the ranking is defined as the linear ranking function [19]:

$$\text{Fitness(Pos)} = \frac{2(\text{Pos} - 1)}{(\text{N} - 1)} \tag{5}$$

where N is the size of the population. A network architecture with the maximum fitness function value is an optimal network architecture and returned by the GA.

4.2. Creation of an Initial Population using Constraint Satisfaction (Step 1 of the GA)

The quality of the initial population of the GA critically affects the performance of the individuals in the offspring populations. An initial population can be created using constraint satisfaction to generate network architectures whose numbers of inputs, numbers of neurons of each hidden layer, and numbers of weights are bounded by the user-specified upper and lower bounds: min_inputs, max_inputs, min_neu, max_neu, min_ws and max_ws. The upper and lower bounds can be specified based on the user's prior knowledge about network architectures with high performances. Then, the problem of creating a 2-hidden layer architecture (H1,H2) can be modelled as a constraint satisfaction problem CSP_MLP:

CSP_MLP
Variables: H1, H2;
Domains of variables: D(H1)={min_neu,min_neu+1,. . .,max_neu};
D(H2)={min_neu,min_neu+1,. . .,max_neu};
Constraint 1 (C1): S$\times H1 + H1 \times H2 + H2 \geq$ min_ws;
Constraint 2 (C2): S$\times H1 + H1 \times H2 + H2 \leq$ max_ws;

where S is the number of inputs, a randomly-generated integer in the range [min_inputs,max_inputs]. CSP_MLP consists of the variables H1, H2 and the constraints C1, C2. The domains D(H1) and D(H2) of H1 and H2 are the sets of valid values of H1 and H2. H1 is the number of the neurons of the 1st hidden layer; H2 is the number of the neurons of the 2nd hidden layer; C1 is the constraint that the number of the weights of an architecture is at least min_ws; C2 is the constraint that the number of the weights of an architecture is at most max_ws. An assignment of values to H1 and H2 satisfying C1 and C2 is a solution of CSP1. For example, for a training set of size 1000 with the settings: min_inputs=5, max_inputs=36, min_neu=5, max_neu=40, S=15, ws_min=490 and ws_max=510, C1 becomes $15 \times H1 + H1 \times H2 + H2 \geq 490$ and C2 becomes $15 \times H1 + H1 \times H2 + H2 \leq 510$. A solution of CSP_MLP is the vector (21,8) which corresponds to a network architecture with 15 inputs, 21 neurons in the 1^{st} hidden layer, 8 neurons in the 2^{nd} hidden layer and 1 output neuron.

The search space of CSP_MLP can be represented as a tree where each node represents a variable with the root node at top of the tree ; each branch represents an assignment of a value in the domain of a variable to that variable and each leaf (a bottom node) represents a solution candidate (Figure 5). The CSP_MLP can be solved using the backtrack search algorithm (BS) [50]. BS assigns a value v_1 in D(H1) to H1 and a value v_2 in D(H2) to H2. Then, BS checks the satisfiability of the solution candidate (v_1,v_2) with regard to C1 and C2. If (v_1,v_2) does not satisfy both C1 and C2, BS backtracks to H2 and assigns a different value v_2' in D(H2) to H2 and checks the satisfiability of (v_1,v_2') with regard to C1 and C2. This process repeats until a solution is found or there is no solution to CSP_MLP. BS traverses a sub-space of the search space during searching for a solution (Figure 5). In the worst case, each solution candidate is checked against all the constraints once, the computational complexity of BS is O(ed^n) [50] where e is the number of the constraints; d is the size of the largest domain and n is the number of the variables.

A random walk heuristic (Figure 7) is proposed to find a solution to a CSP_MLP by exploring a smaller sub-space than the backtracking search explores (Figure 6). The random walk heuristic assigns a random value v_1 to H1 and looks ahead whether this partial solution candidate leads to a solution of CSP_MLP without both assigning a value v_2

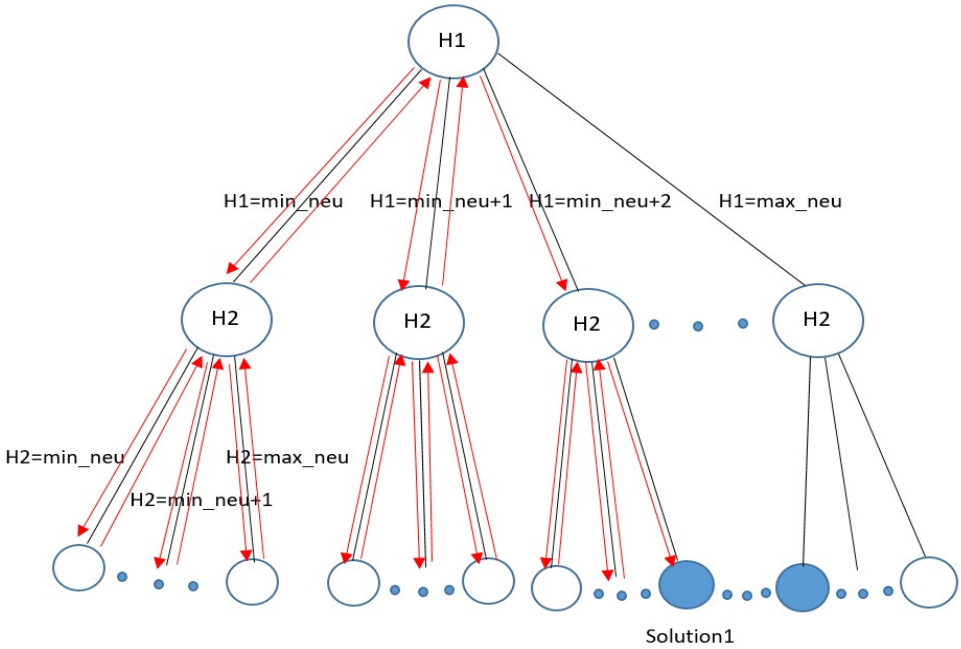

Figure 5: The search space of CSP_MLP: the arrows indicate the order in which backtrack search traverses the search space starting at H1

to H2 and checking the satisfiability of C1 and C2 of the solution candidate (v_1, v_2). If this partial solution does not lead to a solution, a new value v_1' is assigned to H1 repeatedly until a partial solution candidate leads to a solution (v_1',v_2). When a value is assigned to H1, a look-ahead operation is performed by calling the AC-3 algorithm [23] which reduces the CSP_MLP to a simpler CSP called CSP_MLP2 with smaller domain sizes for the variables. Redundant values of a variable are the values which violate the constraints on that variable. AC-3 maintains the arc-consistency of a CSP by removing any redundant values from the domains of the variables of the CSP. An arc-consistent CSP is returned by AC-3.

The arc-consistency of a CSP is defined as follows[23]:

- A variable X is arc-consistent with another variable Y if, for every value a in the domain of X there exists a value b in the domain of Y such that (a,b) satisfies all the constraints between X and Y.
- A CSP is arc-consistent if every variable is arc-consistent with every other one.

If AC-3 reduces the domain of H1 or that of H2 to an empty set, there is no compatible value in D(H1) or D(H2) which satisfies C1 and C2, so there is no solution to the CSP. The look-ahead operation is as follows. After assigning a value v_1 to H1, the domain of H1 is updated to the singleton $\{v_1\}$ (step 6 of Algorithm 2) and the domain of H2 is updated by calling AC-3 (step 7). If the domain of H1 and that of H2 are not empty sets, the vector (v_1, v_2) is a solution; otherwise, assigning v_1 to H1 does not lead to a solution and the next iteration of the loop begins.

The worst case computational complexity of AC-3 is O(ed^3) [50] where e is the number of constraints and d is the size of the largest domain of the variables. In the worst case, the for loop of random walk heuristic iterates d times. Therefore, the worst case computational complexity of Algorithm 2 random walk heuristic is O(ed^4).

Algorithm 3 (Figure 8) is proposed to generate an initial population of M random network architectures by solving M separate CSP_MLPs using the random walk heuristic where D is the number of the features of the training set. The computational complexity of Algorithm 3 is O(ed^4M).

4.3. Breeding Offspring from a Population of Feature Subsets (Step 4 of the GA)

An offspring population is breed from the sub-population P1 of feature subsets as follows:

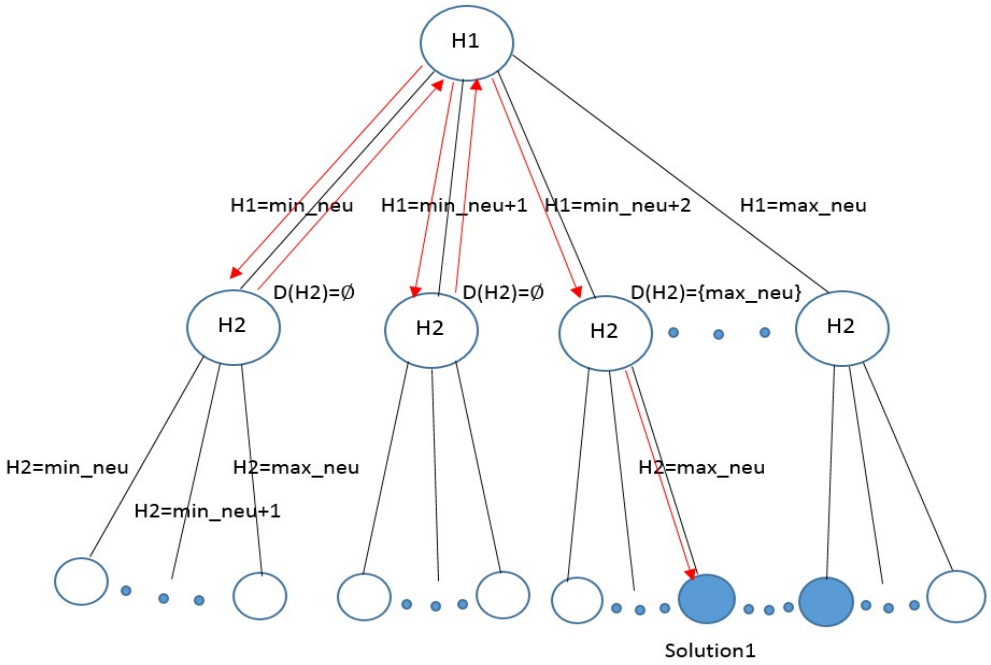

Figure 6: The search space exploration of random walk heuristic (RWH): the arrows indicate the order in which RWH traverses the search space; when min_neu or min_neu+1 is assigned to H1, AC-3 reduces D(H2) to an emptyset; a solution is found after min_neu+2 is assigned to H1 and AC-3 reduces D(H2) to {max_neu}

Algorithm 2: Random walk heuristic
Input: MLP_CSP
Output: a solution S or ∅ if no solution exists
1.Randomly order the values in the domain of H1;
2.Randomly order the values in the domain of H2;
3.S ← ∅;
4.**foreach**($v_1 \in$ D(H1))
5.{
6. D'(H1) ← $\{v_1\}$; /* D'(H1) and D'(H2) are new domains of H1 and H2 */
7. (D'(H1),D'(H2))←AC-3(H1,D'(H1),H2,D(H2),C1,C2);
8. **if**(D'(H1)!=∅ and D'(H2)!=∅)
9. { S ← (v_1, v_2) where $v_2 \in D'$(H2);
10. **break**;
11.}
12.}
13.return S;

Figure 7: Pseudocode of the random walk heuristic

1. 1-point crossover is applied to a fraction f of the individuals of P1 e.g. f=0.5. Let M be the size of P1, 1-point crossover is performed on randomly-chosen pairs of $\lfloor Mf \rfloor$ individuals with a probability P_{xor} to create an offspring population as follows. Given two individuals, a random position i of one of the individuals is chosen; then, the part of that individual from the position i+1 to the last position of that individual is swapped with the corresponding part of the other individual to create 2 offspring (Figure 9).
2. 2-point crossover is applied to the other fraction (1-f) of the individuals of P1. 2-point crossover is performed

Algorithm 3: Create an initial population
Inputs: population size M, min_inputs, max_inputs, min_neu, max_neu, min_ws, max_ws
Output: a population P of network architectures
1.i=0; P=∅;
2.While(i<M){
3. Generate a random integer S representing the number of inputs where min_inputs ≤S≤ max_inputs;
4. Solve CSP_MLP using the random walk heuristic to create a 2-hidden layer architecture (H1,H2);
5. Transform the network architecture (S,H1,H2) to a chromosome C as follows:
    1. Create a bit string F of length D with S randomly-set true bits;
    2. Encode H1 as a 6-bit binary number B1;
    3. Encode H2 as a 6-bit binary number B2;
    4. Merge F, B1 and B2 together to create C;
6. Insert C into P;
7. i← i+1;}
8.Return P;

Figure 8: Pseudocode of the create initial population algorithm

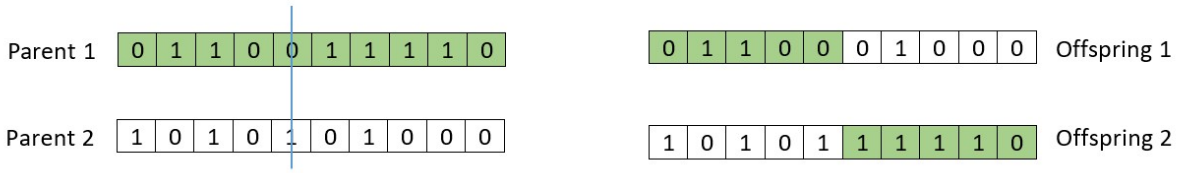

Figure 9: an example of 1-point crossover at the 5^{th} position of two parents

on random-chosen pairs of the remaining M-$\lfloor Mf \rfloor$ individuals with a probability P_{2xor} to create an offspring population as follows. Given two parents, two random positions i and j on one of the parents are chosen with i<j; then, the segments between i and j of the 2 parents are swapped to create two offspring (Figure 10).

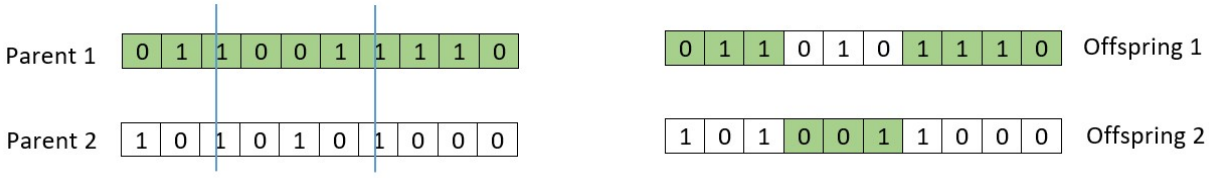

Figure 10: an example of 2-point crossover at the 3^{rd} and 7^{th} positions of two parents

3. Mutate each offspring of the two offspring populations created in steps 1 and 2 with a probability P_m to create a new population. During mutation, one or more bits of an individual are selected at random and flipped with a probability P_m.

4.4. Breeding Offspring from a Population of 2-Hidden-Layer Network Architectures

An offspring population is breed from the sub-population P2 of 2-hidden-layer architectures as follows:

1. 1-point crossover is applied to a fraction f' of the individuals of P2 e.g. f'=0.5. 1-point crossover is performed on randomly-chosen pairs of $\lfloor Mf' \rfloor$ individuals with a probability P'_{xor} to create an offspring population where M is the size of P2.

2. 2-point crossover is applied to the other fraction (1-f') of the individuals of P2. 2-point crossover is performed on random-chosen pairs of the remaining M-$\lfloor Mf' \rfloor$ individuals with a probability P'_{2xor} to create an offspring population as follows. Given two individuals, two random positions i and j on one of the individuals are chosen with i<j; then, the segments between i and j of the 2 individuals are swapped to create two offspring (Figure 10).
3. Mutate each offspring of the two offspring populations created in steps 1 and 2 with a probability P'_m to create a new offspring population.
4. Run the proposed constraint-based nearest neighbour search (section 4.4.1) to find the architectures which are the most similar to the offspring population and satisfy the constraints that the number of neurons in each hidden layer is between min_neu and max_neu.

4.4.1. Constraint-based Nearest Neighbour Search

Definition 1: The nearest neighbour architecture of a 2-hidden layer architecture consists of the nearest neighbours of the binary encodings of the numbers of the neurons in the 2 hidden layers of the architecture.

Definition 2: The nearest neighbour N of the binary encoding H of a hidden layer is a binary code which:

1. has the smallest hamming distance to H and;
2. encodes the decimal integer N_{dec} that is the closest to the integer encoded by H satisfying the constraint: min_neu $\leq N_{dec} \leq$ max_neu.

For example, the offspring 101111011111 encodes an architecture where 101111 encodes the 47 neurons in the 1^{st} hidden layer and 011111 encodes the 31 neurons in the 2^{nd} hidden layer. Given min_neu=5 and max_neu=40, the nearest neighbour of 101111 is 100111 encoding 39 with a hamming distance of 1; the nearest neighbour of 011111 is 011111 which encodes 31 with a hamming distance of 0. The problem of finding a nearest neighbour of a 2-hidden layer architecture (H1,H2) can be modelled as a constraint satisfaction optimization problem (CSOP) MLP_CSOP:

MLP_CSOP
Boolean Variables: B_1, B_2, B_3, B_4, B_5, B_6, C_1, C_2, C_3, C_4, C_5, C_6;
Domains of the Boolean variables: $\{0, 1\}$
Constraint 1: $B_1 \times 32 + B_2 \times 16 + B_3 \times 8 + B_4 \times 4 + B_5 \times 2 + B_6 \times 1 \geq$ min_neu;
Constraint 2: $B_1 \times 32 + B_2 \times 16 + B_3 \times 8 + B_4 \times 4 + B_5 \times 2 + B_6 \times 1 \leq$ max_neu;
Constraint 3: $C_1 \times 32 + C_2 \times 16 + C_3 \times 8 + C_4 \times 4 + C_5 \times 2 + C_6 \times 1 \geq$ min_neu;
Constraint 4: $C_1 \times 32 + C_2 \times 16 + C_3 \times 8 + C_4 \times 4 + C_5 \times 2 + C_6 \times 1 \leq$ max_neu;
Minimize: Objective=hamming_distance(N1,H1)+hamming_distance(N2,H2)+decimal_distance(N1,H1)
+decimal_distance(N2,H2);

where B_i are the Boolean variables corresponding to the bits in the encoding of the 1st hidden layer; C_i are the Boolean variables corresponding to the bits in the encoding of the 2nd hidden layer; H1 is the encoding of the 1st hidden layer of a hidden layer architecture; H2 is the encoding of the 2nd hidden layer of the hidden layer architecture; N1 is the the nearest neighbour of H1 i.e. N1=(B_1, B_2, B_3, B_4, B_5, B_6,); N2 is the nearest neighbour of H2 i.e. N2=(C_1, C_2, C_3, C_4, C_5, C_6); hamming_distance(Ni,Hi) computes the hamming distance between Ni and Hi; decimal_distance(Ni,Hi)=$|Num\_Ni - Num\_Hi|$ with Num_Ni being the decimal integer represented by Ni and Num_Hi being the decimal integer represented by Hi. The optimal solution (B_1, B_2, B_3, B_4, B_5, B_6, C_1, C_2, C_3, C_4, C_5, C_6) to MLP_CSOP can be obtained by minimizing the objective using the branch and bound algorithm [8, 23, 40]. A constraint-based nearest neighbour search (Algorithm 4) is proposed to find the nearest neighbours of the M offspring of mutation by solving M separate MLP_CSOPs Figure 11, where O is an offspring.

4.5. The Computational Complexity of the Genetic Algorithm

The main computational intensive operations of the GA are the creation of an initial population, the SUS selection, fitness evaluation of each individual, the 2 1-point crossover operations, the 2 2-point crossover operations, the 2 mutation operations and the constraint-based nearest neighbour search (CNNS). Let N be the population size, G be the maximum generations and M be the number of the individuals selected by SUS. Creating

Algorithm 4: Constraint-based nearest neighbour search
Inputs: an offspring population OP of 2-hidden-layer network architectures, min_neu, max_neu
Output: a population P of the nearest neighbours
P=$\emptyset$;
foreach(O$\in$OP) {
  Find the nearest neighbour N of O by solving a MLP_CSOP using the branch and bound algorithm;
  Insert N into P;
}
Return P;

Figure 11: Pseudocode of the constraint-based nearest neighbour search

an initial population involves solving N MLP_CSPs using the random walk heuristic. During each generation of the GA, M MLP_CSOPs are solved using CNNS. In general, solving a CSP or a CSOP is an NP-hard problem [8]. The search space of a MLP_CSP is $|D(H1)| \times |D(H2)|$ where D(H1)={mini_neu,mini_neu+1,. . .,max_neu}; D(H2)={mini_neu,mini_neu+1,. . .,max_neu}; $|D(H1)|$ is the domain size of H1 and $|D(H2)|$ is the domain size of H2. With the setting of mini_neu to 5 and max_neu to 40, the search space of a MLP_CSP is 36^2 i.e. 1296. The search space of a MLP_CSOP is 2^{12} i.e. 4096 because a MLP_CSOP consists of 12 Boolean variables and the domain size of each Boolean variable is 2. Therefore, MLP_CSP and MLP_CSOP are small problems in terms of their search space sizes. If solving a MLP_CSP or a MLP_CSOP is counted as an operation, there are N MLP_CSP operations and G×M MLP_CSOP operations. During each generation, there are N fitness evaluations, M selection operations, $\lfloor \lfloor Mf \rfloor /2 \rfloor P_{xor} + \lfloor \lfloor Mf' \rfloor /2 \rfloor P'_{xor}$ 1-point crossover operations, $\lfloor (M - \lfloor Mf \rfloor)/2 \rfloor P_{2xor} + \lfloor (M - \lfloor Mf' \rfloor)/2 \rfloor P'_{2xor}$ 2-point crossover operations, 2M mutation operations and M CSOP1 operations. The computational complexity of the GA is:

$$O(N + G(N + 4M + \lfloor \lfloor Mf \rfloor /2 \rfloor P_{xor} + \lfloor \lfloor Mf' \rfloor /2 \rfloor P'_{xor} + \lfloor (M - \lfloor Mf \rfloor)/2 \rfloor P_{2xor} + \lfloor (M - \lfloor Mf' \rfloor)/2 \rfloor P'_{2xor})). \quad (6)$$

4.6. Implementation of the Genetic Algorithm

The proposed GA (Algorithm 1) was implemented using Matlab and the ECLiPSe constraint logic programming (CLP) system [3, 9]. The Matlab Genetic Algorithms Toolbox of University of Sheffield [19] was used to implement the evolutionary operations of the GA. Algorithms 2, 3 and 4 were implemented using ECLiPSe CLP. The ECLiPSe program is integrated into the Matlab program as a component so that the ECLiPSe program is called by the Matlab program on the platform. Experiments were performed on a Windows 10 desktop computer with an Intel Corei7 3.6GHZ CPU and a 16GB RAM.

4.7. Parameters Settings of the Genetic Algorithm

The settings of the parameters of the GA are illustrated in Table 1. The min_inputs, max_inputs, min_neu, max_neu, min_ws and max_ws are set based on our prior knowledge. The other parameters of the GA are set to some of the common values used in research.

4.8. Parameters Settings of Neural Networks Training

The break size dataset was randomly split into a 50% training set, a 25% validation set and a 25% test set using the random sub-sampling with no replacement method [28, 29, 56, 30]. The skillcraft dataset was randomly split into an 80% training set, a 10% validation set and a 10% test set. For the break size dataset, the inputs and the output of the training set are rescaled to the interval [-1,1] using min-max normalization before training neural networks. Each break size class of the training set consists of 270 instances which are uniformly drawn at random from the break size dataset. For the skillcraft dataset, the inputs and the output of the training set are normalized using zero-mean normalization before training networks. During evaluating the performance of a trained network in detecting the LOCA of a NPP, the output of the network is converted to the original range [0%,200%] by inversing the normalization

	Break Size	Skillcraft
Parameters settings	Population size=80	Population size=80
	Max generation=300	Max generation=300
	Elitism rate=0.1	Elitism rate=0.1
	min_inputs=5	min_inputs=5
	max_inputs=36	max_inputs=17
	min_neu=5	min_neu=5
	max_neu=40	max_neu=40
	min_ws=1000	min_ws=1000
	max_ws=1350	max_ws=2000
	f=0.5	f=0.5
	f'=0.5	f'=0.5
	P_{xor}=0.8	P_{xor}=0.8
	P'_{xor}=0.8	P'_{xor}=0.8
	P_{2xor}=0.8	P_{2xor}=0.8
	P'_{2xor}=0.8	P'_{2xor}=0.8
	P_m=0.05	P_m=0.05
	P'_m=0.05	P'_m=0.05

Table 1: Parameters settings of the GA

calculation. For the skillcraft dataset, the output of the network is converted to the original range [1,8]. This is because the performance of the network in predicting the original targets is more important in practice than its performance in predicting the normalized targets. For each dataset, Levenberg-Marquardt algorithm [15, 16] is used for networks training with the maximum epochs set to 1000 and the learning rate set to 0.001.

5. Performance Evaluation

5.1. Performance Evaluation using the Break Size Dataset

5.1.1. Performance of the Genetic Algorithm

The GA was run for 300 generations. The mean RMSE of the best network found in each generation is illustrated in Figure 12. The largest decrease in the mean RMSEs of the best networks is achieved at the 4th and the 96th generations (Figure 12). This indicates the largest increase in the performance of the best networks is achieved at these 2 generations. The improvement in the performance of the best networks is very little between the 5th and the 95th generations. The GA-optimised network is the best network with the highest performance. Between the 97th and the 295th generations, there is no improvement in the performance of the best networks. The GA-optimised network was obtained at the 296th generation (Figure 12) and consists of 24 inputs, 33 neurons in the 1st hidden layer, 8 neurons in the 2nd hidden layer, 1 output and 1064 weights. The mean RMSE of the GA-optimised network is 1.6764 (Figure 12). For all the break sizes except 200%, the RMSE of the GA-optimised network in detecting the break size is between 0 and 2.404 (Table 2). For the break size 200%, the RMSE is the largest i.e. 4.9727.

Break Size	0%	20%	40%	50%	60%	75%	100%	120%	160%	200%	SD
RMSE	0.0967	0.9172	2.0827	1.0498	1.0086	1.4701	1.6496	1.1127	2.4040	4.9727	1.3263

Table 2: The performance of the GA-optimised network in detecting each break size: SD (standard deviation)

Linear interpolation [33] is a method of constructing new data points within the range of a set of known data points by fitting straight lines using linear polynomials. Having obtained the GA-optimised network, the break sizes 2.5%, 5%, 7.5%, 10%, 12.5%,. . ., 195% and 197.5% which are missing in the break size dataset, were generated using linear interpolation as follows. For each missing break size, 541 instances were generated using linear interpolation

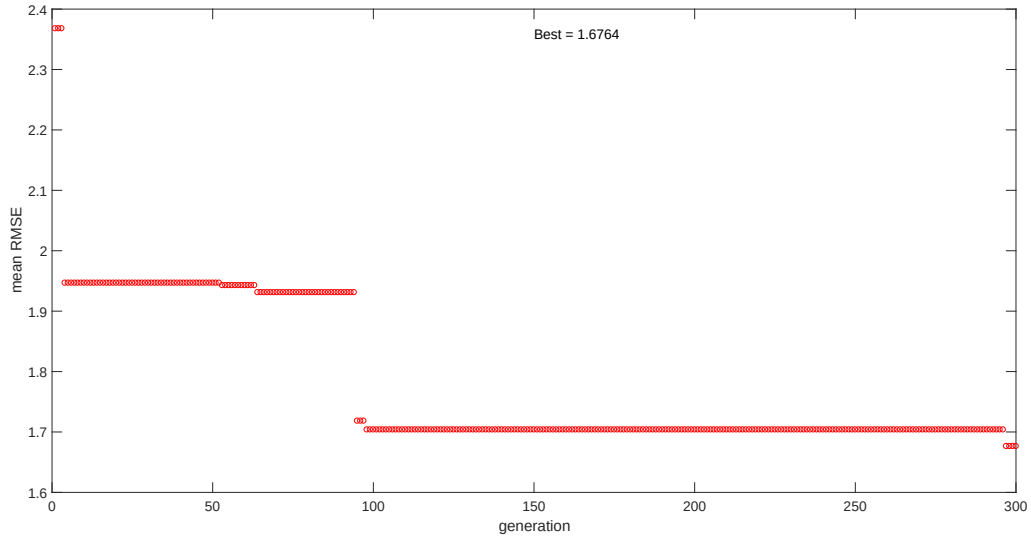

Figure 12: The mean RMSE of the best network found in each generation on the break size dataset: the GA-optimised network was found at the 296th generation

giving a total of 40575 instances. The break size dataset and the linear interpolation dataset were merged into a dataset containing 43821 instances.

In order to optimize the weights of the best GA-optimised network, it was trained and tested iteratively 100 times on the merged dataset to obtain a network with better performance than the GA-optimised network as follows. During each training-testing process, the merged dataset was randomly split into a 50% training set, a 25% validation set and a 25% test set; then, the weights of the network trained in the previous training-testing process were used as the initial weights of the current training-testing process before training began. This would give faster training speed than setting the initial weights to random values because each training process started at a minimum point on the error surface and stopped at another minimum point in the local region of the starting minimum point. The mean RMSE on the test set of the trained network was compared with that of the current best network. The mean RMSEs of the 100 networks are compared in Figure 13. The best network (GA-optimised network2) of the 100 networks has a smaller mean RMSE (1.1548) than the GA-optimised network (mean RMSE of 1.6764). For the break sizes 0%, 20%, 40%, 50%, 160% and 200%, the RMSE of the GA-optimised network2 is smaller than that of the GA-optimised network (Figure 14 and Table 3). For the break sizes 60%, 75% and 120%, the GA-optimised network has a slightly better performance than the GA-optimised network2. For the break size 100%, the GA-optimised network has a better performance than the GA-optimised network2. The standard deviation of the RMSEs of the GA-optimised network2 is 0.6860 which is smaller than that of the GA-optimised network (1.3263). Therefore, the GA-optimised network2 has a significantly more stable performance (less variation of performance) than the GA-optimised network in detecting different break sizes.

Break Size	0%	20%	40%	50%	60%	75%	100%	120%	160%	200%	SD
RMSE	0.0072	0.4736	0.5872	0.9633	1.0515	1.6342	2.1309	1.196	2.0576	1.4462	0.6860

Table 3: The performance of the GA-optimised network2 in detecting each break size: SD (standard deviation)

5.1.2. Performance Comparison with the Neural Network of the Previous Work

The Santhosh et al's neural network [41] consists of 37 inputs, 26 neurons in the 1st hidden layer, 19 neurons in the 2nd hidden layer and 3 output neurons which output the break size, location of the break and the availability of the

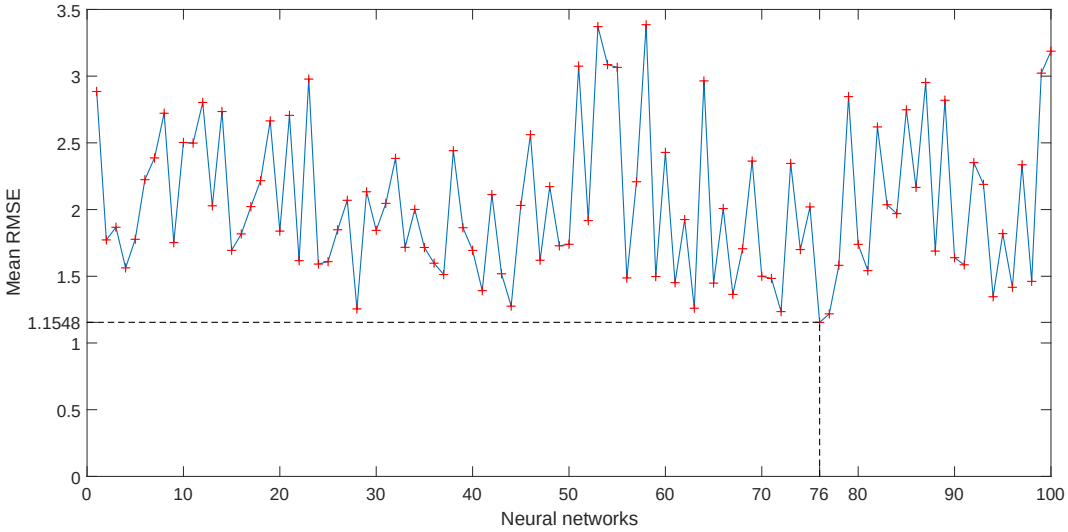

Figure 13: Performances of the 100 neural networks during the iterative training-testing process: the 76th network (GA-optimised network2) has the smallest mean RMSE

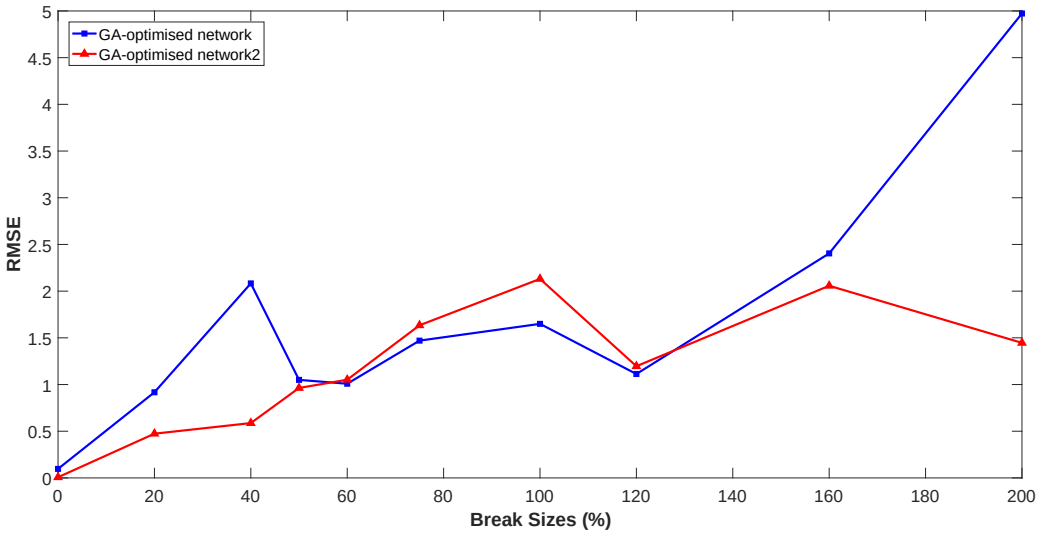

Figure 14: Performance comparison of the GA-optimised network2 and the GA-optimised network on the break size dataset

ECCS. In contrast, this work focuses on detecting the beak size of an IH of the PHT and the GA-optimised network2 detects a break size of an IH rather than the location of the break and the availability of ECCS. The performance of the GA-optimised network2 is compared with the performance of the Santhosh et al's neural network with regard to break size detection (Figure 15). The GA-optimised network2 has smaller RMSEs than the Santhosh et al's neural network with the largest difference in RMSE being 3.3538 at the break size 200% and the smallest difference being 0.1928 at the break size 0% (Figure 15). The mean RMSE (1.1548) of the GA-optimized network2 is smaller than that of the Santhosh et al's neural network (2.9167). The GA-optimised network2 has a significantly more stable performance (standard deviation 0.6860) than the Santhosh et al's neural network (standard deviation 1.5677) in detecting different break sizes. However, it may be noted that the RMSE in Santhosh et al's neural network has been computed based on

3 outputs: the break size, the location of the break and the status of the ECCS.

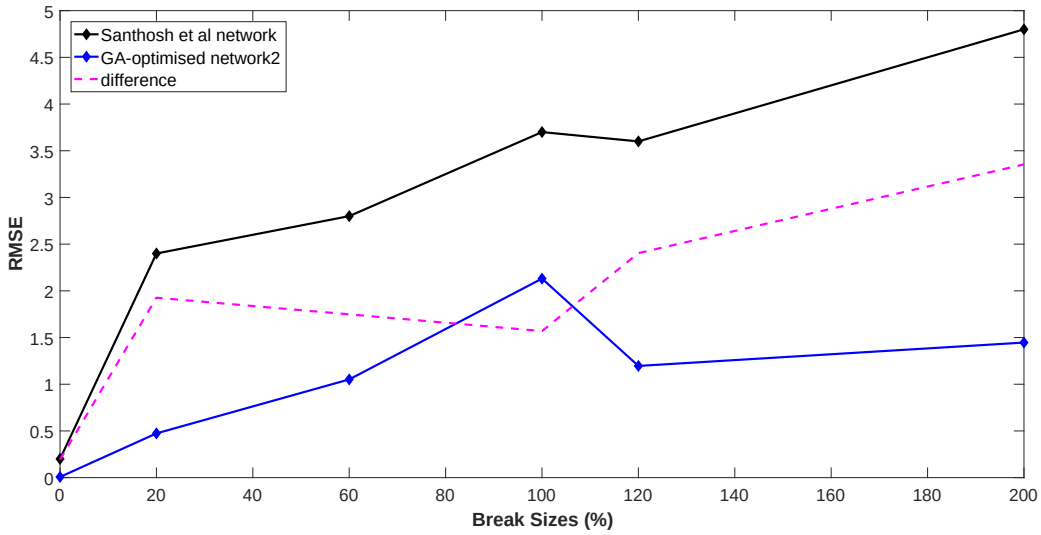

Figure 15: Performance comparison of the GA-optimised network2 and the Santhosh et al's neural network on the break size dataset

5.1.3. Performance Comparison with the Optimal Random Neural Network

A random search method was developed to generate 24000 random 2-hidden layer network architectures with between 5 to 36 inputs and between 5 to 40 neurons in each hidden layer. The mean RMSE of the optimal random network among the 24000 networks is 1.8008. The optimal random network consists of 27 inputs, 19 neurons in the 1st hidden layer, 15 neurons in the 2nd hidden layer, 1 output and 813 weights. In order to optimize the weights of the optimal random network architecture, it was trained iteratively 100 times on the merged dataset consisting of the break size dataset and the linear interpolation dataset. The best network among the 100 networks was obtained after 100 iterations of the training-testing process. The best network (optimal random network2) has a mean RMSE of 1.4617and standard deviation of RMSE of 0.8892. The RMSE of each break size of the optimal random network2 is illustrated in Table 4.

Break Size	0%	20%	40%	50%	60%	75%	100%	120%	160%	200%	SD
RMSE	0.1386	0.7277	1.0083	1.2078	0.9320	1.7137	1.9317	2.1155	3.3495	1.4920	0.8892

Table 4: The performance of the optimal random network2 in detecting each break size: SD (standard deviation)

The performance of the GA-optimised network2 is compared with the performance of the optimal random network2 (Figure 16). The GA-optimised network2 has smaller RMSEs than the optimal random network2 on all the 10 break sizes except the break size 60% with the largest difference in RMSE being 2.6878 at break size 100% (Figure 16, Table 3 and Table 4). The mean RMSE (1.1548) of the GA-optimised network2 is smaller than that of the optimal random network2 (1.4617). The GA-optimised network2 has a more stable performance (standard deviation 0.6860) than the optimal random network2 (standard deviation 0.8892) in detecting different break sizes (Table 4).

5.1.4. Performance Comparison with the Optimal Neural Network of Exhaustive Search

There are in total 1296 2-hidden layer network architectures with all the 37 inputs and between 5 to 40 neurons in each hidden layer. Exhaustive search was used to generate all the 1296 network architectures. The optimal network of exhaustive search consists of 37 inputs, 24 neurons in the 1st hidden layer, 19 neurons in the 2nd hidden layer, 1

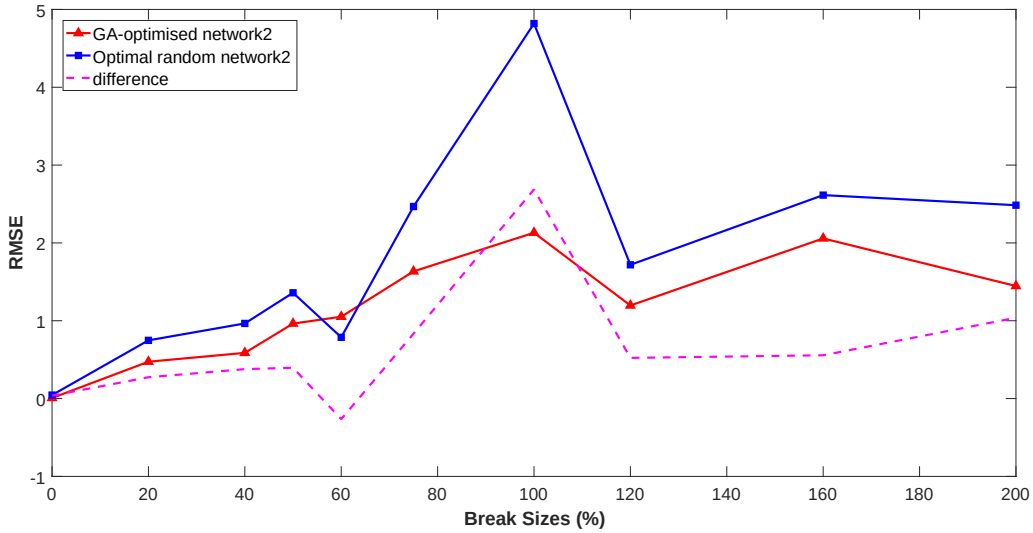

Figure 16: Performance comparison of GA-optimised network2 and optimal random network2 on the break size dataset

output and 1363 weights. The mean RMSE of the optimal network is 2.0427. In order to optimize the weights of the optimal network, it was trained iteratively 100 times on the merged dataset consisting of the break size dataset and the linear interpolation dataset. The best network (optimal network2) among the 100 networks was obtained after 100 iterations of the training-testing process. Optimal network2 has a mean RMSE of 1.1732 and standard deviation of RMSE of 0.6625. The RMSE of each break size of optimal network2 is illustrated in Table 5.

Break Size	0%	20%	40%	50%	60%	75%	100%	120%	160%	200%	SD
RMSE	0.635	0.7887	0.7876	1.1775	0.6924	1.03	1.8312	1.0981	2.7863	0.905	0.6625

Table 5: The performance of optimal network2 in detecting each break size : SD (standard deviation)

The performance of the GA-optimised network2 is compared with the performance of the optimal network2 (Figure 17). The mean RMSE (1.1548) of the GA-optimised network2 is smaller than that of the optimal network2 (mean RMSE 1.1732). The GA-optimised network2 has smaller RMSEs than the optimal network2 on the 5 break sizes 0%, 20%, 40%, 50% and 160% with the largest difference in RMSE being 0.6278 at break size 0% (Figure 17, Table 3 and Table 5). The optimal network2 has smaller RMSEs than the GA-optimised network2 on the other break sizes 60%, 75%, 100%, 120% and 200%. The GA-optimised network2 has a slightly less stable performance (standard deviation 0.6860) than optimal network2 (standard deviation 0.6625) in detecting different break sizes. However, GA-optimised network2 has a smaller number of inputs and a smaller number of weights than optimal network2. Therefore, it is expected that GA-optimised network2 has a better generalization performance than optimal network2.

5.1.5. Performance Comparison with RBF Kernel Support Vector Regression

Radial basis function (RBF) kernel Support vector regression (SVR) is a well-established approach for regression problems [45, 18, 37, 44]. During SVR training, the RBF kernel maps the training patterns to a high dimensional space (the feature space); then a linear regression is fitted to the transformed training patterns by solving a constraint optimization problem. The sequential minimal optimization (SMO) is a well-known algorithm for training SVR. The main parameters of a RBF kernel SVR are the epsilon in the insensitive loss function of the SVR, the tolerance of the termination criterion of SMO, the gamma of the RBF kernel and the cost of incorrect prediction in the objective function. The epsilon was set to 0.001; the tolerance was set to 0.002; the value of gamma was chosen from the set $\{0.0025, 0.005, 0.075, 0.1, 0.125, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5\}$; the cost was chosen from the set

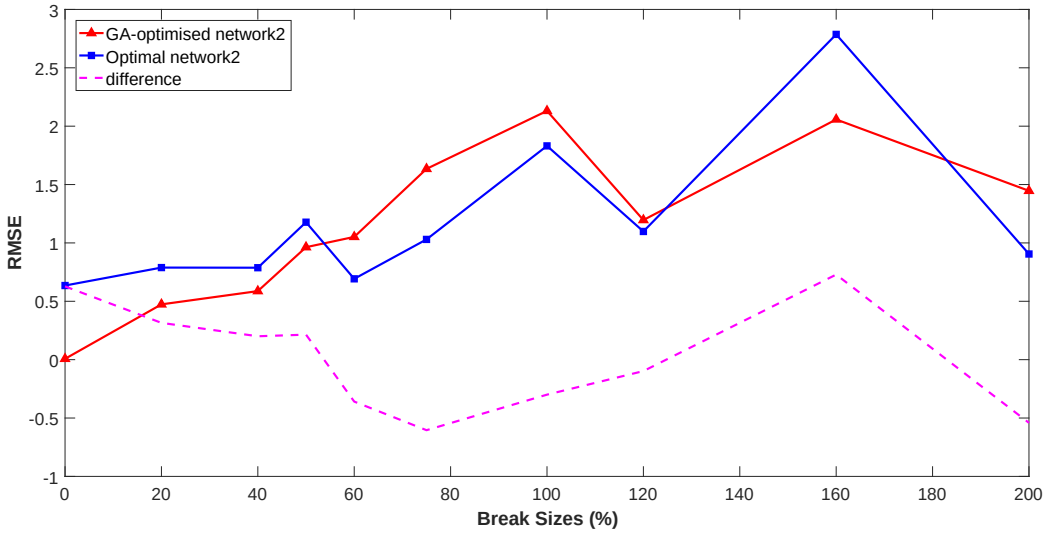

Figure 17: Performance comparison of GA-optimised network2 and optimal network2 on the break size dataset

$\{0.0025, 0.005, 0.05, 0.1, 0.5, 1, 5, 10, 15, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250\}$. The optimal setting of the gamma and the cost was obtained by searching through all the 285 combinations of gamma and cost. The setting of the gamma to 0.5 and the cost to 5 gives the smallest mean RMSE of 3.4525 (Figure 18). The performance of the GA-optimised network2 is compared with the performance of the RBF kernel SVR (Figure 19). For all the 10 break sizes, the GA-optimised network2 has smaller RMSEs than the optimal random network2 with the largest difference in RMSE being 6.66 at break size 160% (Figure 19). The mean RMSE of the GA-optimised network2 (1.1548) is smaller than that of the RBF kernel SVR (3.4525).

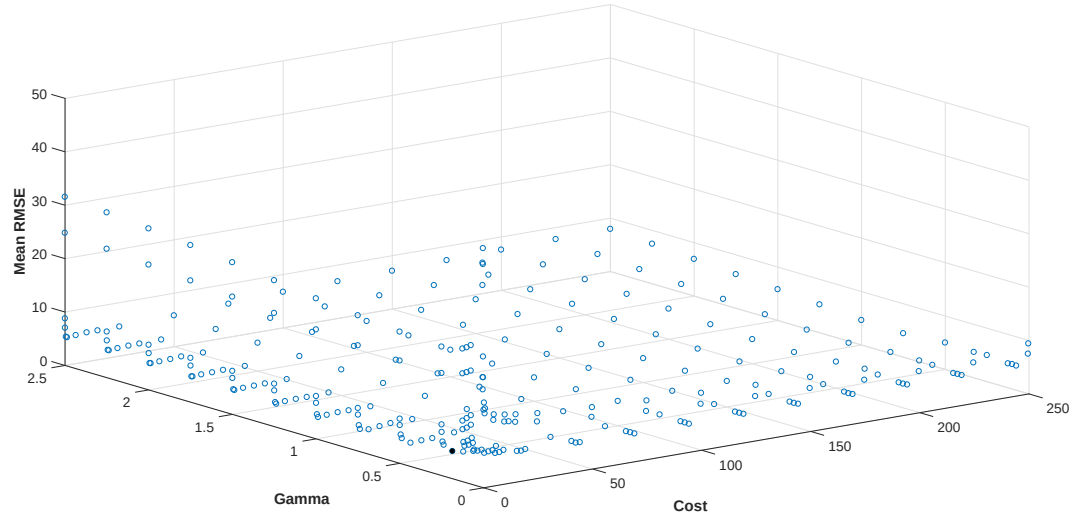

Figure 18: The performance of the RBK kernel SVR using all the combinations of gamma and cost on the break size dataset: the coloured point corresponds to the smallest mean RMSE of 3.4525

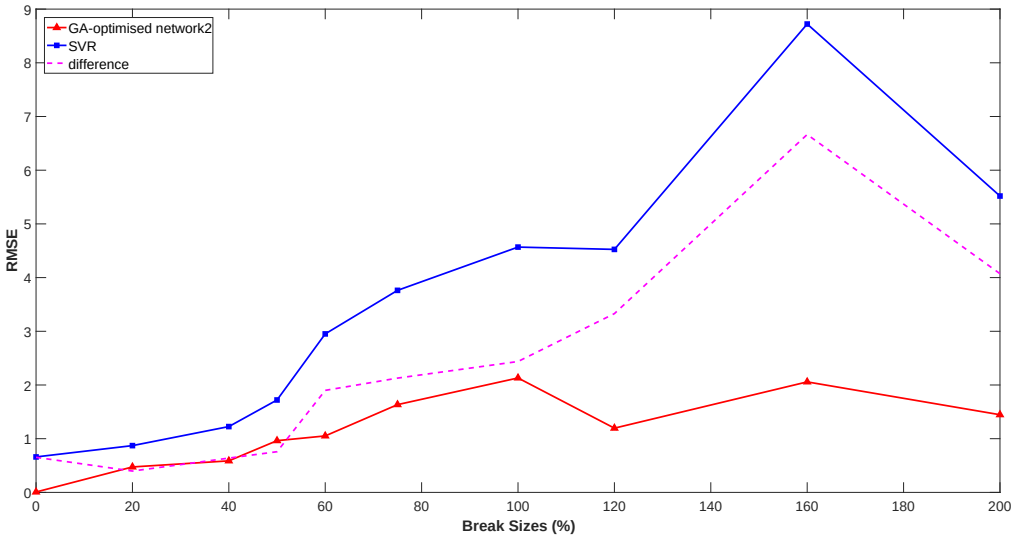

Figure 19: Performance comparison of the GA-optimised network2 and the RBF kernel SVR

For the break size dataset, the GA-optimised network2 has a better generalization performance than the Santhosh et al's neural network, the 2 optimal random networks, the 2 optimal networks of exhaustive search and RBF kernel SVR (Table 6). Moreover, the GA-optimised network and the GA-optimised network2 have fewer weights and hidden neurons than the Santhosh et al's network, the optimal network of exhaustive search and the optimal network2 of exhaustive search. Therefore, the the GA-optimised network and the GA-optimised network2 have simpler architectures than the Santhosh et al's network, the optimal network of exhaustive search and the optimal network2 of exhaustive search. The optimal random network and the optimal random network2 have simpler architectures and higher mean RMSEs than the GA-optimised network and the GA-optimised network2. This indicates that the optimal random network and the optimal random network2 underfit the training set.

Algorithm	Architecture	Weights	Mean RMSE
GA-optimised network	24,33,8,1	1064	1.6764
GA-optimised network2	24,33,8,1	1064	**1.1548**
Santhosh's neural network	37,26,19,3	1513	2.9167
Optimal random network	27,19,15,1	813	1.8008
Optimal random network2	27,19,15,1	813	1.4618
Optimal network of exhaustive search	37,24,19,1	1363	2.0467
Optimal network2 of exhaustive search	37,24,19,1	1363	1.1732
SVR	RBF kernel	cost:5 gamma:0.5	3.4525

Table 6: Performance comparison of the GA and the other approaches on the break size dataset

5.1.6. The Effect of the Elitism Rate and the Population Size on the Performance of the Genetic Algorithm

In order to investigate the effect of the population size and the elitism rate on the performance of the GA, the population size was set to 30, 50 and 80 respectively; the elitism rate was set to 0, 0.0125, 0.05 and 0.1 respectively; the maximum generation was set to 100 with the settings of the other parameters unchanged. Firstly, the population size was set to 30 and the elitism rate was set to 0, 0.0125, 0.05 and 0.1 respectively to obtain 4 GA-optimised networks. Each of the GA-optimised networks was obtained after running the GA for 100 generations. The best

Population size	Elitism rate	mean RMSE	generations
30	0	1.8204	60
	0.0125	**1.3235**	17
	0.05	1.8018	22
	0.1	2.0055	32
50	0.0125	1.4081	33
80	0.0125	1.6677	82

Table 7: Performance comparison of the GA-optimised networks on the break size dataset using the different setttings of population size and elitism rate: the 4th column lists the numbers of generations after which the GA first found the optimised networks

elitism rate is 0.0125 because it corresponds to the GA-optimised network with the highest performance i.e. mean RMSE of 1.3235 (Table 7). Moreover, setting the elitism rate to 0.0125 resulted in the fastest speed of finding the best GA-optimised network (17 generations) (Table 7). Then, with the best elitism rate of 0.0125, the population size was set to 50 and 80 respectively. The best GA-optimised network was obtained using the population size of 30 and the elitism rate of 0.0125 (Table 7). The best GA-optimised network consists of 27 inputs, 40 neurons in the 1st hidden layer, 5 neurons in the 2nd hidden layer, 1 output and 1285 weights. The mean RMSE of the best GA-optimised network is 1.3235. The RMSE of each break size of the best GA-optimised network is illustrated in Table 2. The performance of the best GA-optimised network is higher than that of the GA-optimised network (mean RMSE of 1.6764) found with 300 generations (Section 5.1.1).

5.2. Performance Evaluation using the Skillcraft Dataset

The proposed GA was further evaluated using the skillcraft dataset of the UCI machine learning repository [4]. The problem is to predict the league index (an integer in the range [1,8]) of each game player [4] based on 18 features. The fitness function (equation 5) was defined based on RMSE instead of mean RMSE. There are 3395 instances in the skillcraft dataset.

5.2.1. Performance of the Genetic Algorithm

The GA was run for 300 generations. The RMSE of the best network found in each generation is illustrated in Figure 20. The RMSEs of the best networks decrease continuously until the 45th generation (Figure 20). This indicates that the performance of the best networks continuously improves until the 45th generation. Between the 46th and the 125th generations there is no improvement in the performance of the best networks. Between the 138th and the 300th generations there is no improvement in the performance of the best networks. The GA-optimised network is the best network with the highest performance. The GA-optimised network was obtained at the 135th generation and consists of 11 inputs, 17 neurons in the 1st hidden layer, 21 neurons in the 2nd hidden layer, 1 output and 565 weights. The RMSE of the GA-optimised network is 1.030 (Figure 20).

5.2.2. Performance Comparison with the Optimal Random Neural Network

Random search was used to generate 24000 random 2-hidden layer network architectures with between 5 to 17 inputs and between 5 to 40 neurons in each hidden layer. The optimal random network among the 24000 networks consists of 16 inputs, 22 neurons in the 1st hidden layer, 18 neurons in the 2nd hidden layer, 1 output and 766 weights. The RMSE of the optimal random network is 1.0656. The GA-optimised network has smaller RMSE (1.030) and smaller number of weights (565) than the optimal random network.

5.2.3. Performance Comparison with the Optimal Network of Exhaustive Search

There are in total 1296 2-hidden layer network architectures with all the 18 inputs and between 5 to 40 neurons in each hidden layer. Exhaustive search was used to generate all the 1296 network architectures. The optimal network of exhaustive search consists of 18 inputs, 27 neurons in the 1st hidden layer, 20 neurons in the 2nd hidden layer, 1 output and 1046 weights. The RMSE of the optimal network is 1.0932. The GA-optimised network has smaller RMSE (1.030) and smaller number of weights (565) than the optimal network of exhaustive search.

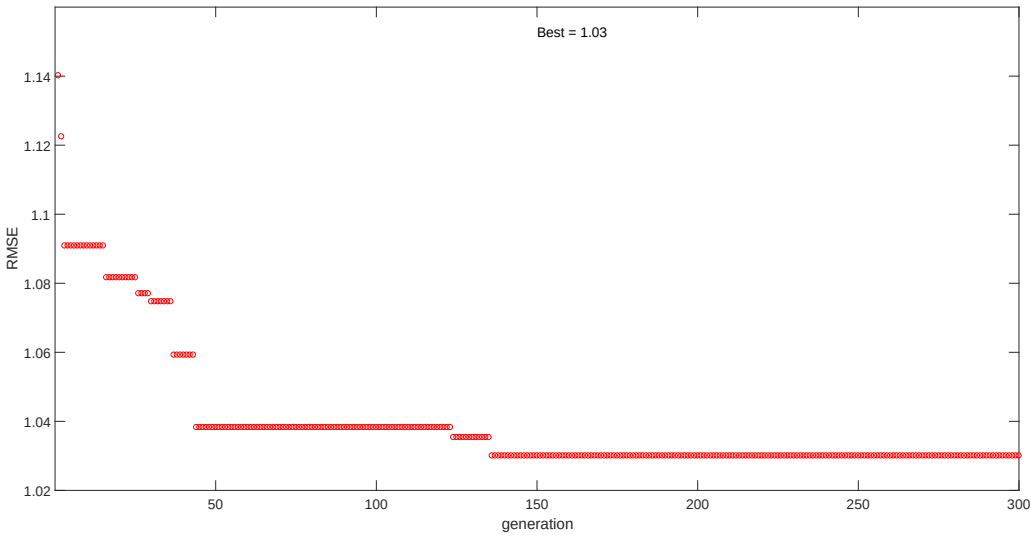

Figure 20: The RMSE of the best network of each generation on the skillcraft dataset: the GA-optimised network was found at the 135th generation

5.2.4. Performance Comparison with RBF Kernel Support Vector Regression

The parameter settings of the RBF kernel SVR used for the break size dataset were used for skillcraft dataset. The optimal setting of the gamma and the cost was obtained by searching through all the 285 combinations of gamma and cost. The setting of the gamma to 0.005 and the cost to 0.5 gives the smallest RMSE of 0.9977 (Figure 21). The RMSE of the RBF kernel SVR (0.9977) is slightly smaller than that of the GA-optimised network2 (1.030).

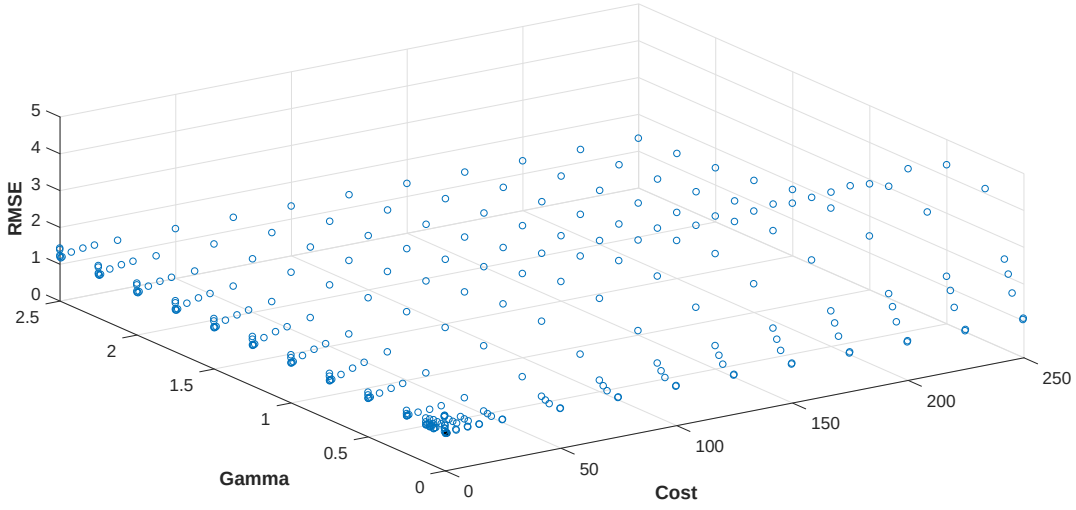

Figure 21: The performance of the RBF kernel SVR using all the combinations of gamma and cost on the skillcraft dataset: the coloured point corresponds to the smallest RMSE of 0.9977

For the skillcraft dataset, the GA-optimised neural network has a better generalization performance and a smaller architecture than the optimal random neural network and the optimal neural network of exhaustive search (Table 8);

the GA-optimised neural network has a similar performance to the RBF kernel SVR.

Algorithm	Architecture	Weights	RMSE
GA-optimised network	11,17,21,1	565	1.030
Optimal random network	16,22,18,1	766	1.0656
Optimal network of exhaustive search	18,27,20,1	1046	1.0932
SVR	RBF kernel	cost:0.5 gamma:0.005	**0.998**

Table 8: Performance comparison of the GA and the other approaches on the skillcraft dataset

5.2.5. The Effect of Elitism and Population Size on the Performance of the Genetic Algorithm

In order to investigate the effect of the population size and the elitism rate on the performance of the GA, the population size was set to 30, 50 and 80 respectively; the elitism rate was set to 0, 0.0125, 0.05 and 0.1 respectively; the maximum generation was set to 100 with the settings of the other parameters unchanged. Firstly, the population

Population size	**Elitism rate**	**RMSE**	**generations**
30	0	0.8897	85
	0.0125	0.8972	23
	0.05	0.8905	15
	0.1	**0.8752**	65
50	0.1	0.8805	72
80	0.1	**0.8679**	56

Table 9: Performance comparison of the GA-optimised networks on the skillcraft dataset using the different settings of population size and elitism rate: the 4th column lists the numbers of the generations after which the GA first found the optimised networks

size was set to 30 and the elitism rate was set to 0, 0.0125, 0.05 and 0.1 respectively to obtain 4 GA-optimised networks. Each of the GA-optimised networks was obtained after running the GA for 100 generations. The best elitism rate is 0.1 because it corresponds to the GA-optimised network with the highest performance i.e. mean RMSE of 0.8752 (Table 9). However, setting the elitism rate to 0.05 resulted in the fastest speed of finding a GA-optimised network (15 generations) (Table 9). Then, with the best elitism rate of 0.1, the population size was set to 50 and 80 respectively. The best GA-optimised network was obtained using the population size of 80 and the elitism rate of 0.1 (Table 9). The best GA-optimised network consists of 11 inputs, 5 neurons in the 1st hidden layer, 5 neurons in the 2nd hidden layer, 1 output and 85 weights. The RMSE of the best GA-optimised network is 0.8679. The performance of the best GA-optimised network is higher than that of the GA-optimised network (RMSE of 1.030) found with 300 generations (Section 5.2.1).

6. Discussion

The good performance of the proposed GA is due to the following:

1. An initial population of network architectures of high performance is created using constraint satisfaction.
2. During each generation, a great diversity of neural network architectures are created by independently breeding offspring from the sub-population of the feature subsets of the neural network architectures in the parents population and breeding offspring from the sub-population of the 2-hidden layer architectures of the neural network architectures.
3. During breeding offspring from the sub-population of the 2-hidden layer architectures, offspring of high performance are created using the proposed constraint-based nearest neighbour search algorithm.

Optimizing the elitism rate or/and the size of the population can improve the performance of the GA. Moreover, setting the elitism rate to optimal values can significantly improve the speed of finding an optimised network. Optimizing the elitism rate may be preferred over optimizing the population size because increasing the size of the population leads to a higher number of fitness evaluations i.e. a higher computational cost, whereas adjusting the elitism rate does not significantly vary the computational cost as the size of the population is kept constant.

One limitation of the proposed GA is that it is not guaranteed to find the global optimal solutions (the global optimal network architectures) because the GA as a metaheuristic explores a sub-space of the search space of all the possible network architectures to find local optimal network architectures. This limitation can be alleviated by making the GA exploring a larger sub-space containing a larger number of diverse network architectures of high performance. This can be achieved by either increasing the size of the initial population or/and optimizing the parameters of the GA.

7. Conclusion and Future Work

This work has proposed a constraint-based genetic algorithm to optimize neural network architectures for detecting LOCA of NPPs. The proposed approach has achieved a high performance in detecting LOCA of NPPs and in predicting the league of game players. The proposed approach is a suitable approach for regression problems of different domains. Constraint satisfaction is an effective approach to create neural network architectures of high performances by modelling the user's prior knowledge about neural network architectures of high performances. Independent breeding of inputs and 2-hidden layer architectures can create a great diversity of network architectures.

The challenges of the GA are to improve the performance of the GA-optimised neural network and the speed of finding the GA-optimised neural network. The performance of the GA can be improved by optimizing the values of its parameters. The GA can be extended to make it self-tune its parameters during the evolutionary search process using a fuzzy system in order to find better quality solutions and improve the speed of convergence. The fuzzy system can be developed to dynamically adjust the elitism rate, the crossover rates and the mutation rates when the performance of the optimised neural network is not satisfactory after a number of generations. For each parameter of the GA, a set of fuzzy rules can be defined to tune that parameter based on the improvement in the performance achieved by the optimised neural network of the current generation compared the optimised neural network of a previous generation. For example, the following fuzzy rules can be defined to tune the elitism rate:

If improvement_in_performance is *Zero* and elitism is *Small* then elitism is *Medium*
If improvement_in_performance is *Zero* and elitism is *Medium* then elitism is *Large*
If improvement_in_performance is *Zero* and elitism is *Large* then elitism is *Small*
If improvement_in_performance is *Small* and elitism is *Small* then elitism is *Medium*
If improvement_in_performance is *Small* and elitism is *Medium* then elitism is *Large*
If improvement_in_performance is *Small* and elitism is *Large* then elitism is *Small*
If improvement_in_performance is *Large* and elitism is *Small* then elitism is *Small*
If improvement_in_performance is *Large* and elitism is *Medium* then elitism is *Medium*
If improvement_in_performance is *Large* and elitism is *Large* then elitism is *Large*

To model the values of the parameters, different fuzzy sets e.g. *Zero*, *Small*, *Medium* and *Large* can be defined.

The speed of finding the GA-optimised neural network can be improved by reducing the computational cost of the GA. The computational cost of the GA is dominated by the number of fitness evaluations needed to find an optimal network architecture. The number of the fitness evaluations can be reduced using a fitness approximation method such as adaptive fuzzy fitness granulation (AFFG) [5]. In AFFG, an adaptive pool of fuzzy granules with exactly computed fitness is maintained. If a new individual is sufficiently similar to a fuzzy granule, then the fitness of that granule is used as an estimate of the individual. Otherwise, that individual is added to the pool as a new fuzzy granule.

The performance of the GA-optimised neural network can be improved by constructing deep neural networks of more than 2 hidden layers because deep neural networks can have better approximation capabilities than 2-hidden layer networks. The proposed GA can be extended to find optimal deep network architectures as follows. The chromosome representation can be extended to represent deep network architectures. The architectures of deep networks can be modeled as CSPs. The random walk heuristic can be extended to solve the CSPs. The constraint-based nearest neighbour search can be extended to find the nearest neighbours of deep network architectures.

Constraint satisfaction can be combined with other metaheuristics such as simulated annealing, ant colony optimization (ACO) and particle swarm optimisation (PSO) to improve the performance of optimised neural networks for detecting LOCA of NPPs and regression problems of other domains.

Acknowledgements

The authors would like to thank the editors and the reviewers for their valuable comments and helpful suggestions.The authors would like to thank the Engineering and Physical Sciences Research Council (EPSRC) of UK for their financial support under the grant number of EP/M018717/1.

References

[1] *Procedures for Conducting Probabilistic Safety Assessments of Nuclear Power Plants (Level 1)*. Safety Series, IAEA, 1992.

[2] *Safety Margins of Operating Reactors: Analysis of Uncertainties and Implication for Decision Making*. Technical Report IAEA-TECDOC-1332, International Atomic Energy Agency (IAEA), 2003.

[3] *ECLiPSe CLP website*. http://eclipseclp.org/index.html, 2018.

[4] *UCI Machine Learning Repository*. https://archive.ics.uci.edu/ml/datasets.html, 2018.

[5] M.-R. Akbarzadeh-T, M. Davarynejad, and N. Pariz. Adaptive fuzzy fitness granulation for evolutionary optimization. *International Journal of Approximate Reasoning*, 49(3):523 – 538, 2008.

[6] Cagdas Hakan Aladag. A new architecture selection method based on tabu search for artificial neural networks. *Expert Systems with Applications*, 38(4):3287 – 3293, 2011.

[7] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6), 2017.

[8] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[9] Krzysztof R. Apt and Mark Wallace. *Constraint Logic Programming Using Eclipse*. Cambridge University Press, 2007.

[10] Ju Hyun Back, Kwae Hwan Yoo, Geon Pil Choi, Man Gyun Na, and Dong Yeong Kim. Prediction and uncertainty analysis of power peaking factor by cascaded fuzzy neural networks. *Annals of Nuclear Energy*, 110(Supplement C):989 – 994, 2017.

[11] A. J. M. A. P. Bandara and N. G. J. Dias. Optimizing neural network architectures for image recognition using genetic algorithms. In *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2015.

[12] Piero Baraldi, Roozbeh Razavi-Far, and Enrico Zio. Bagged ensemble of fuzzy c-means classifiers for nuclear transient identification. *Annals of Nuclear Energy*, 38(5):1161–1171, 2011.

[13] Eric B. Bartlett and Robert E. Uhrig. Nuclear power plant status diagnostics using an artificial neural network. *Nuclear Technology*, 97:272–281, 1992.

[14] Mark Bartlett and James Cussens. Integer linear programming for the bayesian network structure learning problem. *Artificial Intelligence*, 24:258–271, 2017.

[15] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[16] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] Rob Buckingham and Andrew Graham. Nuclear snake-arm robots. *Industrial Robot: the international journal of robotics research and application*, 39(1):6–11, 2012.

[18] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[19] Andrew Chipperfield, Peter Fleming, Hartmut Pohlheim, and Carlos Fonseca. The matlab genetic algorithm toolbox v1.2 user's guide, 1994.

[20] Gagliardi Francesco Ciancio Claudio, Ambrogio Giuseppina and Musmanno Roberto. Heuristic techniques to optimize neural network architecture in manufacturing applications. *Neural Computing & Applications*, 27(7):2001 – 2015, 2016.

[21] James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 153–160. AUAI Press, 2011.

[22] James Cussens, Matti Jrvisalo, Janne H. Korhonen, and Mark Bartlett. Bayesian network structure learning with integer programming: Polytopes, facets and complexity. *Journal of Artificial Intelligence Research (JAIR)*, 58:185–229, 2017.

[23] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003.

[24] Thomas A. Ferguson and Lixuan Lu. Fault tree analysis for an inspection robot in a nuclear power plant. *IOP Conference Series: Materials Science and Engineering*, 235(1):012003, 2017.

[25] Martha Dais Ferreira, Dbora Cristina Corra, Luis Gustavo Nonato, and Rodrigo Fernandes de Mello. Designing architectures of convolutional neural networks to solve practical problems. *Expert Systems with Applications*, 94:205 – 217, 2018.

[26] Antonio C.F. Guimares and Celso M.F. Lapa. Adaptive fuzzy system for fuel rod cladding failure in nuclear power plant. *Annals of Nuclear Energy*, 34(3):233 – 240, 2007.

[27] Zhichao Guo and Robert E. Uhrig. Use of artificial neural networks to analyse nuclear power plant performance. *Nuclear Technology*, 99:36–42, 1992.

[28] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[29] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, Cambridge, MA, USA, 2001.

[30] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[31] LE VAN HONG and B. CHATTERJEE. Large loca analysis of indian pressurized heavy water reactor 220 mwe. *Nuclear Science and Technology*, 1:12–17, 2002.
[32] M. A. J. Idrissi, H. Ramchoun, Y. Ghanou, and M. Ettaouil. Genetic algorithm for neural network architecture optimization. In *2016 3rd International Conference on Logistics Operations Management (GOL)*, pages 1–4, 2016.
[33] Hazewinkel M. *Linear interpolation. In: Hazewinkel M (ed) Encyclopedia of Mathematics*. Springer, Netherlands, 2001.
[34] Francesco Di Maio, Claudia Picoco, Enrico Zio, and Valentin Rychkov. Safety margin sensitivity analysis for model selection in nuclear power plant probabilistic safety assessment. *Reliability Engineering & System Safety*, 162:122 – 138, 2017.
[35] Patricia Melin and Daniela Snchez. Multi-objective optimization for modular granular neural networks applied to pattern recognition. *Information Sciences*, 2017.
[36] Man Gyun Na, Sun Ho Shin, Dong Won Jung, Soong Pyung Kim, Ji Hwan Jeong, and Byung Chul Lee. Estimation of break location and size for loss of coolant accidents using neural networks. *Nuclear Engineering and Design*, 232(3):289–300, 2004.
[37] John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, 1999.
[38] Wilson Q. Wang, M Golnaraghi, and Fathy Ismail. Prognosis of machine health condition using neuro-fuzzy systems. 18:813–831, 2004.
[39] Miguel Rocha, Paulo Cortez, and Jos Neves. Evolution of neural networks for classification and regression. *Neurocomputing*, 70(16):2809 – 2816, 2007.
[40] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.
[41] T.V. Santhosh, M. Kumar, I. Thangamani, A. Srivastava, A. Dutta, V. Verma, D. Mukhopadhyay, S. Ganju, B. Chatterjee, V.V.S.S. Rao, H.G. Lele, and A.K. Ghosh. A diagnostic system for identifying accident conditions in a nuclear reactor. *Nuclear Engineering and Design*, 241(1):177 – 184, 2011.
[42] Piercesare Secchi, Enrico Zio, and Francesco Di Maio. Quantifying uncertainties in the estimation of safety parameters by using bootstrapped artificial neural networks. *Annals of Nuclear Energy*, 35(12):2338 – 2350, 2008.
[43] Randall S. Sexton, Robert E. Dorsey, and John D. Johnson. Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. *European Journal of Operational Research*, 114(3):589 – 601, 1999.
[44] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R.K. Murthy. Improvements to the smo algorithm for svm regression. *Trans. Neur. Netw.*, 11(5):1188–1193, 2000.
[45] Alex J. Smola and Bernhard Scholkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.
[46] Daniela Snchez and Patricia Melin. Optimization of modular granular neural networks using hierarchical genetic algorithms for human recognition using the ear biometric measure. *Engineering Applications of Artificial Intelligence*, 27:41 – 56, 2014.
[47] Symone Soares, Carlos Henggeler Antunes, and Rui Arajo. Comparison of a genetic algorithm and simulated annealing for automatic neural network ensemble development. *Neurocomputing*, 121:498 – 511, 2013.
[48] T.J. Souza, J.A.C.C. Medeiros, and A.C. Gonalves. Identification model of an accidental drop of a control rod in pwr reactors using thermocouple readings and radial basis function neural networks. *Annals of Nuclear Energy*, 103(Complete):204–211, 2017.
[49] The RELAP5-3D Code Development Team. *RELAP5-3D Code Manual Volume V: User's Guidelines*. Idaho National Laboratory, USA, 2014.
[50] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
[51] B. u. Islam, Z. Baharudin, M. Q. Raza, and P. Nallagownden. Optimization of neural network architecture using genetic algorithm for load forecasting. In *2014 5th International Conference on Intelligent and Advanced Systems (ICIAS)*, pages 1–6, 2014.
[52] Muhammet Unal, Mustafa Onat, Mustafa Demetgul, and Haluk Kucuk. Fault diagnosis of rolling bearings using a genetic algorithm optimized neural network. *Measurement*, 58:187 – 196, 2014.
[53] Fevrier Valdez, Patricia Melin, and Oscar Castillo. Modular neural networks architecture optimization with a new nature inspired method using a fuzzy combination of particle swarm optimization and genetic algorithms. *Information Sciences*, 270:143 – 153, 2014.
[54] Y. Wang, H. Shen, and D. Duan. On stabilization of quantized sampled-data neural-network-based control systems. *IEEE Transactions on Cybernetics*, 47(10):3124–3135, 2017.
[55] Y. Wei, J. H. Park, H. R. Karimi, Y. C. Tian, and H. Jung. Improved stability and stabilization results for stochastic synchronization of continuous-time semi-markovian jump neural networks with time-varying delay. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2018.
[56] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[57] Wei X, Wan J, and Zhao F. Prediction study on pci failure of reactor fuel based on a radial basis function neural network. *Science and Technology of Nuclear Installations*, pages 1–6, 2016.
[58] Jian Ye, Junfei Qiao, Ming ai Li, and Xiaogang Ruan. A tabu based neural network learning algorithm. *Neurocomputing*, 70(4):875 – 882, 2007.
[59] Enrico Zio, Francesco Di Maio, and Marco Stasi. A data-driven approach for predicting failure scenarios in nuclear systems. *Annals of Nuclear Energy*, 37(4):482–491, 2010.