Available online at www.sciencedirect.com
SCIENCE DIRECT
Pergamon
Cement and Concrete Research 34 (2004) 2083–2090
CEMENT AND CONCRETE RESEARCH

Prediction of cement strength using soft computing techniques

Adil Baykasoğlu*, Türkay Dereli, Serkan Tanış

Department of Industrial Engineering, University of Gaziantep, 27310 Gaziantep, Turkey

Received 26 August 2003; accepted 9 March 2004

Abstract

In this paper, it is aimed to propose prediction approaches for the 28-day compressive strength of Portland composite cement (PCC) by using soft computing techniques. Gene expression programming (GEP) and neural networks (NNs) are the soft computing techniques that are used for the prediction of compressive cement strength (CCS). In addition to these methods, stepwise regression analysis is also used to have an idea about the predictive power of the soft computing techniques in comparison to classical statistical approach. The application of the genetic programming (GP) technique GEP to the cement strength prediction is shown for the first time in this paper. The results obtained from the computational tests have shown that GEP is a promising technique for the prediction of cement strength.

Keywords: Modelling; Compressive strength; Cement manufacture

1. Introduction

The standard 28-day compressive strength test is widely used for the characterisation of cement properties [1]. Compressive strength is the most important cement property, inasmuch as it is the main parameter for quality control [2]. It is a long time for the industry to wait for 28 days to get the experimental results for the compressive cement strength (CCS). Therefore, faster determination of CCS is a need for the cement industry and deserves research interest from the researchers.

There are mainly two different ways for CCS determination: (a) accelerated strength test methods and (b) use of mathematical models. The focus in this paper is on the second one. The most widely used mathematical approach in the past is to use simple regression models [1,2]. CCS depends on many different factors, which are chemical and physical in nature. Analytical models including the statistical ones (e.g., regression analysis) used to describe the effects of these factors on strength can be very complex [3]. Therefore, the use of soft computing techniques seems a promising approach to the CCS prediction problem. In this study, such an attempt is made by employing gene expression programming (GEP) and neural networks (NNs) for the prediction of the compressive strength of Portland composite cement (PCC). In addition to these techniques, stepwise regression analysis is also used to have an idea about the predicting power of these soft computing techniques in comparison to a classical statistical approach.

Soft computing techniques, namely, NNs and fuzzy logic, were already used in the literature for the prediction of CCS. However, the number of published papers on the subject is very small. Akkurt et al. [3] used NNs for the CCS prediction. They also analysed effects of various parameters on the 28-day strength. Fa-Liang [4] applied fuzzy logic to CCS prediction successfully. Other studies made use of regression analysis. Tsivilis and Parissakis [1] and de Siquera Tango [2] applied regression methods for CCS prediction. Interestingly, the number of papers that make use of the regression analysis is also very small. There is no published work in the literature that makes use of genetic programming (GP) approaches on the prediction of CCS. This paper makes such an attempt by using GEP [5] for the prediction of CCS.

In the following sections of this paper, GEP and NNs are briefly described; then, the model construction for GEP and NNs is explained along with the comparison and discussion of the obtained results.

* Corresponding author. Tel.: +90-342-3601200; fax: +90-342-3601100.
E-mail addresses: baykasoglu@gantep.edu.tr (A. Baykasoğlu), dereli@gantep.edu.tr (T. Dereli).

0008-8846/$ – see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cemconres.2004.03.028

2. Brief overview of GEP

In this section, a brief overview of GEP is given for motivation. For a detailed explanation of GEP, refer to Ferreira [5].

GP is proposed by Koza [6]. It is a generalization of genetic algorithms (GAs) [7]. The most general form of a solution to a computer-modelled problem is a computer program. GP takes cognizance of this and attempts to use computer programs as its data representation. Similarly to GA, GP needs only the problem to be defined. Then, the program searches for a solution in a problem-independent manner [6,7]. GEP is a natural development of GA and GP. GEP was invented by Ferreira [5]. Most of the genetic operators used in GAs can also be implemented in GP and GEP with minor changes. Like GP, there are mainly five components in GEP: the function set, terminal set, fitness function, control parameters, and stop condition that must be determined when using GEP to solve a problem. Unlike the parse-tree representation in canonical GP, GEP uses a fixed length of character strings to represent solutions to the problems, which are afterwards expressed as parse trees [called "expression tree" (ET) in GEP] of different sizes and shapes when evaluating their fitness. A brief definition of GEP is given next [8].

2.1. GEP genes and ETs

Each GEP gene is composed of a list of symbols with a fixed length that can be any element from a function set like {+, −, *, /, Sqrt} and the terminal set like {1, a, b, c, d}. A typical GEP gene with the given function and terminal sets can be

$$\underline{+*.\mathrm{Sqrt}.a.*.} + . + . - .c.a.d.\mathrm{Sqrt}.c.d.2 \quad (1)$$

where "." is used to separate elements for easy reading; Sqrt is the square-root function; 2 is a constant; and a, b, c, and d are variable (or attribute) names. The above is typically named Karva notation or K-expression [5]. A K-expression can be mapped into an ET following a width-first fashion. The conversation starts from the first position in the K-expression, which corresponds to the root of the ET, and reads through the string one by one. For each node (from left to right) in one layer in the ET, if it is a function with $m (m \geq 1)$ arguments, then the next m symbols in the K-expression are attached below it as m child branches. Otherwise, each terminal node forms a leaf of the ET. This tree-expanding process continues layer by layer until all leaf nodes in the ET are composed of elements from the terminal set. For example, the above sample gene can be expressed as Fig. 1(a), which can be further expressed in a mathematical form as

$$a[(c-d)(d+\sqrt{2})] + \sqrt{c+a} \quad (2)$$

The inverse process, i.e., the conversion of an ET into a K-expression, is also simple, just recording the nodes from left to right in each layer of the ET, from root layer down to the deepest one to form the string.

Like GP, the function set and terminal set must have the closure property: each function must able to take any value of data type which can be returned by a function or assumed by a terminal. For example, the division function must be protected to return a reasonable value in the event of division by zero.

As explained before, GEP genes have fixed length, which is predetermined for a given problem. Thus, in GEP, what varies is not the length of genes but the size of the corresponding ETs. This means that there exist a certain number of redundant elements, which are not useful for the genome-ET mapping. For example, the following gene

$$+. + . - .c.a.d.\mathrm{Sqrt}.c.d.2.\underline{+.*.\mathrm{Sqrt}.a.*.} \quad (3)$$

only rotates the first five elements of gene (1) to the end and has the same length of 15, but its valid K-expression size is

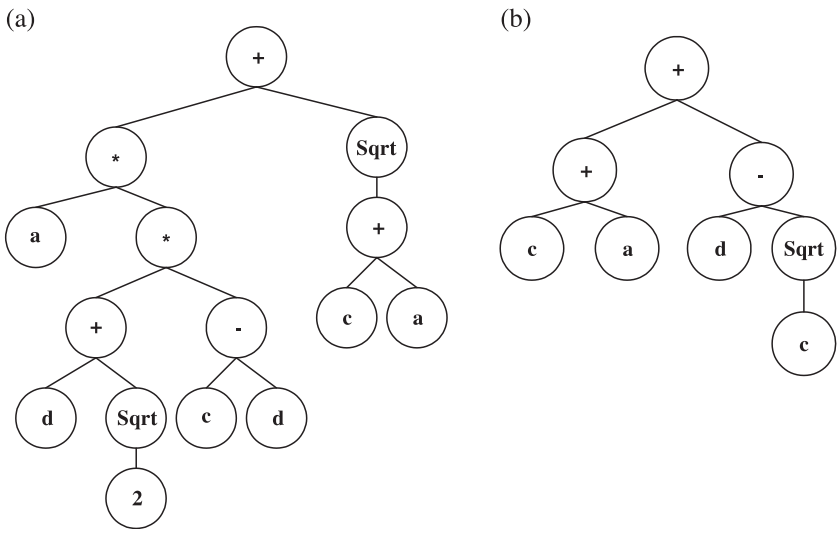

Fig. 1. Example of GEP ETs.

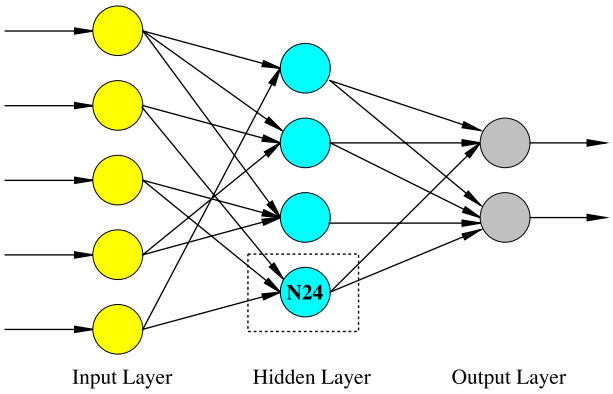

Fig. 2. A typical NN structure.

8; that is, only the first eight elements are used to construct the solution function (c + a)+(d- √ c), with the corresponding expression shown in Fig. 1(b).

Therefore, the valid length of a K-expression may be equal or less than the length of the gene. To guarantee the validity of a randomly selected genome, the original GEP technique employs a head–tail method. Each gene is composed of a head and a tail. The head may contain symbols from both the function set and the terminal set, whereas the tail contains only terminals [5].

2.2. GEP algorithm and operators

Like GA and GP, the GEP algorithm begins with the random generation of the fixed-length chromosome of each individual for the initial population. Then, the chromosomes are expressed and the fitness of each individual is evaluated. The individuals are then selected according to fitness to reproduce with modification. The individuals of this new generation are, in their turn, subjected to the same developmental process: expression of the genomes, confrontation of the selection environment, and reproduction with modification. The process is repeated for a certain number of generations or until a solution has been found [5–7].

In GEP, the individuals are often selected and copied into the next generation according to the fitness by roulette wheel sampling with elitism, which guarantees the survival and cloning of the best individual to the next generation. Variation in the population is introduced by conducting single or several genetic operators on selected chromosomes, which include:

- Crossover, in which two parent genomes are randomly chosen and paired to exchange some elements between them. There are two kinds of crossover: one-point and two-point crossovers, working in the same fashion as that in GAs.
- Mutation, which can happen with any times at any position in a genome, as long as the mutated individual passes the validity test. Note that like crossover, a mutation in the coding sequence of a gene usually drastically reshapes the ET.
- Rotation, in which two subparts of element sequence in a genome are rotated with respect to a randomly chosen point (this is similar to the inversion in GAs). Rotation can also drastically reshape the ETs, as shown in the example genes shown in Eqs. (1) and (3).

3. Brief overview of NNs

In this section, a brief presentation of the basic NNs is given for the novice readers. Many different resources are available in the literature for comprehensive explanations on NNs [9–11].

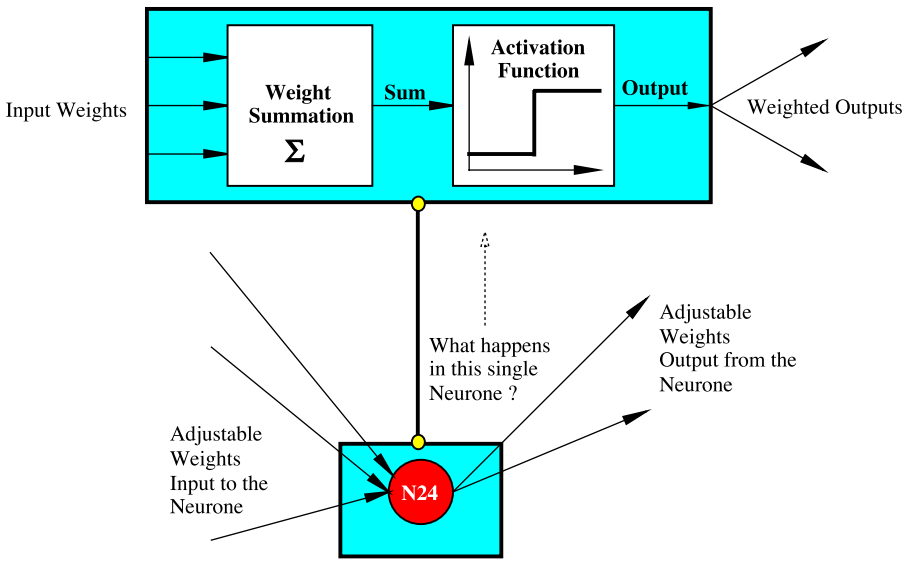

Fig. 3. An example of an artificial neurone unit.

NNs, also called parallel distributed processing systems (PDPs) and connectionist systems, are intended for modelling the organizational principles of the central nervous system, with the hope that biologically inspired computing capabilities of the NNs will allow the cognitive and sensory tasks more easily and more satisfactorily than with conventional serial processors [11]. The study of NNs is an attempt to understand the functionality of human brain. NN technology mimics the brain's own problem-solving process. NNs are systems composed of many simple processing elements operating in parallel, whose functions are determined primarily by the pattern of connectivity. These systems are capable of high-level functions, such as adaptation or learning, and lower-level functions, such as data preprocessing, for different kinds of inputs. NNs have been inspired both by biological nervous systems and by mathematical theories of learning, information processing, and control.

Adaptation or the ability to learn is the most important property of NNs. An NN can be trained to map a set of input patterns onto a corresponding set of output patterns simply by means of exposure to examples of the mapping. This training is performed by a gradient descent algorithm, which gradually adapts the internal weights of the network, so as to reduce differences between the actual network outputs (for given set of inputs) and the desired network outputs. These types of NNs can produce reasonable output vectors for input patterns outside of the set of training examples [11].

An NN may be considered as a black box that can accept a series of input data and produce from these one or more outputs [9]. An NN consists of many nonlinear computational elements operating in parallel and arranged in patterns resembling those of biological neurones. The neurones are interconnected into networks via weights, which are usually adapted in learning or training process to improve performance. Inputs are supplied to the input nodes. From these nodes, the input values are propagated through the links to the other region of the network. As they propagate, they are combined at common nodes and changed according to the computational rules of the links at nodes through which they pass. Outputs from the output nodes form the numeric outputs of the network. In summary, an NN takes an input numeric pattern and produces an output numeric pattern. A typical NN structure is illustrated in Fig. 2.

As depicted in Fig. 2, the basic components of NNs are neurones. A group of neurones is called a slab. Neurones are also grouped into layers by their connection to the outside world. If a neurone receives data from the outside world, it is considered to be in the input layer. If a neurone contains network's predictions or classifications, it is in the output layer. Neurones between the input and output layers are in the hidden layer(s). A layer may contain one ore more slabs of neurones. What happens in a single neurone (e.g., neurone N24) is shown in Fig. 3.

The neurones are usually characterised by an internal threshold and by the type of their activation (transfer) function. The most common transfer functions are hard-limiter, threshold-logic, and sigmoidal function being the most widely used transfer function. The neurones (nodes) in the input layer are fed with input data. Each node sums all the inputs, with one input per node in the input layer but many inputs per node in the next layer. Each node transfers

Table 1
The variables used in model construction

Code	Input variable	Code	Output variable
d_1	SiO_2 (%)	y	28-day CCS (MPa)
d_2	Loss on ignition (%)		
d_3	Al_2O_3 (%)		
d_4	Fe_2O_3 (%)		
d_5	CaO (%)		
d_6	MgO (%)		
d_7	SO_3 (%)		
d_8	Alumina modulus		
d_9	K_2O (%)		
d_{10}	Free lime (%)		
d_{11}	Litre weight (l/g)		
d_{12}	Percentage of composite (%)		
d_{13}	Sieve residue on 45 μm (%)		
d_{14}	Sieve residue on 90 μm (%)		
d_{15}	Specific surface (cm^2/g)		
d_{16}	Setting time (min)		
d_{17}	1-day CCS (MPa)		
d_{18}	2-day CCS (MPa)		
d_{19}	7-day CCS (MPa)		

Table 2
Parameters of the GEP algorithm

p_1	Number of generation	Between 3000 and 20,000
p_2	Population size	50
p_3	Function set	+, −, *, /, ^, √, e, log, sin, tan
p_4	Number of genes	1, 2, 3
p_5	Head size	5, 8, 10
p_6	Linking function	+, *
p_7	Mutation rate	0.014, 0.044, 0.084
p_8	One-point recombination rate	0.3, 0.5, 0.7
p_9	Two-point recombination rate	0.3, 0.5, 0.7
p_{10}	Gene recombination rate	0.1, 0.2, 0.4
p_{11}	Gene transposition rate	0.1, 0.2, 0.4

Table 3
The best and the worst results obtained from the GEP tests

GEP parameters									
p_4	p_5	p_7	p_8	p_9	p_{10}	p_{11}	p_6	p_1	R-square error on test data
1	8	0.014	0.7	0.7	0.4	0.4	+	6149	0.587
3	5	0.044	0.5	0.5	0.2	0.2	+	4415	0.625
2	8	0.014	0.7	0.7	0.4	0.4	*	3780	0.717
2	5	0.014	0.3	0.3	0.1	0.1	*	5391	0.734
2	5	0.084	0.7	0.7	0.4	0.4	*	3355	0.734
2	8	0.084	0.5	0.5	0.2	0.2	*	5710	0.573
2	8	0.044	0.3	0.3	0.1	0.1	*	3549	0.717
2	5	0.014	0.3	0.3	0.1	0.1	+	15,769	0.695
2	**5**	**0.044**	**0.5**	**0.5**	**0.2**	**0.2**	**+**	**16,503**	**0.775**
3	8	0.084	0.5	0.5	0.2	0.2	*	13,472	0.762
2	10	0.014	0.5	0.5	0.2	0.2	+	10,675	**0.320**

Bold entries represent optimal settings.

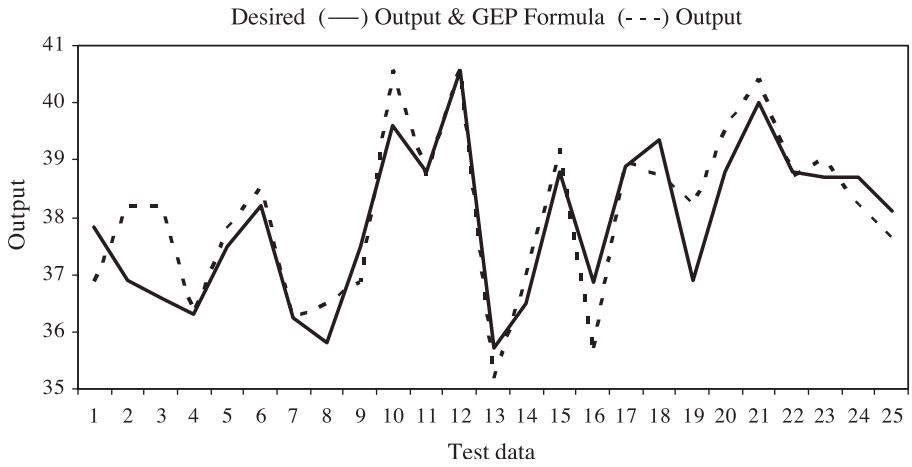

Fig. 4. Evaluation of the GEP method for the cement strength prediction.

the data, according to a transfer function, to all the elements in the next layer. However, each node receives a different signal due to different connection weights between the nodes. The output of each node in the output layer is compared to the desired output. The difference between the desired output and the calculated one (the error) is fed back to the network, so that the weights of the connections between the neurones are adjusted in a way that minimises this error. After repeating this procedure that is called "training" several times, the network is "trained" and can be used with input data not shown previously.

In training of a network, the network is forced to take on particular output form by means of weighting selection for a given input pattern. The biggest secret to building successful NNs is to know when to stop training. The choice of learning examples is very important. If there are not enough examples, the network realises a route learning, and then it is unable to answer to new inputs. On the other hand, if the set of examples is too big, the NN is overtrained and cannot converge to a solution.

For a class of multilayered NNs through its simplicity, back propagation learning algorithm made the NN approach very popular and resolved doubts regarding the viability of the NN approach once and for all. The back propagation procedure, the most frequently used learning procedure for NNs, is a generalization of the least mean square (LMS) procedure for feedforward, multilayered networks with "hidden layers". It uses a gradient descent procedure that changes the weights in its original and simplest form by an amount proportional to the partial derivative of the error function (ε) with respect to the given weight. In a system that has no hidden units, the computation of these derivatives is relatively easy, but for multilayered networks, it is a harder task. The central idea of back propagation is that these derivatives can be computed efficiently by starting with the output layer and working backwards through the layers.

A variety of NN architectures has been proposed and at least 50 different types are being explored in research for different applications [12,13]. Among those, back propagation networks have quickly become the most widely encountered NN, particularly within the area of systems and control [14].

An NN system can be developed for a certain application using any conventional programming tools, although it is very difficult to write in-house algorithms. However, some application platforms (shells) for NN development can be used. Some of the well-known NN development tools are NeuralSolutions and MATLAB Neural Network Tool Box.

4. Data collection

The data used in the GEP, NN, and regression analysis are collected from a cement plant located in Adıyaman, Turkey. The data collected are for a 4-month period (data for 104 days of production). The cement strength testing is performed according to European Standard EN 196-1 [15]. The type of cement used in this research was Cem II/B 32.5R European standard EN 197-1 [16]. The format of the data used in this research is explained in Table 1. There are 19 variables that are considered as the inputs to the GEP and NN learning algorithms. There is only one output (28-day CCS) that will be predicted. The data for the first 3 months

Table 4
Parameters of NN algorithms

1	Number of hidden layers	1, 2, 3
2	Number of neurones in hidden layers[a]	7, 10, 13
3	Type of NN	Multilayer perceptron
		Generalized feedforward
		Modular network
		Jordan/Elman
		Self-organizing map
		Principal component analysis
		Recurrent network

[a] The mean value of the number of neurones in hidden layers is estimated by 2 × square root (number of inputs + 1) [19]. Other values are obtained by adding and subtracting 3 from the mean value.

Table 5
Results obtained from the NN algorithms

Test no.	Number of hidden layers	Number of neurones in hidden layers	Type of NN	*R*-square error on test data
1	1	7	Multilayer perceptron	.318
2	1	10	Multilayer perceptron	.188
3	1	13	Multilayer perceptron	.141
4	2	7	Multilayer perceptron	.445
5	2	10	Multilayer perceptron	.296
6	2	13	Multilayer perceptron	.618
7	3	7	Multilayer perceptron	.483
8	3	10	Multilayer perceptron	.444
9	3	13	Multilayer perceptron	.432
10	1	7	Generalized feedforward	.629
11	1	10	Generalized feedforward	.415
12	1	13	Generalized feedforward	.296
13	2	7	Generalized feedforward	.647
14	2	10	Generalized feedforward	.535
15	2	13	Generalized feedforward	.515
16	3	7	Generalized feedforward	.248
17	3	10	Generalized feedforward	.382
18	3	13	Generalized feedforward	**.006**
19	1	7	Modular NN	.672
20	1	10	Modular NN	.684
21	1	13	Modular NN	.673
22	2	7	Modular NN	.652
23	2	10	Modular NN	.606
24	2	13	Modular NN	.460
25	3	7	Modular NN	.575
26	3	10	Modular NN	.680
27	3	13	Modular NN	.478
28	1	7	Jordan/Elman network	.224
29	1	10	Jordan/Elman network	.408
30	1	13	Jordan/Elman network	.596
31	2	7	Jordan/Elman network	.446
32	2	10	Jordan/Elman network	.597
33	2	13	Jordan/Elman network	.632
34	3	7	Jordan/Elman network	.586
35	3	10	Jordan/Elman network	.524
36	3	13	Jordan/Elman network	.352
37	1	7	Principal component network	.463
38	1	10	Principal component network	.100
39	1	13	Principal component network	.281
40	2	7	Principal component network	.376
41	2	10	Principal component network	.510
42	2	13	Principal component network	.268
43	3	7	Principal component network	.365
44	3	10	Principal component network	.371
45	3	13	Principal component network	.367
46	1	7	Self-organizing feature map network	.290
47	1	10	Self-organizing feature map network	.472
48	1	13	Self-organizing feature map network	.459
49	2	7	Self-organizing feature map network	.627
50	2	10	Self-organizing feature map network	.402
51	2	13	Self-organizing feature map network	.466
52	3	7	Self-organizing feature map network	.511
53	3	10	Self-organizing feature map network	.508
54	3	13	Self-organizing feature map network	.558
55	1	7	Recurrent network	.484
56	1	10	Recurrent network	.188
57	**1**	**13**	**Recurrent network**	**.695**
58	2	7	Recurrent network	.590
59	2	10	Recurrent network	.592
60	2	13	Recurrent network	.560
61	3	7	Recurrent network	.460
62	3	10	Recurrent network	.403
63	3	13	Recurrent network	.645

Bold entries represent optimal settings.

(data for 79 days of production) are used as the training data, and those for the 4th month (data for 25 days of production) are used as the test data for the GEP and NN algorithms. A similar approach is also followed during the regression analysis.

5. Model construction and analysis using GEP

The experimental data described in Table 1 is used for the modelling of the 28-day CCS of PCC. The major task is to define the hidden function connecting the input variables (d_1, d_2, d_3, …, d_{18}, d_{19}) and output variable (y). This can also be written in the form of the following equation: $y = f(d_1, d_2, d_3, \ldots, d_{18}, d_{19})$. The function obtained by the GEP algorithm will be used in predicting the 28-day CCS of the PCC. The parameters used in the GEP algorithm are presented in Table 2. The first three parameter sets are fixed in Table 3. There are 4374 different combinations of the GEP parameters, which mean 4374 different GEP models. Running the GEP algorithm for all of these combinations requires a huge amount of computational time. Therefore, a subset of these combinations is selected intuitively to investigate the performance of the GEP algorithm in predicting the CCS. In Table 3, the 10 best solutions obtained from these tests are presented. In the final row of Table 3, the worst solution obtained from the GEP analysis is given.

As can be seen from Table 3, the best result obtained from the GEP tests has a .775 *R*-square error. The function generated for the best result by the GEP algorithm to use in the CCS prediction of the PPC is given in Eq. (4).

$$y = d_5 - \log(\tan(d_{10}) + d_{18}) + \sqrt{d_{18}d_4 - \tan(d_{17})} \qquad (4)$$

The test result for the best function (Eq. (4)) is shown in Fig. 4. As can be seen from Fig. 4, there is a close match between the actual and predicted strength curves.

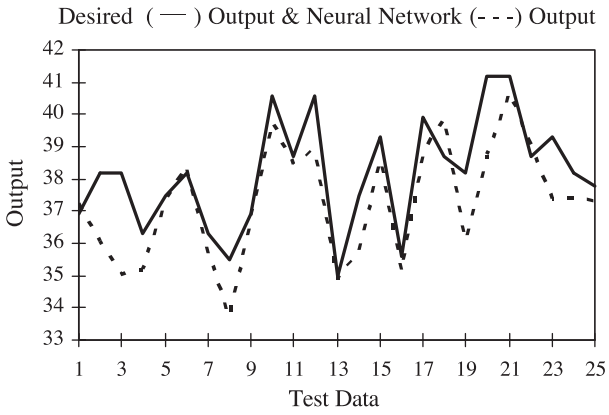

Fig. 5. Evaluation of the NN method for the cement strength prediction.

The GEP function is able to closely follow the trend of the actual data.

6. Model construction and analysis using NNs

In this research, the NN suite, which is known as NeuroSolutions, developed by NeuroDimension [17], is used. Seven different NN architectures are used for the CCS prediction of the PCC. These are multilayer perceptron, generalized feedforward, modular network, Jordan/Elman, self-organizing map, principal component analysis, and recurrent network. The experimental data described in Table 1 are also used for modelling the 28-day CCS of the PCC. The major task is to determine best possible weights (i.e., trained NN) of the NN for connecting the input variables (d_1, d_2, d_3, …, d_{18}, d_{19}) and output variable (y). The Delta–Bar–Delta algorithm [18] is used for training the NN. This algorithm is one of the best training algorithms for NNs [18]. The obtained trained NN is used in predicting the 28-day CCS of the PCC. The parameters used in the NN tests are presented in Table 4. There are 63 different combinations of the NN structures to test to have an idea of the performance of NNs in predicting the CCS. In Table 5, all of the solutions obtained from these tests are tabulated.

As can be seen from Table 5, the best result obtained from the NN algorithms has a .696 R-square error. The recurrent NN with single hidden layer and 13 neurons in the hidden layer produced the best result. The test result for this NN is shown in Fig. 5. However, the best result obtained from the NN is approximately 10% worse than the GEP algorithm's best solution shown in Table 3. Moreover, the worst solution obtained from the GEP analysis is .320 R-square error, whereas the worst solution obtained from NN analysis is found to be .006 R-square error. From this point of view, the GEP outperforms NNs based on the results of our computational experiments. Additionally, the GEP algorithm results in an equation that can be easily programmed even into a pocket calculator to use in future predictions. However, NN is a black box and does not say much about the interrelationships between the inputs and the output.

7. Regression analysis

A stepwise regression analysis is also performed to have an idea about the predictive power of the soft computing techniques, in comparison to a classical statistical approach. The experimental data described in Table 1 is used for modelling the 28-day CCS of the PCC. The major task is to determine the multivariable regression equation connecting the input variables (d_1, d_2, d_3, …, d_{18}, d_{19}) to the output variable (y). The obtained equation will be used in predicting 28-day CCS of the PCC. SPSS software package [20] is used to perform the stepwise regression analysis. The

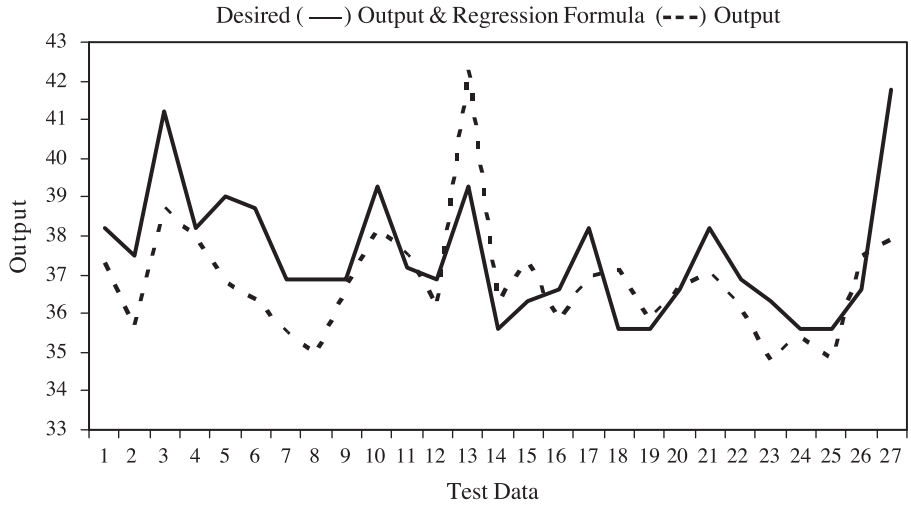

Fig. 6. Evaluation of the stepwise regression analysis for the cement strength prediction.

regression equation obtained from this analysis is given by Eq. (5).

$$\begin{aligned} y = {} & 129.2629 - 1.06461d_1 + 0.228794d_2 \\ & + 0.197611d_3 - 4.97767d_4 - 0.58978d_5 \\ & + 0.349407d_6 - 2.05767d_7 - 3.03088d_8 \\ & - 12.0376d_9 + 0.861342d_{10} + 0.00072d_{11} \\ & + 0.191208d_{12} - 0.19533d_{13} - 0.80748d_{14} \\ & - 0.00296d_{15} - 0.03243d_{16} + 0.039985d_{17} \\ & - 0.07229d_{18} + 0.602732d_{19} \end{aligned} \quad (5)$$

The R-square of the prediction by the regression equation on the test data is .357. The maximum and minimum error percentages for the regression equation on the test data are 9.38% and 0.15%. The test result for the stepwise regression analysis is shown in Fig. 6. As can be seen from the results, the stepwise regression analysis performed very much worse than the GEP and NN algorithms. This was an expected result, mainly because the CCS depends on many different chemical and physical factors, and the relations between these factors are highly nonlinear and complex [4]. The power of the soft computation techniques on making predictions on such complex problems is presented in different applications [11]. The results obtained from this study have also shown that the soft computing techniques are good candidates for making predictions on the behaviour of complex systems including the CCS prediction of the PCC.

8. Conclusions

In this study, two soft computing techniques, namely, GEP and NN, are applied to the problem of cement strength prediction. Moreover, the stepwise regression analysis is also applied to the problem to have an idea about the predicting power of these soft computing techniques in comparison to a classical statistical approach. The results obtained from our extensive computational tests have shown that soft computing techniques give far better results than the regression analysis. Moreover, GEP performed considerably better than NN algorithms in our tests. It is concluded that GEP is a good soft computing technique for use in cement strength prediction. Another important advantage of GEP is coming from its ability to generate mathematical equations that can be easily programmed even into pocket calculators for use in everyday practices during the cement production process.

References

[1] S. Tsivilis, G. Parissakis, A mathematical-model for the prediction of cement strength, Cem Concr Res 25 (1995) 9–14.
[2] C.E. de Siquera Tango, An extrapolation method for compressive strength prediction of hydraulic cement products, Cem Concr Res 28 (1998) 969–983.
[3] S. Akkurt, S. Ozdemir, G. Tayfur, B. Akyol, The use of GA-ANNs in the modelling of compressive strength of cement mortar, Cem Concr Res 33 (2003) 973–979.
[4] G. Fa-Liang, A new way of predicting cement strength—fuzzy logic, Cem Concr Res 27 (1997) 883–888.
[5] C. Ferreira, Gene expression programming: a new adaptive algorithm for solving problems, Complex Syst 13 (2) (2001) 87–129.
[6] J.R. Koza, Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press, USA, 1992.
[7] M. Gen, R. Cheng, Genetic Algorithms and Engineering Design, Wiley, USA, 1997.
[8] C. Zhou, W. Xiao, T.M. Tirpak, P.C. Nelson, Discovery of classification rules by using gene expression programming, In the Proceedings of the 2002 International Conference on Artificial Intelligence (IC-AI'02), CSREA Press, Las Vegas, 2002 (June), pp. 1355–1361.
[9] J. Zupan, J. Gasteiger, Neural Networks for Chemists, VCH, Weinheim, Germany, 1993.
[10] C.V. Altrock, Fuzzy Logic & NeuroFuzzy Applications Explained, Prentice-Hall, USA, 1995.
[11] T. Dereli, A. Baykasoğlu, The use of artificial intelligence techniques in design and manufacturing: a review, J Polytech 3 (2000) 27–60.
[12] D. Wang, Pattern recognition: neural networks in perspective, IEEE Expert, (1993 August) 52–60.
[13] R.H. Nielsen, Neurocomputing: picking the human brain, IEEE Spectrum 25 (3) (1988) 36–41.
[14] T. Kohonen, An introduction to neural computing, Neural Netw 1 (1988) 3–16.
[15] European Committee for Standardization (CEN), Methods of Testing Cement: Part 1. Determination of Strength, European Standard EN196-1.
[16] European Committee for Standardization (CEN), Cement: Part 1. Composition, specifications and conformity criteria for common cements. European Standard EN 197-1, June (2000).
[17] NeuroDimension, http://www.neurosolutions.com.
[18] Ç. Elmas, Yapay Sinir Ağları, Seçkin Yayıncılık, Ankara, ISBN: 975-347-612-4, 2003.
[19] NeuNet Pro 2.2, http://www.cormactech.com/neunet.
[20] M. Ergün, SPSS for Windows, Ocak Yayınları, Ankara, ISBN: 975-422-044-1, 1995.