Available online at www.sciencedirect.com
ScienceDirect
Applied Soft Computing 7 (2007) 387–397
Applied Soft Computing
www.elsevier.com/locate/asoc

A genetic algorithms based multi-objective neural net applied to noisy blast furnace data

F. Pettersson [a], N. Chakraborti [b,*], H. Saxén [a]

[a] Faculty of Chemical Engineering, Heat Engineering Laboratory, Åbo Akademy University, Biskopsgatan 8, FIN-20500 Åbo, Finland
[b] Department of Metallurgical & Materials Engineering, Indian Institute of Technology, Kharagpur 721302, West Bengal, India

Received 18 February 2005; received in revised form 30 August 2005; accepted 9 September 2005

Abstract

A genetic algorithms based multi-objective optimization technique was utilized in the training process of a feed forward neural network, using noisy data from an industrial iron blast furnace. The number of nodes in the hidden layer, the architecture of the lower part of the network, as well as the weights used in them were kept as variables, and a Pareto front was effectively constructed by minimizing the training error along with the network size. A predator–prey algorithm efficiently performed the optimization task and several important trends were observed.

Keywords: Genetic algorithms; Artificial neural net; Evolutionary computation; Evolutionary multi-objective optimization; predator–prey algorithm; Iron making; Blast furnace

1. Introduction

Integrated steel plants all over the world use iron-making blast furnaces shown schematically in Fig. 1. This is a highly complicated countercurrent heat and mass exchanger, where iron ore, coke and limestone are charged at the top, air is injected through the tuyeres located at its lower region, and molten pig iron and slag are periodically removed as separate output streams. A blast furnace normally runs round the clock for a very prolonged period, as the process of shutting down a blast furnace and restarting it is extremely cumbersome.

The process chemistry and the transport phenomena in blast furnaces are highly complex [1] and despite decades of intensive research, a fully reliable analytical model for the blast furnace is yet to emerge. Therefore, data-driven models are of vital importance for throwing light on the complex interrelations between variables in the process. Combines of genetic algorithms and neural nets have already been exploited to study various phenomena associated with the ferrous production processes including the blast furnace [2–5]. All these studies so far have utilized only the single objective genetic algorithms. The concept of multi-group classification exists in the

* Corresponding author.
E-mail address: nchakrab@iitkgp.ac.in (N. Chakraborti).

1568-4946/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2005.09.001

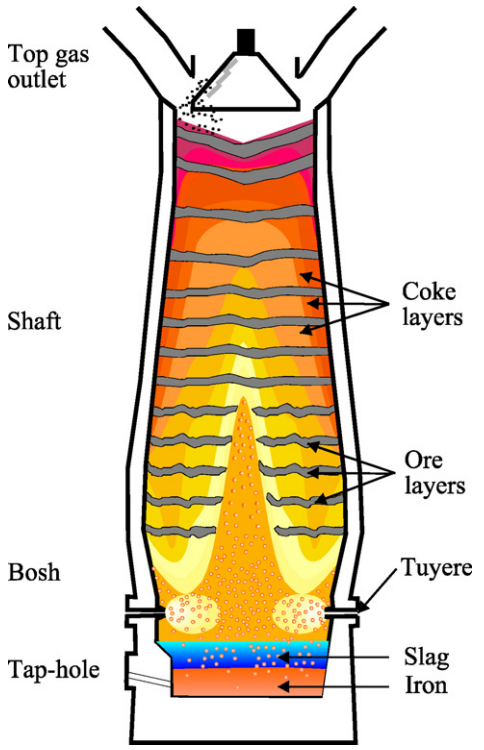

Fig. 1. Schematic diagram of an iron making blast furnace.

literature [6] for studies that tend to combine both genetic algorithms and neural nets. However, a direct application of the currently emerging evolutionary multi-objective optimization techniques [7–8] is still quite an unchartered territory in the neural network research. Using some noisy industrial blast furnace data we will demonstrate in this paper how that can be done in a meaningful fashion.

2. The problem formulation and solution strategy

The pig iron, that a blast furnace produces, contains several alloying elements dissolved in it. Among them C, S and Si are three crucial components which need to be controlled efficiently within a specified target. Several input parameters can significantly affect these compositions. In this study, we chose to select five of them, all related to the reductants, i.e., coke and the injected oil: the ash content, size, strength and alkali content of the coke as well as the specific oil injection rate. These five input variables and the concentration of the three alloying elements in the pig iron were assembled into a feed forward neural network schematically shown in Fig. 2. Actual blast furnace data for a period of 200 weeks were utilized for training this network. Since the absolute levels of the variables change with time, it was decided to base the analysis on differentiated series, i.e., changes (from one week to the next) in the input and output quantities. Denoting the change in a variable at week k by $\Delta\tilde{\Omega}(k) = \tilde{\Omega}(k) - \tilde{\Omega}(k-1)$, the quantities were normalized as:

$$\Delta\Omega(k) = \frac{\Delta\tilde{\Omega}(k) - \Delta\bar{\tilde{\Omega}}}{s_{\Delta\tilde{\Omega}}} \quad (1)$$

where $\Delta\bar{\tilde{\Omega}}$ denotes the mean value and s is the estimated standard deviation. After normalization the weeks with at least one value of $\Delta\Omega(k) \notin (-3, 3)$ were considered as outliers and were removed from the data set. In the present case, this implied a rejection of about 7% of the total observations. In order to be able to capture higher-order dynamics, several lagged values of the inputs were included in the analysis. For a detailed description of the data preprocessing steps the readers are referred to [9].

The learning process of the upper part of this neural net was carried out through a linear least square

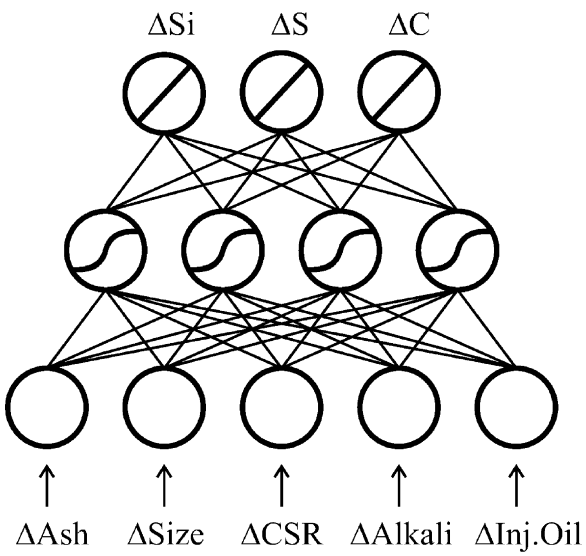

Fig. 2. Schematics of the feed forward neural net used in this study.

technique described elsewhere [2], while the lower part was affected by the genetic processes. A *population* of such networks was utilized for the multi-objective formulation, and further details are provided below.

2.1. The multi-objective treatment of the problem

A good network should be able to reproduce the data in an acceptable manner. At the same time, it should try to do so with a small number of connections, so that the trained network does not get over-trained. In order to achieve both of these objectives, here we had attempted to simultaneously minimize (i) the training error of the network (E) and (ii) the required number of active connections in the lower part of it (N). The architecture of the lower part of the network and their corresponding weights were treated as variables influencing the objective functions. These two objectives are however conflicting, in many instances; with smaller number of nodes the training error is expected to rise and vice versa. The trade off situation between them can therefore be represented as a *Pareto-front*, the basic features of which are described elsewhere [5,7–8].

In the framework of genetic algorithms, the Pareto-optimality in this study has been computed using a predator–prey algorithm, which is essentially an adapted version of a computing scheme suggested earlier by Li [10]. The main features of this algorithm are provided below.

2.2. The predator–prey algorithm

The *preys* in this algorithm are initiated as a family of randomly generated sparse neural networks, which constituted an initial *population* in the usual genetic algorithms sense. Any *individual* in the prey population differs from its other members both by the nature of the lower part connections and the weight values associated with it. The *predators, on* the other hand, are a family of externally induced entities whose sole purpose is to prune the prey populations based upon a *fitness* value (ϕ), which is related to both the objective functions in this particular case. In this study, the effective fitness of any prey i was taken as a weighted sum of E and N such that:

$$\Phi_{ij} = \omega_j E + (1 - \omega_j) N; \quad 0 \leq \omega_j \leq 1 \tag{2}$$

where ϕ_{ij} is the fitness of prey i as taken by predator j, and the weight value ω_j was attributed to the predator j using a uniform random number. A similar scheme was suggested earlier by Deb [7].

A two-dimensional lattice was constructed as a computational space and both the predators and the prey were randomly introduced there. Any predator or a prey would have its own neighborhood, as shown in Fig. 3. While constructing the neighborhood, the first and the last rows in the lattice were taken as contiguous. The same consideration was applied to the first and the last columns as well and the diagonally opposite corner points at the end of the lattice were also taken as neighbors. The basic idea is similar to what has been used earlier for cellular automata with a Moore's neighborhood [11]. However, unlike cellular automata, where the lattice denotes the discretized physical space, the lattice here is simply a mathematical construction to facilitate a smooth function of this algorithm.

The predator is allowed to move one step at a time, up to a maximum number of steps that is dynamically adjusted at the end of a generation. During this movement, it kills the weakest prey in its neighborhood. If only one prey exists there it gets killed by default. If more than one prey exists, the predator kills the one that performs worst in terms of its ϕ value and

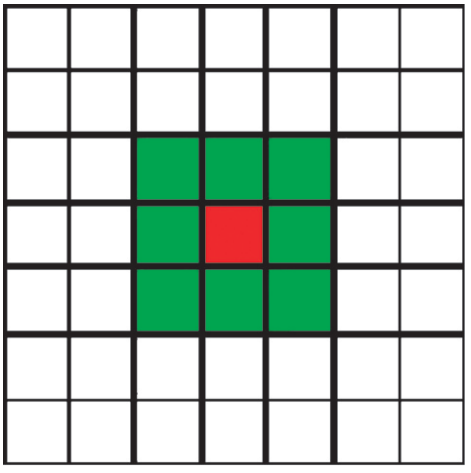

Fig. 3. A lattice with Moore's neighborhood shown for a predator or a prey.

takes its position. In case the neighborhood is empty besides the predator, it makes one random move in any direction, provided it has not exhausted its allowed number of moves and can continue to hunt. A target population of the preys (P) is set to begin with. At the onset of any generation, if the population contains a total of π preys along with Π predators, the allowed number of moves for a predator (υ) is updated as:

$$\upsilon = \left\lfloor \frac{P - \pi}{\Pi} \right\rfloor \tag{3}$$

In most situations Π will be larger than P.

This equation helps to put a check on drastic killing or an unwarranted growth of the prey population.

Once the killing stops, the survivors in the prey population are allowed to move randomly, one step at a time, up to a prescribed maximum, provided the movement does not result in landing on a location that is already occupied. It is also done with a probability so that every member of the prey population does not move. A new neighborhood is thus established where the prey population members are allowed to do both *crossover* and *mutation*. Each prey is allowed to choose a random mate in the new neighborhood, provided at least one more prey resides there. The crossover process that we have utilized here is shown schematically in Fig. 4. Essentially, it involved swapping the similarly positioned hidden nodes of two parents participating in the crossover. Each node is swapped with a predefined probability and in Fig. 4 only the second node has been swapped. During the exchange process, the recipient individual also inherited the connectivity and the weights of the incoming node. If the incoming node was not connected to any input streams to begin with, then an additional sparseness is introduced in the recipient gene. Mutation was done in a self-adjusting fashion, only to the real coded weight values, following the standardized procedures of *differential evolution* [12].[1] The mutated version of any weight belonging to a population member m, connecting an input i to a hidden node j was determined as:

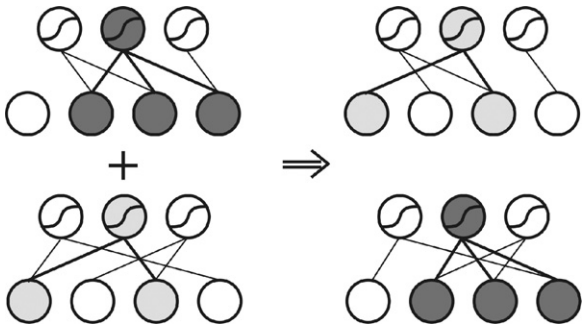

Fig. 4. The crossover scheme. The shaded regions are participating in the crossover process.

$$W_{ij}^{m\mu} = W_{ij}^{m} + \lambda(W_{ij}^{k} - W_{ij}^{l}) \tag{4}$$

where the superscript μ denotes the mutated version of the weight W_{ij}^{m}, the superscripts k and l refer to two randomly picked weights from the network population, both connecting an input i to hidden node j, while λ is a user defined mutation constant. To the readers familiar with differential evolution, it should be apparent that the second term in Eq. (4) renders the mutation linearly self adjusting [13], as it is expected to remain large in the initial population, but should increasingly become smaller as the population converges.

Two *children* are produced as a result of a crossover operation and the corresponding mutation. The children were randomly placed in any unoccupied location in the lattice. Only a prescribed number of attempts were allowed to locate one such location. Such *migration* of the children facilitates better mixing of the available gene pool, as the children are allowed to participate in the forthcoming crossover processes. In this algorithm the predators are not allowed to evolve by any means and their number remained constant from start to end.

The new generation started with the new progeny of the prey population: each of them an individual neural network of different architecture. They were now individually run with the blast furnace data in hand, and the objective functions E and N were determined for the entire population. This information was now passed on to the *predator–prey algorithm* and the genetic processing continued. The prey members which evolved this way were *ranked* at a certain

[1] Differential evolution applies such mutation on a vector of variables. Here, it is applied to an individual weight, which can be treated as a scalar.

interval following the Fonseca procedure [14] where the rank of any individual i ($\Re_i$) was taken as:

$$\Re_i = 1 + \Theta_i \quad (5)$$

where Θ_i denotes the number of individuals dominating i. To determine it the well known weak dominance condition [8] was used:

$$(\Omega_l \prec \Omega_m) \Leftrightarrow (\forall_i)(f_{il} \leq f_{im}) \wedge (\exists_m)(f_{il} < f_{im}) \quad (6)$$

The Ω terms denote vectors in the objective function space represented by the f terms.

The prey members worse than rank four were removed after each ranking operation. This was not done in the original version of this algorithm [10]. However, we found it to be highly effective for the problem in hand. A large number of superfluous solutions were pruned because of this action and the Pareto front maintained a very steady convergence. Once the algorithm had finally converged, the non-dominating set of rank value 1 was accepted as the final Pareto-solution.

3. Computational

A MATLAB^{TM} code was developed during this study which executed under a WINDOWS-XP environment. Before applying to the present problem the code was tested on several test functions used by Li [10] and the reproduction was excellent. We found the code quite stable for a wide range of parameters. After some careful numerical experiments the final runs were obtained with the parameter set described in Table 1.

A speedy convergence could be achieved with the above-mentioned parameters. The code was run for a maximum of 150 generations, although a good

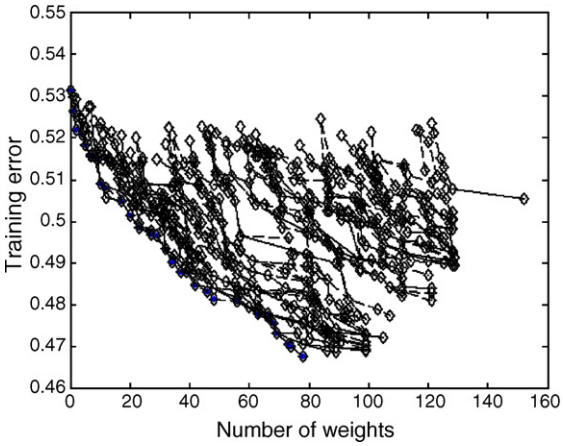

Fig. 5. Movement of rank 1 population in different generations. The final Pareto points are shown as partially filled diamonds.

Table 1
The parameters used in computation

Parameter	Value	Remarks
Lattice size	100 × 100	Smaller lattice sizes like 80 × 80 also gave acceptable results
No. of predators	30	Worked better than both 20 and 40 predator simulations
No. of prey (target)	200	Worked very well in the current lattice size
Probability of crossover	0.95	High crossover rate was beneficial
Probability of mutation	0.7	High mutation was beneficial
Mutation constant	0.16	Adjusted through numerical experiments
Probability of prey movement	0.3	Worked satisfactorily
Maximum steps for prey movement	10	Worked satisfactorily
Maximum steps for predator movement	Adjusted dynamically	Worked satisfactorily
No. of hidden layers	1	No more were needed
Maximum number of nodes in the hidden layer	10	It worked better than six nodes, as the system gets more degrees of freedom
Probability of omitting a node in the initial population	0.3–0.99	The entire range was utilized through a loop providing adequate population diversity
Upper bound for randomly generated weight	5	The nearly linear regime of the sigmoidal transfer function could be effectively used
Lower bound for randomly generated weights	−5	The nearly linear regime of the sigmoidal transfer function could be effectively used

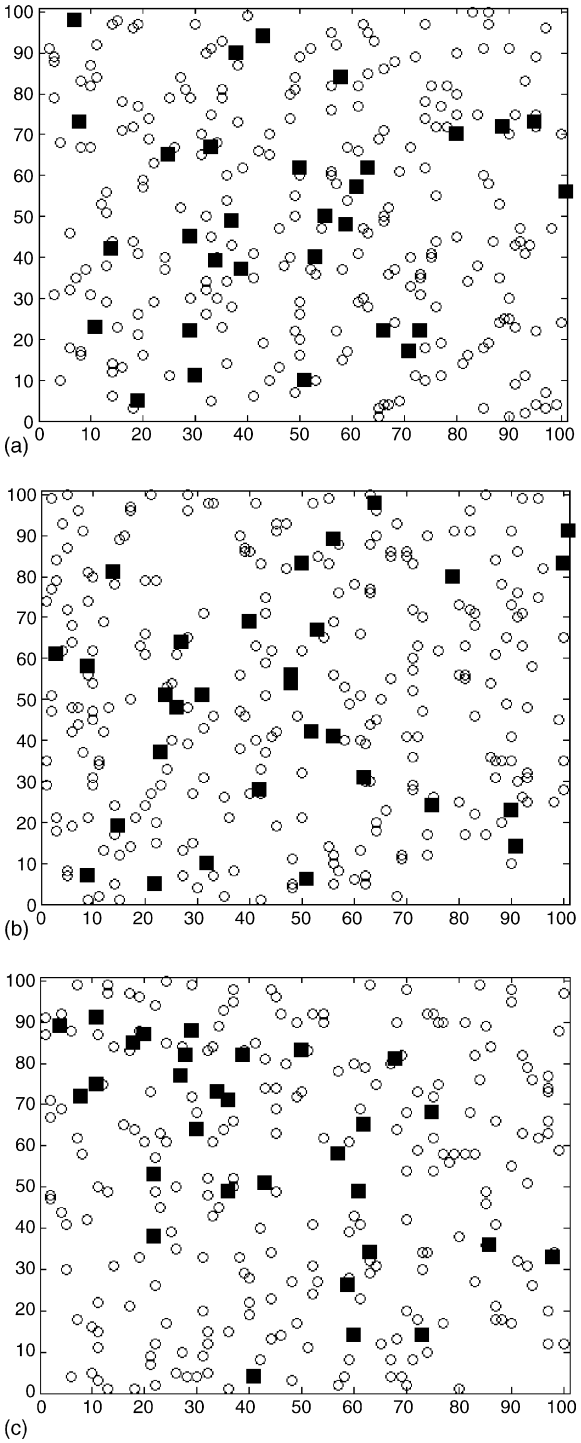

Fig. 6. (a) Predator (dark rectangles)–prey (open circles) distribution at generation 5, (b) same as in (a), at generation 50, (c) same as in (a), at generation 150.

convergence was possible to achieve early. It seemed an initial population biased around the *Nadir* point [7] could provide a more steady convergence towards the ultimate Pareto front, compared to a fully random initial population. A typical progress towards convergence is shown in Fig. 5, where the rank 1 solutions are plotted for several generations. The rank 1 solution obtained after the first generation appeared at the upper right corner of the diagram, and the front steadily moved towards the lower left corner since then.

The dynamics of predator–prey interaction is further elaborated in Fig. 6. The three diagrams presented there are basically snapshots of the lattice after the killings ended at a prescribed generation. It seems both the predators and the preys remained well dispersed in the lattice for the entire simulation, although towards the end, some predators started to cluster around in a region heavily inhibited by the preys. These are some desirable features, which facilitated a steady progress towards the optimum Pareto solution.

4. Results and discussion

The computed Pareto front is shown in Fig. 7. The entire data set for approximately 200 weeks at hand here was too small for setting aside a meaningful part of it for testing, without sacrificing data necessary of training. We have however demonstrated already [15] that the networks trained using the same procedure and some fragments of the total data set could be convincingly used for the entire period of 200 weeks. A testing for the efficacy of the genetic algorithms based training procedure adopted here thus has already been performed, and is not attempted, once again, in the present work. Furthermore, the approach made here can be seen as a truly multi-objective alternative to more traditional schemes, e.g., the Akaike information criterion (AIC) or the final prediction error (FPE) [16], for finding parsimonious models that estimate the expected approximation error on independent (‘‘unseen’’) data sets by adding a complexity-penalizing term to the training error. In the present approach, a good tradeoff between the two objectives has nicely worked out in this problem. For a further elaboration of the consequences of this trade

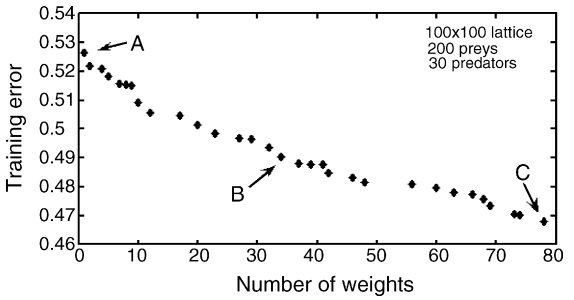

Fig. 7. The Pareto front computed with a target prey population of 200.

off we have selected three representative points in the diagrams: the point A which requires only a few connections but results in a high training error, point B which uses a moderate number of connections and also receives a moderate training error and finally, the point C which provides a very small training error but requires a very large number of connections. The neural nets that represented these points are shown in Fig. 8, the computed weight values are presented in Table 2 and their performances against the actual blast furnace data are presented in Fig. 9.

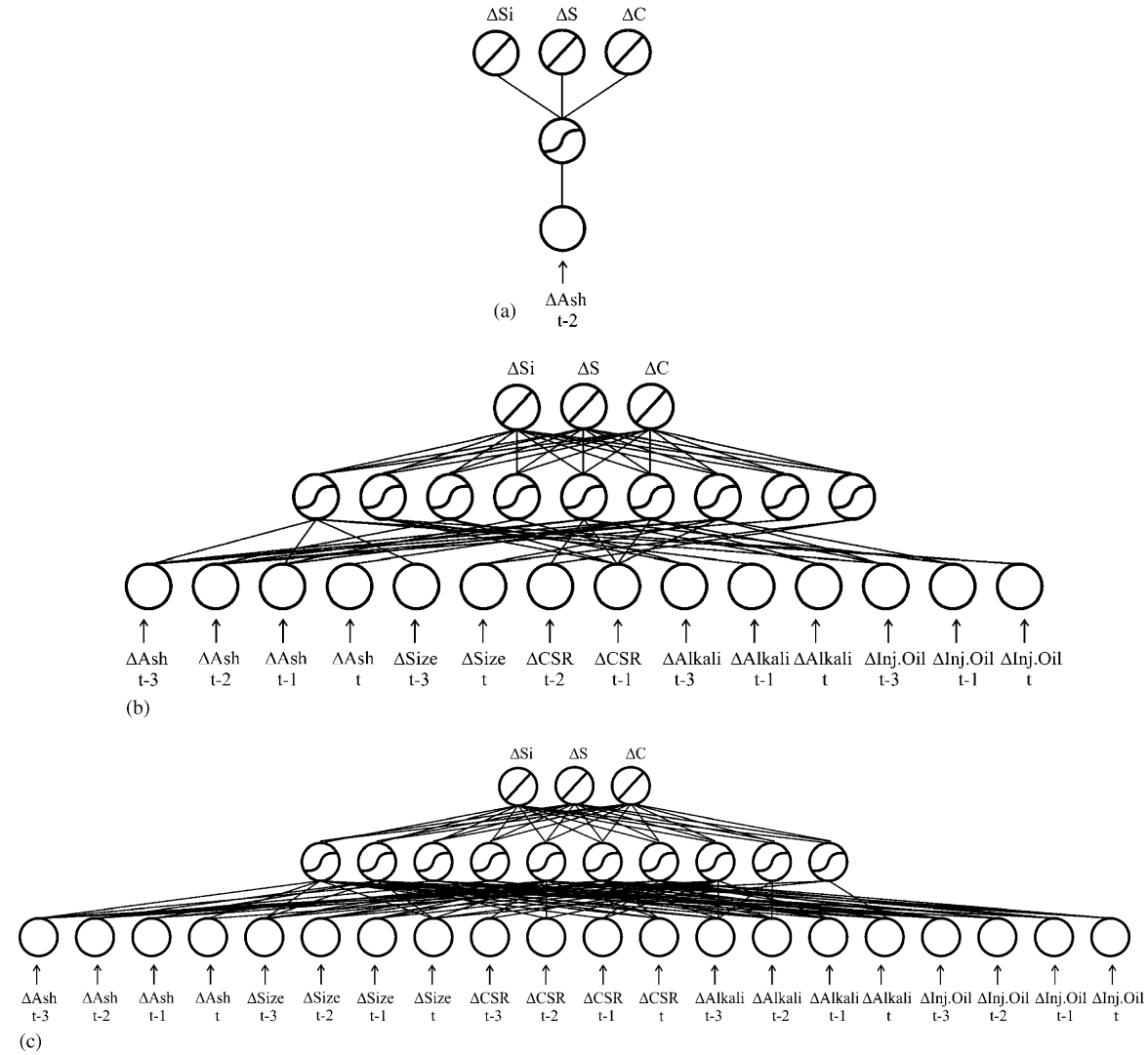

Fig. 8. (a) Network architecture corresponding to point A in (Fig. 7), (b) network architecture corresponding to point B in (Fig. 7) and (c) network architecture corresponding to point C in (Fig. 7).

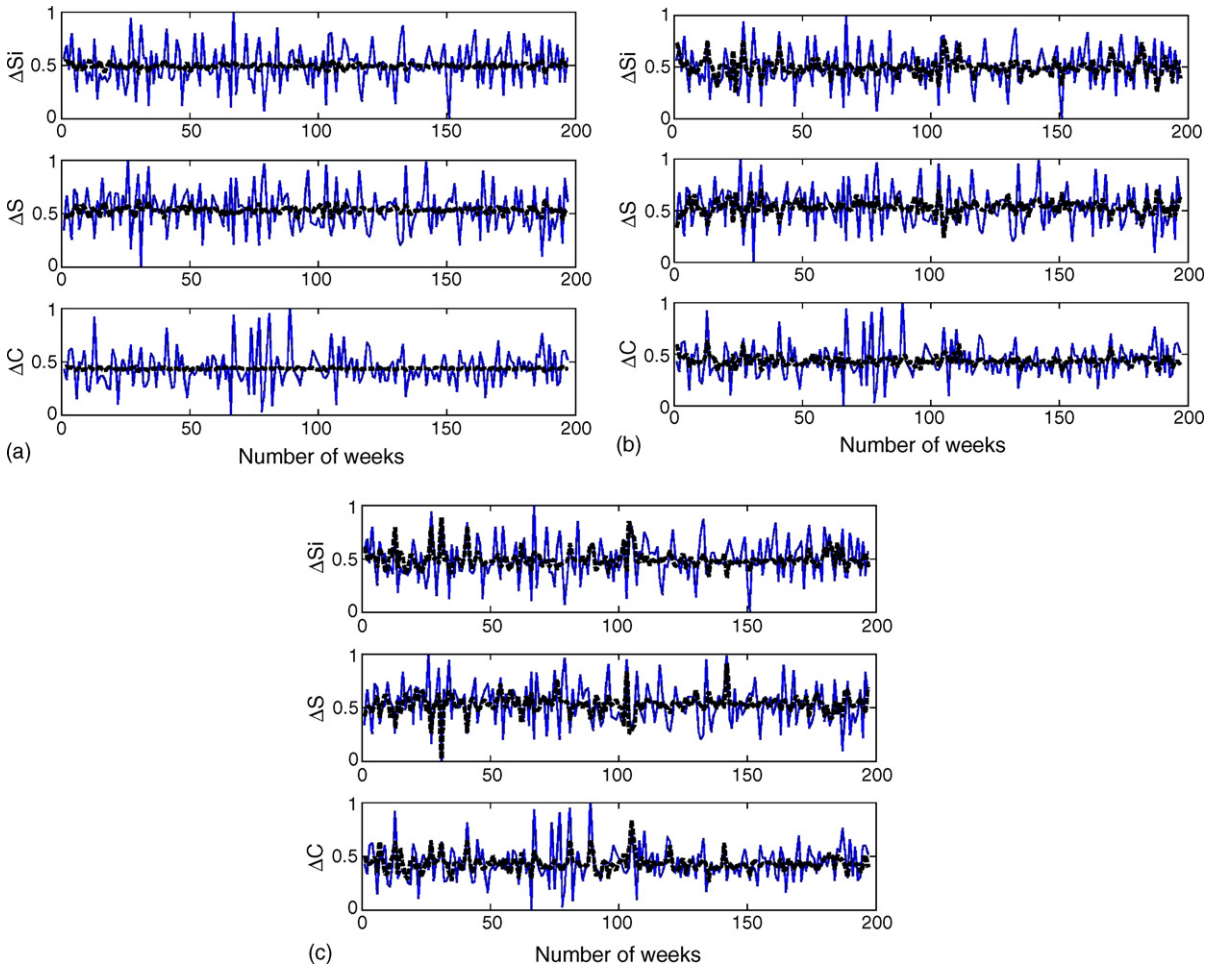

Fig. 9. (a) Network training corresponding to point A in (Fig. 7), (b) network training corresponding to point B in (Fig. 7) and (c) network training corresponding to point C in (Fig. 7). The model predictions are shown as darker lines in all the three figures.

It needs to be emphasized at this point that Fig. 7 is plot in the *functional space*, rather than in the *variable space*, where only the non-dominating points determined using the weak dominance criterion (Eq. (6)) are shown. Although not shown in this figure, very large networks containing even up to 150 weights, were observed in early stages of the evolution.

Very large networks, and for that matter, very low training errors are actually not the requirements in the present problem. As discussed in more detail in [15], the intent of the model is not to capture all of the variation in the hot metal chemistry, since it is extremely unlikely that all these changes would be driven by variation in the reductant properties and their input rates.

Analyzing the models on the Pareto front, it appears that among the input variables, the ash content in the coke is perhaps the one, which is most important. This is the only input that has been utilized in the smallest network shown in Fig. 8(a), and it appeared without any exceptions in every meaningful network that evolved in this study. In the order of importance the coke strength appeared to be the next most significant input followed by the rate of oil injection. These are some important information for the operation of the industrial blast furnace. A comparison between the Figs. 7–9 would readily reveal that a very sparse network, as in point A, would execute very fast but cannot really reproduce the data

Table 2
The weight values for the networks shown in Fig. 8

	Si	S	C
Network A: weights between hidden nodes and outputs			
Bias	−26.186	35.926	−8.496
Node 1	26.938	−35.742	9.015

	Bias	Ash content ($t-2$)
Network A: weights between input and hidden nodes		
Node 1	4.375	0.543

	Si	S	C
Network B: weights between hidden nodes and outputs			
Bias	−42.452	33.387	−48.087
Node 1	122.451	−92.445	37.219
Node 2	1.787	−0.470	1.257
Node 3	0.072	−0.221	0.141
Node 4	−0.371	−0.139	0.507
Node 5	0.459	−0.326	0.293
Node 6	−0.523	0.472	−0.418
Node 7	−0.695	0.706	−0.330
Node 8	42.150	−33.359	47.852
Node 9	−0.413	0.529	−0.194

	Bias	Ash content				Size		CSR		Alkali			Injected oil		
		$t-3$	$t-2$	$t-1$	t	$t-3$	t	$t-2$	$t-1$	$t-3$	$t-1$	t	$t-3$	$t-1$	t
Network B: weights between input and hidden nodes															
Node 1	−2.296	1.520	0.000	−4.124	0.000	−7.209	0.000	0.000	0.000	0.000	0.000	0.000	−3.981	0.000	−2.131
Node 2	4.618	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.754	0.000	−1.520	0.000	0.000	0.000	−0.959
Node 3	−4.501	0.000	1.990	0.000	0.000	0.000	0.000	0.000	4.504	1.732	0.000	0.000	0.000	0.000	0.000
Node 4	−0.957	0.000	0.000	−4.647	0.000	0.000	0.000	0.000	0.000	−1.635	0.000	0.000	0.000	0.000	0.000
Node 5	0.782	2.576	3.011	0.000	0.000	0.000	0.000	−0.682	−3.223	0.000	0.000	−4.240	0.000	4.906	0.000
Node 6	4.700	1.267	0.000	−1.267	−4.599	0.000	0.000	0.000	4.069	0.000	0.000	−2.048	0.211	0.000	0.000
Node 7	−2.337	4.583	−2.322	0.000	0.000	0.000	5.309	0.000	0.043	0.000	0.000	0.000	1.621	3.633	0.000
Node 8	6.868	0.000	−1.488	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Node 9	0.240	0.000	0.000	0.000	0.000	−0.046	−3.040	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

	Si	S	C
Network C: weights between hidden nodes and outputs			
Bias	0.856	0.860	−0.333
Node 1	12.927	−10737	7.978
Node 2	0.585	−1.122	0.288
Node 3	−0.787	−1.748	−2.430
Node 4	−1.083	1.048	0.358
Node 5	34823	−30475	24879
Node 6	7.945	−6.875	14.334
Node 7	−0.383	0.303	−0.436
Node 8	11.350	−14.093	5.708
Node 9	0.222	−0.363	0.259
Node 10	0.017	−0.005	−0.010

Table 2 (*Continued*)

	Bias	Ash content				Size				CSR				Alkali				Injected oil			
		$t-3$	$t-2$	$t-1$	t	$t-3$	$t-2$	$t-1$	t	$t-3$	$t-2$	$t-1$	t	$t-3$	$t-2$	$t-1$	t	$t-3$	$t-2$	$t-1$	t
Network C: weights between input and hidden nodes																					
Node 1	−8.317	−3.698	0.000	0.000	6.088	9.055	0.000	0.000	−0.507	0.000	0.000	−5.993	2.184	−0.188	−7.036	0.000	1.610	−2.591	−3.940	0.000	0.000
Node 2	3.857	0.000	1.164	5.538	0.000	−3.119	0.000	0.000	8.385	0.000	−0.990	0.000	1.161	0.549	0.000	−0.517	0.000	0.000	−1.625	−7.433	0.000
Node 3	0.419	0.510	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	−2.716	0.000	−5.111	−3.672	0.000	−3.465	0.000
Node 4	2.496	0.000	−2.494	0.000	0.000	3.435	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	−1.281	0.000	4.645	0.000	0.000	0.000	0.000
Node 5	−1.397	0.000	0.000	−9.009	0.000	−9.953	0.078	2.435	−3.191	2.515	−1.376	0.000	−0.585	0.000	0.000	−6.299	0.000	−6.601	−0.147	0.000	−3.201
Node 6	0.674	0.000	0.480	0.000	0.000	0.000	−3.859	−1.545	0.000	0.000	0.000	0.000	−2.927	−5.054	3.665	−1.950	−6.926	0.000	−0.189	−4.144	4.498
Node 7	−1.589	−2.251	−0.855	1.080	0.000	0.371	0.000	0.000	0.000	0.000	0.000	1.478	0.000	0.000	2.351	0.000	0.470	0.376	0.000	−3.087	0.203
Node 8	−4.070	−6.938	0.000	0.000	−1.598	0.000	−0.353	−2.593	3.492	4.299	−5.267	−4.344	0.000	1.428	0.000	−3.908	0.000	0.000	0.000	0.000	0.000
Node 9	−1.133	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.861	3.587	0.000	0.000	0.000	0.000	0.000	0.000
Node 10	2.895	−2.030	0.000	0.000	0.000	0.000	0.000	0.000	−6.736	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.470	0.000	0.000	0.000	0.000

faithfully, other than showing an average trend line. The very dense rework, as in point C, on the other hand would tend to reproduce every peak and valley in the noisy data set, adding in the process, some undesirable significance to certain attributes in the data that could be nothing but a manifestation of the noise. A wise strategy would be to come up with a right amount of neural connections [17], and the recent past at least two studies [18,19] have addressed to this problem by treating the number of nodes in the hidden layer as variables, and trying to optimize it through single objective genetic algorithms. The obvious pitfall of a single objective analysis is, however, that it provides a unique solution that often becomes too rigid to implement in an industrial scenario. For the blast furnace operators, the methodology proposed here can not only provide a long term prediction capability, as the data are being acceptably trained for a period of 200 weeks, but can also provide the flexibility of choosing a workable compromise between the accuracy of fit and size of the network, as per their own requirements. In the complex realm of a steel plant, the specific requirements might even change from day to day, and the alternates provided by a Pareto front like Fig. 7 could be very effectively utilized in such an environment. It is also interesting to note that the work could be further extended by developing a scheme for updating the weights of a set of possible models (from the Pareto front) that can be applied in different operation regimes to produce optimal forecasts of future changes in the hot metal composition. Finally, it should be pointed out that the findings of the work has provided insight into the complex relations between raw material and product quality variables of the process, which are of utmost practical importance.

5. Concluding remarks

Although recently various other multi-objective evolutionary algorithms have been tried out for solving problems of chemical or materials interest [20–24], this is perhaps the first time that a predator–prey algorithm has been successfully applied to a sufficiently complex problem in this domain. The method performed in a highly robust manner, and is therefore deemed well suited for tackling other similar

modeling problems in the future. The rather novel method of a multi-objective analysis that has been demonstrated here for the noisy data of an iron making blast furnace is not only to benefit the steel producers, but would perhaps be of interest to soft computing researchers at large. Neural networks, for example, have been widely used for classification research, where a tradeoff between learning and generalization is known to occur [25]. Such problems possibly can be quite efficiently addressed through a Pareto-optimality approach, where the genetic algorithms can contribute in a big way. Genetic algorithms have already demonstrated their capabilities of extracting effective rule bases from trained neural network without changing the training results [26], and a multi-objective optimization strategy added to it can perhaps further rejuvenate any such worthwhile efforts, and a significant amount of further research is deemed justified to achieve that.

References

[1] Y. Omori (Ed.), Blast Furnace Phenomena and Modelling, The Iron and Steel Institute of Japan, Elsevier, London, 1987.

[2] F. Pettersson, J. Hinnelä, H. Saxén, A genetic algorithm evolving charging programs in the ironmaking blast furnace, Mater. Manuf. Process. 20 (2005) 351.

[3] F. Pettersson, J. Hinnelä, H. Saxén, Evolutionary neural network modeling of blast furnace burden distribution, Mater. Manuf. Process. 18 (2003) 385.

[4] B. Deo, A. Datta, B. Kukreja, R. Rastogi, K. Deb, Optimization of back-propagation algorithm and gas-assisted ANN models for hot metal desulfurization, Steel Res. 65 (1994) 528.

[5] N. Chakraborti, Genetic algorithms in materials design and processing, Int. Mater. Rev. 49 (2004) 246.

[6] R. Östermark, A hybrid genetic fuzzy neural network algorithm designed for classification problems involving several groups, Fuzzy Sets Syst. 114 (2000) 311.

[7] K. Deb, Multi-objective Optimization using Evolutionary Algorithms, John-Wiley, Chicheter, 2001.

[8] C.A. Coello Coello, D.A. Van Veldhuizen, G.B. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers, New York, 2002.

[9] M. Helle, H. Saxén, A method for detecting cause–effects in data from complex processes, in: B. Ribeiro, et al. (Eds.), Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms, SpringerComputerScience, Wien, 2005, p. 104.

[10] X. Li, A real-coded predator–prey genetic algorithm for multiobjective optimization, in: C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, L. Thiele (Eds.), Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, vol. 2632, LNCS, 2003, p. 207.

[11] N. Chakraborti, R. Dewri, Simulating recrystallization through cellular automata and genetic algorithms, Model. Simul. Mater. Sci. Eng. 13 (2005) 173.

[12] K. Price, R. Storn, Differential evolution, Dr. Dobbs J. 22 (4) (1997) 18.

[13] N. Chakraborti, A. Kumar, The optimal scheduling of a reversing strip mill: studies using multipopulation genetic algorithms and differential evolution, Mater. Manuf. Process. 18 (2003) 433.

[14] C.M. Fonseca, Multiobjective genetic algorithms with applications to control engineering problems, Ph.D. Thesis, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, UK, 1995.

[15] M. Helle, F. Pettersson, N. Chakraborti, H. Saxén, Modeling noisy blast furnace data using genetic algorithms and neural networks Steel Res. Int. (2006), in press.

[16] L. Ljung, System Identification, in: Theory for the User, second ed., PTR Prentice Hall, Upper Saddle River, NJ, 1999

[17] R. Reed, Pruning algorithms—a survey, IEEE Trans. Neural Netw. 4 (1993) 740.

[18] F. Pettersson, H. Saxén, A hybrid algorithm for weight and connectivity optimization in feedforward neural networks, in artificial neural nets and genetic algorithms, in: D.W. Pearson, N.C. Steele, R.F. Albrecht (Eds.), Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms Roanne, France, Springer-Verlag Vienna, Austria, April 23–25, 2003.

[19] H. Maaranen, K. Miettinen, M.M. Mäkelä, Training multi layer perceptron network using a genetic algorithm as a global optimizer in applied optimization, in: M.G.C. Resende, J.P. De Sousa, A. Viana (Eds.), Metaheuristics: Computer Decision-Making (Applied Optimization), Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[20] B.J. Reardon, Optimizing the hot isostatic pressing process, Mater. Manuf. Process. 18 (2003) 493.

[21] I.N. Egorov-Yegorov, G.S. Dulikravich, Chemical composition design of superalloys for maximum stress, temperature and time-to-rupture using self-adapting response surface optimization, Mater. Manuf. Process. 20 (2005) 569.

[22] N. Chakraborti, P. Mishra, A. Aggarwal, A. Banerjee, S.S. Mukherjee, The Williams and Otto chemical plant re-evaluated using a Pareto optimal formulation aided by genetic algorithms, Appl. Soft Comp. 6 (2006) 189–197.

[23] N.K. Nath, K. Mitra, Mathematical modeling and optimization of two-layer sintering process for sinter quality and fuel efficiency using genetic algorithm, Mater. Manuf. Process. 20 (2005) 335.

[24] A. Kumar, D. Sahoo, S. Chakraborty, N. Chakraborti, Gas injection in steelmaking vessels: coupling a fluid dynamic analysis with a genetic algorithms based Pareto-optimality, Mater. Manuf. Process. 20 (2005) 363.

[25] G.Q.P. Zhang, Neural networks for classification: A survey, IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. 30 (2000) 451.

[26] A.E. Elalfi, R. Haque, M.E. Elalami, Extracting rules from trained neural network using GA for managing E-business, Appl. Soft Comp. 4 (2004) 65.