Variance sensitivity analysis of parameters for pruning of a multilayer perceptron: application to a sawmill supply chain simulation model

Philippe Thomas, Marie-Christine Suhner, André Thomas

Variance Sensitivity Analysis of Parameters for Pruning of a Multilayer Perceptron: Application to a Sawmill Supply Chain Simulation Model

Philippe Thomas, Marie-Christine Suhner and André Thomas

Centre de Recherche en Automatique de Nancy (CRAN-UMR 7039), Nancy-University, CNRS
philippe.thomas@cran.uhp-nancy.fr

Simulation is a useful tool for the evaluation of a Master Production/Distribution Schedule (MPS). The goal of this paper is to propose a new approach to designing a simulation model by reducing its complexity. According to the theory of constraints, a reduced model is built using bottlenecks and a neural network exclusively. This paper focuses on one step of the network model design: determining the structure of the network. This task may be performed by using the constructive or pruning approaches. The main contribution of this paper is twofold; it first proposes a new pruning algorithm based on an analysis of the variance of the sensitivity of all parameters of the network, and then uses this algorithm to reduce the simulation model of a sawmill supply chain.

In the first step, the proposed pruning algorithm is tested with two simulation examples and compared with three classical pruning algorithms from the literature. In the second step, these four algorithms are used to determine the optimal structure of the network used for the complexity-reduction design procedure of the simulation model of a sawmill supply chain.

***Index Terms*— multilayer perceptron, pruning, reduced model, simulation.**

I. Introduction

Simulation is a useful tool for the evaluation of planning or scheduling scenarios [1]. Indeed, simulation highlights the evolution of the machine states, WIP (work in process) and queues. This information is useful for "predictive scheduling" [1] or rescheduling. Considering the theory of constraints [2], the optimization of production processes requires maximizing the utilization rate of the bottlenecks. This is the main indicator for evaluating a Master Production/Distribution Schedule (MPS). For this, a useful technique is simulating dynamic discrete events of the material flow [3].

In fact, simulation models of actual industrial cases are often very complex and modelers encounter problems of scale [4]. In addition, many works use the simplest (reduced/aggregated) models of simulation [5]–[8].

Neural networks can extract performing models from experimental data [9]. Consequently, the use of neural networks has been proposed in order to reduce simulation models [8][10]. To build a neural model, an important issue is determining the structure of the network. The main techniques used to control the complexity of the network are architecture selection, regularization [11] [12], early stopping [13] and training with noise [14]; the last three are closely related [14][15]. This paper focuses on architecture selection. To determine the optimal structure of the network, two approaches can be used. The first is constructive, where the hidden neurons are added one after another [16]–[19]. The second approach exploits a structure with too many hidden neurons, and then prunes the least significant ones [20]–[25].

In addition, it is necessary to determine the optimal input data set in order to make a model. This set of data must be as small as possible in order to avoid the overfitting problem, but must contain all the explicative inputs [26]. Different approaches can be used to perform feature selection, for example principal component analysis [27], curves component analysis [28] and random features ranking [29][30]. Other methods have been designed to perform feature selection only with neural networks [31]–[34]. Only some of these allow simultaneous feature selection and spurious parameter pruning [20][35][36].

The goals of this paper are dual. The first one is to present a reduction approach of simulation model using neural network. The second one is to deal with the optimal neural network structure determination by using pruning procedure.

This paper presents a new pruning algorithm that allows the selection of the input neurons and the number of hidden neurons. This algorithm, based on one proposed by Engelbrecht [36], is investigated and compared with three existing algorithms: Engelbrecht [36], Setiono and Leow [35] and Hassibi and Stork [20]. These algorithms are used for the structural determination of the neural network used in the reduced model of a sawmill flow shop.

In the next section, the topics of model reduction and multilayer perceptron are presented. The third section presents the pruning algorithms, including the proposed algorithm and the three comparison algorithms. Two simulation examples are then presented and the results obtained with the four algorithms are investigated. Section V presents the industrial application. The reduced model of the sawmill supply chain and the structure of the neural networks obtained using the different algorithms are investigated in section VI. The final section enumerates our conclusions.

II. THE MODEL REDUCTION

A. The algorithm

Zeigler [37] was the first to deal with the problem of model reduction when he stated that the complexity of a model is related to the number of elements, connections and model calculations. He distinguished three methods to simplify a discrete simulation model: replacing part of the model with a random variable, degrading the range of values taken by a variable, and grouping parts of a model together.

Innis *et al.* [38] listed 17 simplification techniques for general modeling. Their approach comprises four steps: hypothesizing (identifying the important parts of the system), formulation (specifying the model), coding (building the model) and experimentation. Leachman [39] proposed a model that considers cycle times in production planning, especially for the semiconductor industry, which uses cycle time as an indicator. Brooks and Tobias [6] suggest a "simplification of models" approach for cases where the indicators to be followed are the average throughput rates. Other cases have been studied in [40] and [41].

The reduction algorithm proposed in this paper is an extension of those presented by Thomas and Charpentier [3]. It is presented in Figure 1 and its principal steps are summarized as follows.

1. Identify the structural bottleneck (the work center (WC) that has been constrained in capacity for several years).
2. Identify the conjunctural bottleneck for the bundle of manufacturing orders (MOs) of the MPS under consideration.
3. Among the WCs not listed in 1 and 2, identify one (the synchronization WC) that satisfied two conditions:
   - it is present in at least one of the MOs that has a bottleneck;
   - it is widely used in the MOs.
4. If all MOs have been considered, go to step 5; otherwise repeat step 3.
5. Use neural networks to model the intervals between the WCs found during the previous steps.

As example, in the considered application (part VI), there are three WC: "Canter line" (figure 3), "Kockums line" (figure 5) and "Trimmer line" (figure 6). Only one of these WC is a bottleneck, in our case, this is the "Trimmer line".

The WCs remaining in the model are conjunctural bottlenecks, structural bottlenecks, or WCs that are vital to the synchronization of the MO. All other WCs are incorporated into "aggregated blocks" upstream or downstream of the bottlenecks.

A "conjunctural bottleneck" is a WC that is saturated for the particular MPS and predictive schedule in question, and therefore uses all available capacity. A "structural bottleneck" is a WC that has often been in such a condition in the past. Actually, for one specific portfolio (one specific MPS) there is only one bottleneck—the most loaded WC—but this WC can be different from the traditional bottlenecks.

"Synchronization work centers" are resources used jointly with bottlenecks for at least one MO and for the planning of different MOs that do not have a bottleneck. The number of these "synchronization work centers" must be minimized. To achieve this, WCs must be found that (a) are most common among the bundle of MOs using no bottleneck, and (b) figure in the routing of at least one MO using bottlenecks.

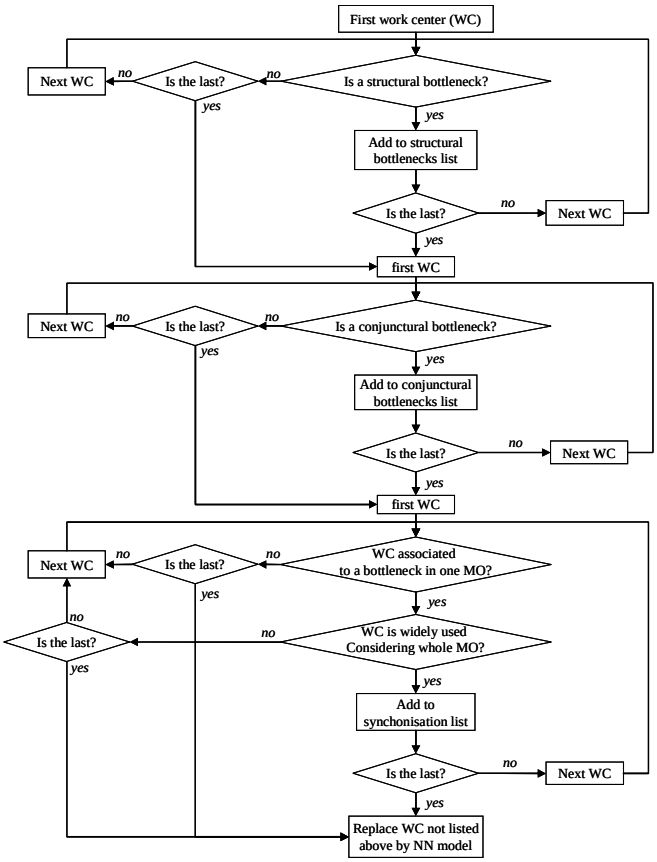

Figure 1. Algorithm used

B. The multilayer perceptron

The work of Cybenko [42] and Funahashi [43] has proved that a multilayer neural network with only one hidden layer (using a sigmoidal activation function) and an output layer (using a linear activation function) can approximate all nonlinear functions with any desired accuracy. These results explain the great interest in this type of neural network, which is called a multilayer perceptron (MLP). In this research work, it is assumed that a part of the modeled production system can be approximated with a nonlinear function obtained from an MLP.

The structure of the MLP is discussed here. Its architecture is shown in Figure 2. The neurons of the first (or input) layer distribute the n_0 inputs $\left\{x_1^0,\cdots,x_{n_0}^0\right\}$ of the MLP to the neurons of the next layer (hidden layer). A special input neuron (depicted by a square in Figure 2) represents a constant input (equal to one) that is used for the representation of the biases or thresholds of the hidden layer.

The i^{th} neuron (i = 1 … n1) in the hidden layer receives the n_0 inputs $\left\{x_1^0,\cdots,x_{n_0}^0\right\}$ from the input layer with associated weights $\left\{w_{i1}^1,\cdots,w_{in_0}^1\right\}$. This neuron first computes the weighted sum of the n_0 inputs:

$$z_i^1 = \sum_{h=1}^{n_0} w_{ih}^1 . x_h^0 + b_i^1 , \quad (1)$$

where b_i^1 is the bias or threshold term of the i^{th} hidden neuron. The output of this neuron is given by a so-called activation function of the sum in (1):

$$x_i^1 = g\left(z_i^1\right), \quad (2)$$

where g(.) is chosen as a hyperbolic tangent:

$$g(x) = \frac{2}{1+e^{-2x}} - 1 = \frac{1-e^{-2x}}{1+e^{-2x}} . \quad (3)$$

Lastly, the outputs of the hidden neurons $\left\{x_1^1,\cdots,x_{n_1}^1\right\}$ are distributed with associated weights $\left\{w_1^2,\cdots,w_{n_1}^2\right\}$ to the unique neuron of the last (or output) layer.

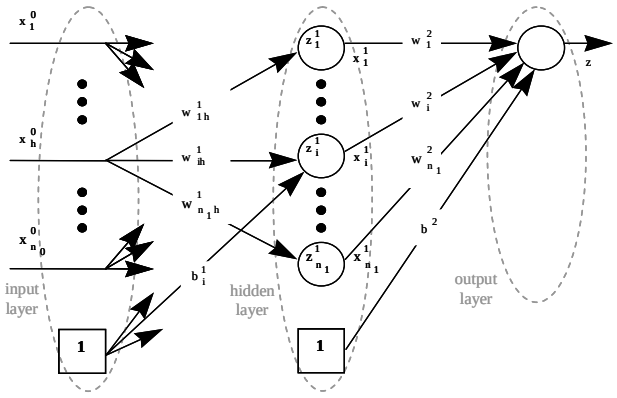

Figure 2. Architecture of the multilayer perceptron

As in the input layer, a particular hidden neuron (depicted by a square in Figure 2) represents a constant input equal to one that is used for the representation of the bias or threshold of the output layer.

The neuron of the last layer simply performs the following sum, its activation function being chosen to be linear:

$$z = \sum_{i=1}^{n_1} w_i^2 . x_i^1 + b , \quad (4)$$

where w_i^2 are the weights connecting the outputs of the hidden neurons to the output neuron, and b is the threshold of the output neuron.

Thus, only the number of hidden neurons is always unknown. To find this, a pruning algorithm may be used.

III. Pruning Algorithm

Pruning algorithms have been classified into two broad groups [44][45]: during learning pruning and post-learning pruning.

The "during learning pruning" methods add terms to the objective function that reward the network for choosing an efficient solution. These methods are also known as "weight decay" methods.

The "post-learning pruning" methods estimate the sensitivity of the error function to the removal of an element. The element with the least effect is then removed. Pruning continues until the effect of every element is deemed significant.

In this paper, only "post-learning pruning" methods are considered. This group can be divided into two subgroups [15]: weight saliencies pruning and output sensitivity analysis pruning.

Weight saliencies pruning considers the change in the error function due to small changes in the values of the weights. A measure of the relative effect of the different weights, or saliency, can be computed. The weights with low saliencies are deleted. The Optimal Brain Damage (OBD) [46] and Optimal Brain Surgeon (OBS) [20] algorithms, and all algorithms derived from them [22][47][48], use a second-order Taylor expansion of the error function to estimate how the training error will change as the weights are perturbed. Tang *et al.* [49] have proposed another method based on the use of an improved Extended Kalman Filter. In this approach, they use the error covariance matrix obtained during the learning in a similar way to the Hessian matrix in the OBS algorithm. Another approach is to use an approximation of the Fisher information matrix to determine the optimal number of hidden neurons [21].

The output sensitivity analysis method is based on a variance analysis of sensitivity information, given by the derivative of the neural network output with respect to the parameters [36]. It is a powerful method because the neural network structure inherently contains all the information to compute these derivatives efficiently [50]. Sabo and Yu [51] have proposed restricting the comparison of one parameter to those from the same hidden neuron. Zeng and Yeung [24] insert an input perturbation and study its effect on the output sensitivity. Chandrasekaran *et al.* [52] propose a sensitivity-based method utilizing linear unit models. These methods can be grouped with the so-called "local methods" of Sensitivity Analysis of Model Output (SAMO) approaches [53]. There exists a second Sensitivity Analysis (SA) algorithm, the global SAMO. In this approach, the space of the parameters (also

called factors or input factors in the SA terminology) is explored within a finite region, and the variation of the output induced by a factor is measured globally. Within this framework, Lauret *et al*. [15] proposed using the extended Fourier amplitude test to quantify the relevance of the hidden neurons.

Other algorithms use different approaches. Some methods, such as Neural Network Pruning for Function Approximation ("N2PFA"), attempt to remove units directly [35]. Some authors have proposed using genetic algorithms for pruning [45][54]. Liang [25] proposed using an orthogonal projection to determine the importance of hidden neurons.

In the following sections, the algorithm proposed by Engelbrecht will be discussed. Then, the modification of this algorithm will be presented and, finally, two other algorithms (OBS and "N2PFA") used for the comparison will be summarized.

A. The Engelbrecht algorithm

This algorithm uses the variance nullity measure (VNM) [55][56], where the variance of the sensitivy of an input or an output of a hidden neuron is measured for the different patterns. If this variance is not significantly different from zero, and if the average sensitivity is small, the input or the hidden neuron under consideration has no effect on the output of the network. Therefore, the VNM can be used in hypothesis testing to determine if an input or a hidden neuron has a statistical impact on the network using the χ^2 distribution. If not, it must be pruned.

To determine if a hidden neuron i must be pruned, the VNM of the weight w_i^2 ($i=1\ldots n_1$) that connects this hidden neuron to the output neuron must be calculated. For this, knowledge of the sensitivity of the network output z to the parameter w_i^2 is necessary. This sensitivity corresponds to the contribution of this parameter to the error at the output of the network. This contribution is determined by the partial derivative of the network output z with respect to the parameter w_i^2 being considered:

$$S_{w_i^2}(p) = \frac{\partial z(p)}{\partial w_i^2} = x_i^1(p) \qquad p = 1 \cdots P, \tag{5}$$

where P is the number of data patterns from the learning database.

Similarly, the sensitivity of the network output z to the input x_h^0 ($h=1\ldots n_0$) is obtained by performing the partial derivative of the output with respect to the input x_h^0 under consideration:

$$\begin{aligned} S_{x_h^0}(p) &= \frac{\partial z(p)}{\partial x_h^0(p)} = \sum_{i=1}^{n_1} \frac{\partial z(p)}{\partial x_i^1(p)} . \frac{\partial x_i^1(p)}{\partial z_i^1(p)} . \frac{\partial z_i^1(p)}{\partial x_h^0(p)} \\ &= \sum_{i=1}^{n_1} w_i^2 . g'\left(z_i^1(p)\right) . w_{ih}^1 \\ &= \sum_{i=1}^{n_1} w_i^2 . \left(1 - \left(x_i^1(p)\right)^2\right) . w_{ih}^1 \qquad p = 1...P \end{aligned} \tag{6}$$

The sensitivity of the network output to a hidden neuron or to an input can be explained with a unified notation using $S_{\theta_k}(p)$ ($p=1\ldots P$ and $k=1\ldots K=n_0+n_1$), with θ_k corresponding to x_h^0 if the input h is considered, or corresponding to w_i^2 if the hidden neuron i is considered. The notation $S_{\theta_k}(p)$ is given by equation (5) or (6) according to the considered case.

The VNM is the unknown variance $\sigma_{\theta_k}^2$ of the parameter θ_k. An estimator of this variance can be given by:

$$\hat{\sigma}_{\theta_k}^2 = \frac{\sum_{p=1}^{P} \left(S_{\theta_k}(p) - \overline{S_{\theta_k}}\right)^2}{P-1}, \tag{7}$$

where $\overline{S_{\theta_k}}$ is the mean of the sensitivity of the output to θ_k:

$$\overline{S_{\theta_k}} = \frac{\sum_{p=1}^{P} S_{\theta_k}(p)}{P}. \tag{8}$$

Using the VNM, the null hypothesis (that the variance in parameter sensitivity is approximately zero) is tested, where the null hypothesis $\mathcal{H}_0$ and its alternative $\mathcal{H}_1$ are:

$$\begin{cases} \mathcal{H}_0 : & \sigma_{\theta_k}^2 = \sigma_0^2 \\ \mathcal{H}_1 : & \sigma_{\theta_k}^2 < \sigma_0^2 \end{cases} \tag{9}$$

and where σ_0^2 is a small positive real.

Using the fact that, under the null hypothesis, the relation

$$\Gamma_{\theta_k} = \frac{(P-1).\hat{\sigma}_{\theta_k}^2}{\sigma_0^2} \tag{10}$$

has a $\chi^2(v)$ distribution with $v = P - 1$ degrees of freedom in the case of P patterns, the test (9) is performed by comparing the relation (10) with the critical value Γ_c obtained from χ^2 distribution tables:

$$\Gamma_c = \chi^2(v, 1-\alpha), \quad (11)$$

where v = P – 1 degrees of freedom, and α is the significance level of the test. If $\Gamma_{\theta_k} < \Gamma_c$, the hidden neuron or the input under consideration must be pruned.

The value of σ_0^2 is crucial to the success of this algorithm. If σ_0^2 is too small, no parameters will be pruned. On the other hand, if σ_0^2 is too large, too many inputs or hidden neurons will be pruned. The algorithm therefore starts with a small value of σ_0^2 (0.001) and multiplies this value by 10 if nothing is pruned, until σ_0^2 equals 0.1 [36]. In this paper, this algorithm is denoted “Engel”.

B. The proposed algorithm

The previous algorithm allows the simultaneous pruning of spurious hidden neurons and feature selection. However, the pruning is approximate because the input variable may be either pruned or conserved; in the latter case, the input under consideration is distributed to all hidden neurons without distinction.

An input variable may be useful for the evaluation of the output of one hidden neuron and spurious for the evaluation of another. In this case, the algorithm “Engel” may have two extreme behaviors:

- pruning of a partially used variable, which implies a loss of information;
- retention of spurious parameters, which can cause perturbations and overfitting.

Therefore, the proposed approach decides whether to keep or prune each parameter individually, and does not consider all parameters related to an input together.

In the “Engel” algorithm, two different categories of elements θ_k (inputs and hidden neurons) may be eliminated. Now, three categories must be considered:

- weights connecting the input to the hidden neurons w_{ih}^1;
- the bias of the hidden neurons b_i^1; and
- weights connecting the hidden neurons to the output neuron w_i^2.

The sensitivity of the output network to the bias b of the output neuron is constant and is equal to one, so this algorithm (and the “Engel” algorithm) may not prune this parameter if required.

For each type of parameter, the sensitivity of the network output to the parameters is needed. For the weight w_i^2 connecting the hidden neuron i to the output neuron, this sensitivity is given by (5).

For the bias b_i^1 of the hidden neurons, the sensitivity corresponds to the contribution of this parameter to the global error of the output of the network. This contribution is determined by the partial derivative of the output of the network z with respect to the bias b_i^1 being considered:

$$\begin{aligned} S_{b_i^1}(p) &= \frac{\partial z(p)}{\partial b_i^1} = \frac{\partial z(p)}{\partial x_i^1(p)} \cdot \frac{\partial x_i^1(p)}{\partial z_i^1(p)} \cdot \frac{\partial z_i^1(p)}{\partial b_i^1} \\ &= w_i^2 . g'\left(z_i^1(p)\right) 1 \\ &= w_i^2 . \left(1 - \left(x_i^1(p)\right)^2\right) \end{aligned} \quad p = 1 \cdots P \quad . \quad (12)$$

Similarly, the sensitivity for the weights w_{ih}^1 connecting the input neurons to the hidden neurons is given by:

$$\begin{aligned} S_{w_{ih}^1}(p) &= \frac{\partial z(p)}{\partial w_{ih}^1} = \frac{\partial z(p)}{\partial x_i^1(p)} \cdot \frac{\partial x_i^1(p)}{\partial z_i^1(p)} \cdot \frac{\partial z_i^1(p)}{\partial w_{ih}^1} \\ &= w_i^2 . g'\left(z_i^1(p)\right) x_h^0(p) \\ &= w_i^2 . \left(1 - \left(x_i^1(p)\right)^2\right) . x_h^0(p) \end{aligned} \quad p = 1 \cdots P \quad . \quad (13)$$

As for the previous algorithm, the sensitivity of the output to a parameter is $S_{\theta_k}(p)$ (p=1…P and k=1…K=$(n_0+2).n_1$), with θ_k corresponding to:

- w_{ih}^1 if the weight connecting the input h to the hidden neuron i is considered;
- b_i^1 if the bias of the hidden neuron i is considered; or
- w_i^2 if the weight connecting the hidden neuron i to the output neuron is considered.

The notation $S_{\theta_k}(p)$ is then given by equations (13), (12) or (5) according to the considered case.

The following part of the algorithm is identical to the previous part. The hypothesis test is described by (9), which leads to a comparison between the value (10) and the threshold (11). The determination of (10) requires the calculation of the VNM using (7) and (8). The choice of the significance level α and the parameter σ_0^2 are the same as for the “Engel” algorithm, in order to allow the comparison of these two algorithms. In the following sections, this algorithm will be denoted “Engel_mod”.

C. The comparison algorithms

The proposed algorithm will be compared with three classical ones, the "Engel" one which has been presented in section III.A and two others, the OBS [20] and “N2PFA” [35]

algorithms. These two algorithms are summarized here.

1) Optimal Brain Surgeon (OBS)

This algorithm minimizes the sensitivity of the error criterion subject to the constraint of nullity of a weight. This nullity constraint expresses the deletion of this weight. The criterion considered is generally a quadratic criterion:

$$V(\theta) = \frac{1}{P}\sum_{p=1}^{P}\left(y(p) - z(p,\theta)\right)^2 , \tag{14}$$

where θ is a vector grouping all the weights and biases of the network, z is the network output and y is the desired output.

The sensitivity $\delta V(\theta)$ of the criterion $V(\theta)$ is approximated by a Taylor expansion around θ of order two:

$$\delta V(\theta) \approx \delta\theta^T V'(\theta) + \frac{1}{2}\delta\theta^T.H.\delta\theta . \tag{15}$$

Because the gradient $V'(\theta)$ is null after convergence, the first term in (15) vanishes, leading to:

$$\delta V(\theta) \approx \frac{1}{2}\delta\theta^T.H.\delta\theta , \tag{16}$$

which involves only the Hessian H. Vector e_q can be defined as a canonical vector selecting the q^{th} component of θ ($e_q^T = [0\cdots010\cdots0]$). The deletion of the weight θ_q (i.e., $e_q^T(\delta\theta + \theta) = 0$) must lead to a minimal increase of the criterion. The following Lagrangian may thus be written:

$$\mathcal{L}(\delta\theta) = \frac{1}{2}\delta\theta^T.H.\delta\theta + \lambda\left(e_q^T(\delta\theta + \theta)\right). \tag{17}$$

Minimizing this leads to:

$$\delta\theta = -\frac{\theta_q}{H^{-1}_{qq}}H^{-1}e_q , \tag{18}$$

where H^{-1}_{qq} is the q^{th} diagonal term of H^{-1}. The weight to be deleted is that which minimizes (15). Equation (18) allows the q^{th} weight to be forced to zero, and can update the remaining weights without retraining. It can nevertheless be useful to retrain the network after each pruning of a weight in order to compensate for the approximation introduced by the Taylor expansion (15). The main difficulty with this algorithm is the choice of the optimal structure, because no stop criterion is included. Different criteria may be used to evaluate the different structures. The two adopted here are the mean sum square error (MSSE) criterion, using the validation data set, and the final prediction error (FPE) criterion, which is used on the learning data set and accounts for the number of parameters [57]. The algorithm used for the experiment was programmed by Norgaard [47] and allows a relearning phase between each deletion of a parameter. The results presented are those obtained with or without these additional learning phases, so four different names are used for this algorithm:

- "OBS_L_FPE": FPE criterion with additional relearning;
- "OBS_L_MSSE": MSSE criterion with additional relearning;
- "OBS_WL_FPE": FPE criterion without additional relearning; and
- "OBS_WL_MSSE": MSSE criterion without additional relearning.

2) Neural Network Pruning for Function Approximation ("N2PFA")

This algorithm uses the mean absolute deviation (MAD) to measure the performance of the neural network. Two MAD values are calculated: MAD_T, which is based on the learning data set, and MAD_V, which is based on the validation data set

$$\begin{aligned} MAD_T &= M_T = \frac{1}{P_T}.\sum_{p=1}^{P_T}\left|y_T(p) - z(p)\right| \\ MAD_V &= M_V = \frac{1}{P_V}.\sum_{p=1}^{P_V}\left|y_V(p) - z(p)\right| \end{aligned} , \tag{19}$$

where subscript T represents the learning data set, subscript V represents the validation data set, and P and y are the number of data points and given data set, respectively. The values MAD_T and MAD_V are used to stop the pruning.

The "N2PFA" algorithm starts with an oversized neural network, and its parameters are then learned. The initialization of the algorithm is performed by calculating M_T and M_V (19) and by initializing the memories $M_T^{best} = M_T$ and $M_V^{best} = M_V$, and a threshold $Er_{max} = \max\left\{M_T^{best}, M_V^{best}\right\}$.

The algorithm then proceeds as two steps:

Step 1: Deletion of hidden neurons

- Set $w_i^2 = 0$ and calculate the MAD values $M_T(i)$ $(i=1\ldots n_1)$.
- Find the minimum $MT(ind)=\min(M_T(i), i=1\ldots n_1)$.
- Set $w_{ind}^2 = 0$, and relearn the network.
- Update the MAD values M_T and M_V.
- If $M_T \le Er_{max}(1+\beta)$ and $M_V \le Er_{max}(1+\beta)$:
  - prune the hidden neuron ind;
  - $M_T^{best} = \min\left(M_T^{best}, M_T\right)$;
  - $M_V^{best} = \min\left(M_V^{best}, M_V\right)$;
  - $Er_{max} = \max\left\{M_T^{best}, M_V^{best}\right\}$;
  - repeat step 1 with the new structure.
- Otherwise, restore the old weights and go to the next

step.

Step 2: Deletion of inputs

- Set $w_{ih}^1 = 0\ (\forall i)$ and calculate the MAD values $M_T(h)$ $(h=1\ldots n_0)$.
- Find the minimum $MT(ind)=\min(M_T(h), h=1\ldots n_0)$.
- Set $w_{i,ind}^1 = 0\ (\forall i)$, and relearn the network.
- Update the MAD values M_T and M_V.
- If $M_T \le Er_{max}(1+\beta)$ and $M_V \le Er_{max}(1+\beta)$:
  - prune the input ind;
  - $M_T^{best} = \min\left(M_T^{best}, M_T\right)$;
  - $M_V^{best} = \min\left(M_V^{best}, M_V\right)$;
  - $Er_{max} = \max\left\{M_T^{best}, M_V^{best}\right\}$;
  - repeat step 2 with the new structure.
- Otherwise, end the algorithm.

The value β is used to avoid an early halt of the pruning algorithm [35]. It is tuned to 0.025. In the following sections, this algorithm will be named "N2PFA".

IV. The Simulation Example

To test and evaluate the proposed pruning algorithm, two simulation examples were constructed.

A. Modeling a static system

The nonlinear simulation system to be modeled is based on a simple one-hidden-layer perceptron structure with three inputs and one output. This system, supposedly unknown, is chosen to avoid problems related to the differences between the form of the 'true' model and that of the fitted model. The system is described by:

$$y(t) = 1 + \tanh\left(2.x_1(t) - x_2(t) + 3.x_3(t)\right) + \tanh\left(x_2(t) - x_1(t)\right) + e(t), \quad (20)$$

where e(t) is an additive Gaussian noise whose mean is 0 and variance is 0.2.

Two data sets of 500 points are created, the first for model learning and the second for test or validation. These two data sets include five input variables (x_1, x_2 and x_3 used here and two supplementary ones). The five inputs are sequences of steps of random length and amplitude. To give each input a different influence, the input ranges are [–1; 1], [0; 1.5], [–1; 1.5], [0; 0.5] and [–1; 0]. The learning algorithm is from Levenberg–Marquardt [47].

The initial learning is carried out with a neural network comprising five inputs, eight hidden neurons and one output (i.e., 57 parameters), for a maximum of 50,000 iterations. Fifty sets of initial parameters were constructed using a modified Nguyen–Widrow algorithm [58]. The four pruning algorithms use the same set of initial parameters.

All the results obtained from the first system were grouped and synthesized in Table 1. The first column presents the number of inputs retained by the algorithms and the second column gives the number of hidden neurons retained. The third column gives the number of parameters (weights and biases) in the resulting models. The last column presents the time spent to complete the algorithms. For each column, the minimum, maximum and mean values of the parameters under consideration are noted for the different algorithms using the 50 sets of initial weights. For each column, two percentages are indicated. The first indicates the percentage of initial weight sets that yield values lower than the mean value. The second indicates the percentage of initial weight sets that yield values higher than the mean value. The lines correspond to the different tested algorithms.

Recall the optimal structure of the neural model (the objective of the different pruning algorithms). This structure comprises three inputs and two hidden neurons and is comprised of eight parameters as shown in (20).

First, the number of inputs remaining in the models with the different algorithms is studied. No algorithm prunes the two spurious inputs. Only the algorithms "Engel", "N2PFA" and OBS (using the MSSE criterion with and without relearning, i.e., "OBS_L_MSSE" and "OBS_WL_MSSE") prune one of the two spurious inputs. The percentages and the mean values show that the "N2PFA" algorithm gives the best results, before the OBS algorithm with relearning (when no relearning is performed, the performances of the algorithm deteriorates) and the "Engel" algorithm, which deletes one spurious input in 42% of the cases.

Next, the numbers of hidden neurons remaining in the models are compared for the different algorithms. The results are more dispersed than for the previous study. However, only the algorithms "Engel_mod", "N2PFA", "OBS_L_MSSE" and "OBS_WL_MSSE" reach a satisfactory number of hidden neurons (two). The algorithms "Engel_mod", "N2PFA", "OBS_L_MSSE" give similar results for mean values (3.63, 3.58 and 3.78, respectively), the percentage of results obtained that are lower than the mean values (46%, 56% and 58%, respectively) and maximal values (6, 8 and 8, respectively). When no relearning is performed with the OBS algorithms, the results deteriorate greatly.

At this point, the "N2PFA" algorithm seems to give the best results. This stance must be moderated when considering the number of parameters comprising the models. Only the OBS algorithm (using the MSSE criterion with and without relearning, i.e., "OBS_L_MSSE" or "OBS_WL_MSSE") finds eight parameters of the optimal structure. However, for these considered structures, some spurious parameters have been retained, to the detriment of others that were incorrectly pruned. In particular, one spurious input remains. Unlike the OBS algorithm, the "Engel_mod" and "N2PFA" algorithms reach structures close to optimal without pruning useful connections and retaining at best 11 and 13 parameters, respectively. The analysis of the percentages and of the mean and maximum values shows that these two algorithms give similar results.

TABLE 1: RESULTS OBTAINED ON THE SYSTEM 1

		Nb_I		Nb_H		Nb_θ		duration	
		val	%	val	%	val	%	val	%
Engel	min	4	42%	5	52%	31	40%	3.10E-02	26%
	mean	4.6	< >	7.2	< >	47.9	< >	4.70E-02	< >
	max	5	58%	8	48%	57	60%	6.30E-02	74%
Engel_mod	min	5		2	46%	11	46%	0.11	50%
	mean	5	< >	3.68	< >	24.4	< >	0.35	< >
	max	5		6	54%	43	54%	0.61	50%
N2PFA	min	4	98%	2	56%	13	56%	1.07	52%
	mean	4.02	< >	3.58	< >	22.6	< >	1.6	< >
	max	5	2%	8	44%	49	44%	2.27	48%
OBS_L_FPE	min	5		5	2%	25	44%	13.4	56%
	mean	5	< >	7.9	< >	50.1	< >	20	< >
	max	5		8	98%	57	56%	26.1	44%
OBS_L_MSSE	min	4	80%	2	58%	8	76%	13.4	56%
	mean	4.2	< >	3.78	< >	14.4	< >	20	< >
	max	5	20%	8	42%	57	24%	26.1	44%
OBS_WL_FPE	min	5		8		47	32%	7.56	56%
	mean	5	< >	8	< >	55.4	< >	9.85	< >
	max	5		8		57	68%	11.9	44%
OBS_WL_MSSE	min	4	8%	2	18%	9	26%	7.56	56%
	mean	4.9	< >	7.5	< >	49.8	< >	9.85	< >
	max	5	92%	8	82%	57	74%	11.9	44%

TABLE 2: RESULTS OBTAINED ON THE SYSTEM 2

		Nb_I		Nb_H		NB_θ		duration	
		val	%	val	%	val	%	val	%
engel	min	8	14%	7	38%	71	42%	3.10E-02	66%
	mean	9.84	< >	9.44	< >	112.9	< >	5.00E-02	< >
	max	10	89%	10	62%	121	58%	9.40E-02	34%
engel_mod	min	10		2	70%	22	48%	0.22	62%
	mean	10	< >	3.14	< >	36.8	< >	0.7	< >
	max	10		6	30%	64	52%	1.03	38%
N2PFA	min	4	70%	2	52%	13	60%	3.49	48%
	mean	5.52	< >	3.92	< >	32.7	< >	5.74	< >
	max	10	30%	10	48%	97	40%	8.11	52%
OBS_L_MSSE	min	4	48%	2	46%	8	52%	142.1	52%
	mean	5.72	< >	4.7	< >	17	< >	160.4	< >
	max	9	52%	9	54%	32	46%	180	48%

Finally, consider the time spent to complete the algorithms. The algorithm OBS is slower than the other three, even if no relearning occurs. The "Engel_mod" algorithm is faster than its rival ("N2PFA") with an average ratio of four between the duration of the "Engel_mod" algorithm (0.35 s), and the duration of "N2PFA" algorithm (1.60 s). This difference is due, in particular, to the relearning used in the "N2PFA" algorithm and not used in the "Engel_mod" algorithm.

A. Modeling a dynamic system

The second system model is also based on a single hidden layer perceptron, but this time using delayed inputs. This system is described by:

$$y(t) = 1 + \tanh\left(x_1(t-2) - x_2(t) + 3.x_2(t-1)\right) + \tanh\left(x_1(t-2) - x_2(t-2)\right) + e(t) \quad (21)$$

where e(t) is an additive Gaussian noise whose mean is zero and variance is 0.2. The delayed inputs x_1 and x_2 are sequences of steps of random length and amplitude. The duration of the steps of input x_1 (respectively x_2) is randomly chosen between 5 and 10 (respectively 8 and 15). The amplitude of x_1 (respectively x_2) is randomly chosen between –1 and 1 (respectively 0 and 1.5).

Two data sets of 500 points were created, the first for model learning and the second for testing or validation. The input vector used for the learning is comprised of the two inputs x_1 and x_2 and their respective delays t, t–1, t–2, t–3 and t–4.

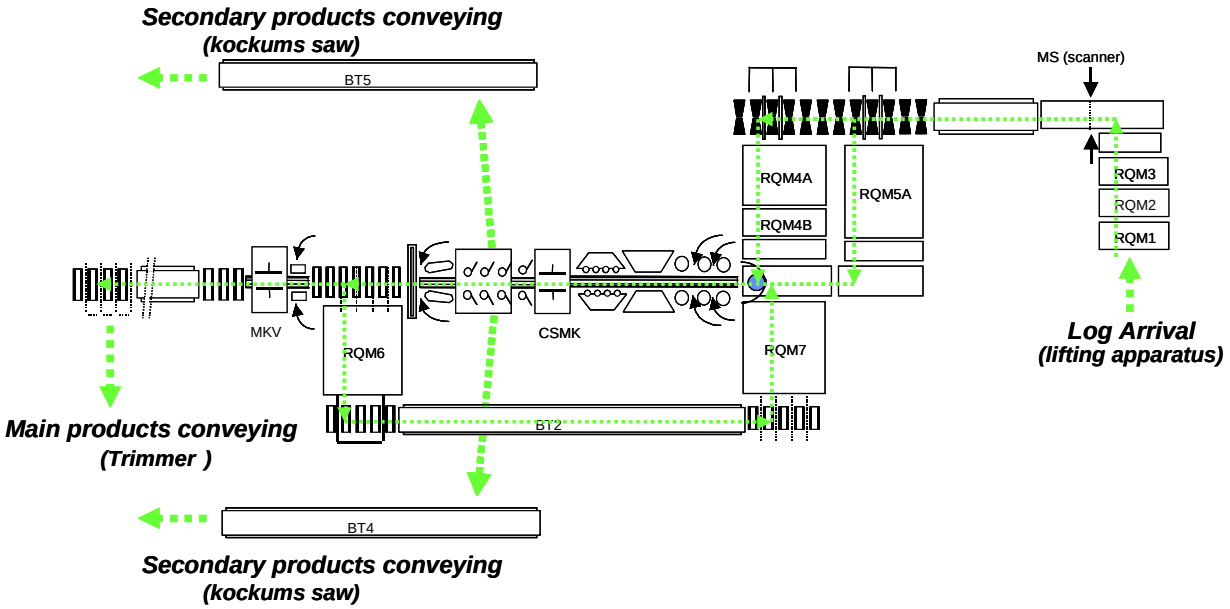

Figure 3. The canter line

This leads to 10 input neurons for the initial structure of the neural network.

The initial learning is carried out with a neural network comprising 10 inputs, eight hidden neurons and one output (i.e., 97 parameters), for a maximum of 50,000 iterations. Fifty sets of initial parameters were constructed using a modified Nguyen–Widrow algorithm. The four tested pruning algorithms used the same set of initial parameters.

All results obtained on the second system are grouped and synthesized in Table 2. The optimal structure of the neural model (the objective of the different pruning algorithms) comprises four inputs and two hidden neurons and is comprised of eight parameters, as shown in (21).

For this example, the OBS algorithm is tested using only the MSSE criterion and relearning phases. As in the previous example, the OBS algorithm prunes useful parameters. The algorithms “N2PFA” and “Engel_mod” perform best and give similar results. In particular, no other algorithm finds the optimal number of hidden neurons (two). The “N2PFA” algorithm reaches a satisfactory four; however, it is slower than the “Engel_mod” algorithm and takes eight times the execution time.

The results obtained from these two examples show that the OBS algorithm gives the worst results, so this algorithm will not be used for the industrial application.

I. Industrial Application

At the time of the study, the sawmill had a capacity of 270,000 m^3/year, a turnover of 52 million euros and 300 employees.

The internal supply chain can be described from a process point of view, and so the physical industrial production system can be broken down into three main parts. To understand the functioning of the process, the course of a log will be described, from its admission into the process to its exit in plank form.

The first part of the process corresponds to the canter line, which is presented in Figure 3. The product flow is represented by dashed arrows. The log is taken into the process by using conveyors RQM1, RQM2 and RQM3. Depending on its characteristics (scanner MS), the log is driven to RQM4 or RQM5, which are used as input inventory for the canter line. Next, the log goes on to the first canter’s machine, and later the CSMK saw transforms the log into a parallelepiped, the square in Figure 4.

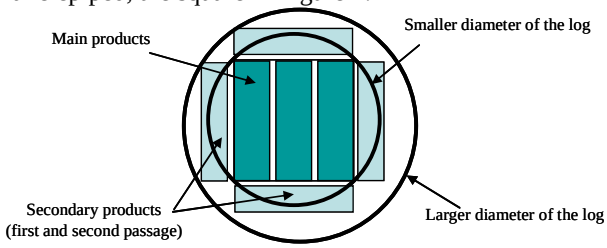

Figure 4. The cutting plan

This first step, which gives the two first sides of the parallelepiped, produces two planks (called secondary products) that are taken out of the canter line using the BT4 and BT5 conveyors. The log is then driven on the RQM6 conveyor, rotated 90° and stored in RQM7 to wait for its second passage to the CSMK saw. After the second passage, the squaring is complete and two other secondary products are taken out of the canter line (using the BT4 and BT5 conveyors) toward the second part of the process, the kockums line.

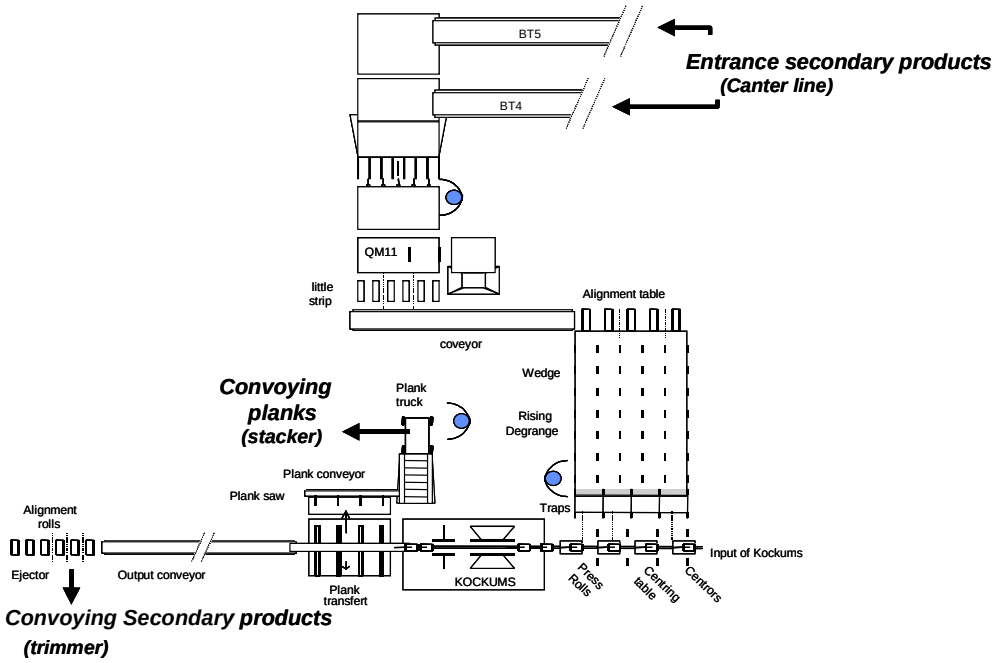

Figure 5. The Kockums line

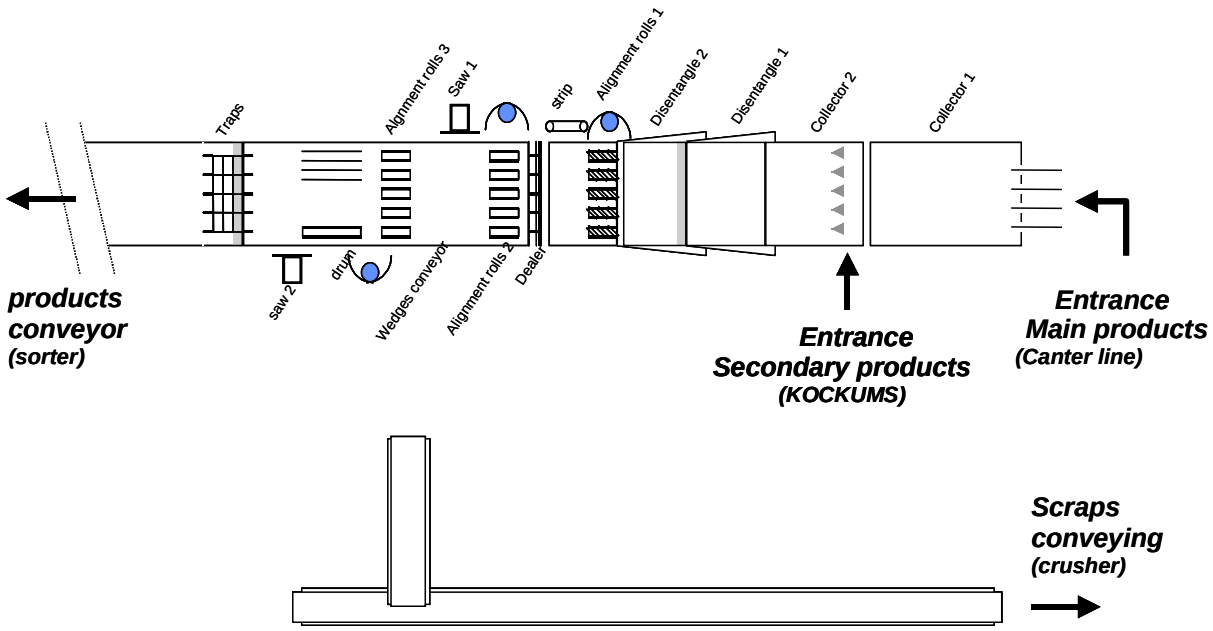

Figure 6. The trimmer line

The square is cut into three planks (called main products) on the MKV saw. These main products are driven to the third part of the process, the trimmer line. The cutting of the log into main and secondary products is described by the cutting plan (Figure 4).

Figure 5 shows the second part of the process, where the main machine is the kockums saw. Only secondary products are driven onto this part of the process. The secondary products are taken into the line using the BT4 and BT5 conveyors. They are then sawn up using the QM11 saw before reaching the kockums saw, which optimizes the plank depending on the needed products. The alignment table is used for the input inventory of the kockums saw. Finally, the secondary products are sent to the third part of the process using the exit conveyor.

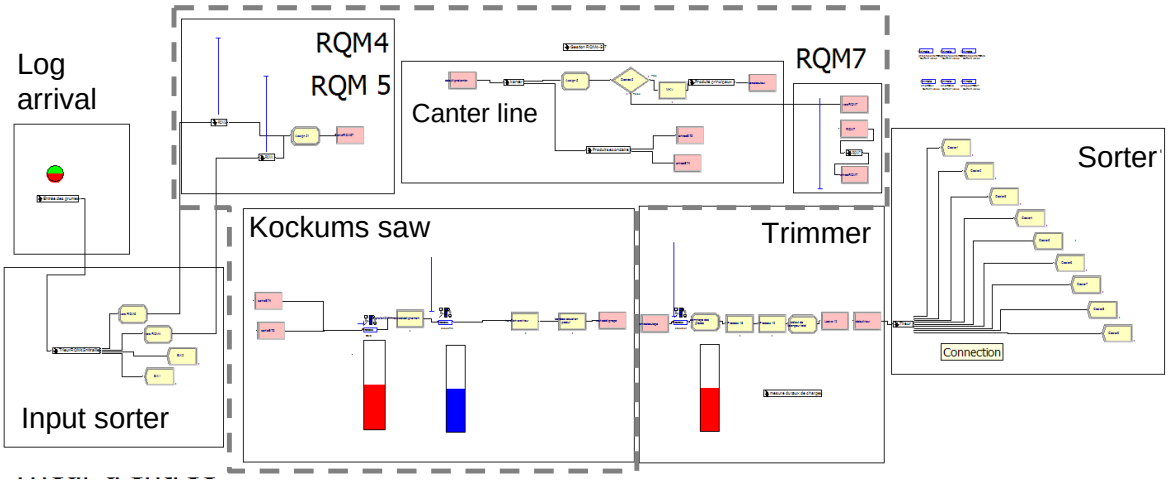

Figure 7. The complete model

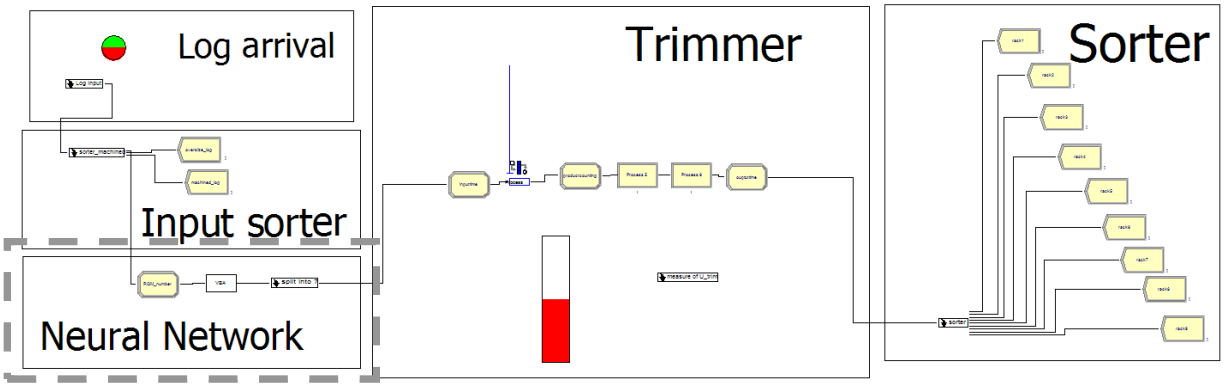

Figure 8. The reduced model

The third part of the process is the trimmer line, which is presented in Figure 6. This line performs the final operation: cross cutting, or cutting products to length. The inputs of the line are two collectors (1 and 2), which are used to collect secondary and main products from the kockums line and the canter line, respectively. Saw 1 is used to perform a default bleeding. Saw 2 cuts products to length.

Previous work [3] has shown that the trimmer saw is the bottleneck of the entire process.

I. The Reduced Model

A. *The complete model*

The complete model of the sawmill process was constructed in previous work [3]. This model is presented in Figure 7 and is composed of different modules. The first module is used to model the log arrival, which follows a homogeneous Poisson process. In this module, the characteristics of the log are measured using the scanner (Figure 3) and associated with the log.

A second module, the “input sorter”, directs the log to RQM4 or RQM5 depending on its characteristics. It may also eject the log from the process if it is machine-gunned or if its dimensions are out of range. The logs go to the next module, which models the RQM4 and RQM5 queues. Conveyors RQM4, RQM5 and RQM7 are used as input inventory for the canter line. Two other modules are used for the simulation of the canter line and the passage of the square in RQM7. The canter line model uses two submodels for the management of main and secondary products. The canter line has three outputs, which lead to the kockums line for the secondary products and to the trimmer line for the main products.

The other modules, which correspond to the core of the process, are simpler. They are used to model the kockums and trimmer lines and to model the sorting of products into different racks. The different submodels make the model presented in Figure 7 more complex than represented here. In particular, constructing the submodel to manage the priority rules for selecting the input inventory for the canter line is difficult.

B. The reduced model

The design of a complete model for the simulation of a workshop is a difficult task that leads to a complex model. The bottleneck of this line is the trimmer. According to the theory of constraints [2], the main industrial objective is to optimize the use of bottlenecks.

Within this framework, modeling the dependencies of inventories RQM4, RQM5 and RQM7, the canter line and the kockums line is unnecessary. In addition, all parts surrounded by the gray dashed line in Figure 7 give no direct or useful information for the evaluation of an MPS. Actually, only the arrival times of the products in the trimmer queue are useful for simulating the load of this bottleneck. This is why a multilayer perceptron is used to replace all parts surrounded by the gray dashed line in Figure 7. The neural network then uses the available shop floor information. This network will transform the information given by the "log arrivals" and "input sorter" modules into the arrival times of products at the entrance of the trimmer. It does not require the path used by the product or the transformations undergone by the product.

The reduced model is therefore obtained, where a large part of the model comprises a multilayer perceptron (Figure 8). The structure of this network must be determined.

C. Database and initial learning

For this, the available input data of the process are required [8]. First, each log is scanned at the input of the canter line. This information relates to the product dimension, with length (Lg) and three values for timber diameter (diaPB, diaGB and diaMOY). These variables are used to control the path of the log to the RQM4 or RQM5 queue. This choice is an additional information (RQM).

In addition to this dimensional information, the process variables must be characterized at the time of the log's arrival, so the input stock of the trimmer (Q_trim), the utilization rate of the trimmer (U_trim) and the number of logs present in the process between the inputs of RQM4 or RQM5 and the exit of the canter line (Q_RQM) must be measured.

The last type of information is related to the cutting plan of the logs. In fact, each log will be cut into *n* main or secondary products. In our application, the cutting plan (Figure 4) divides the log into seven products:

- two secondary products resulting from the first step of the cutting process on the CSMK saw of the canter line;
- two secondary products resulting from the second step of the cutting process on the CSMK saw of the canter line after staying in the RQM7 queue;
- three main products resulting from the third step of the cutting process on the MKV saw of the canter line.

These two saws (CSMK and MKV) belong to the canter line. These seven products can be classified into three categories according to the location (CSMK or MKV) and to the stage in the cutting process (first or second cutting). This information is given by the variable "type_piece". The last information is the thickness (in mm) of the product, which is also the reference. In this case, only two references are considered: main products are 75 mm and secondary products are 25 mm (ref). Consequently, the neural networks input variables are Lg, diaGB, diaMOY, diaPB, ref, type_piece, Q_trim, U_trim, Q_RQM and RQM. In this application, 12,775 products were simulated.

The objective is to estimate the delay (ΔT) corresponding to the throughput time for the 12,775 products, between the process input time and the trimmer queue input time. In practice, ΔT is the output of the neural network:

$$\Delta T = \sum_{i=1}^{n_1} w_i^2 . g\left(\sum_{h=1}^{10} w_{ih}^1 . x_h^0 + b_i^1 \right) + b \,. \tag{22}$$

The learning of the network is supervised, so it is necessary to divide the database into learning and validation data sets. Previous work [8][10] using the OBS algorithm has shown that 24 hidden neurons are sufficient for modeling the system, so the initial network structure uses 25 hidden neurons. The initial learning is therefore carried out with a neural network comprising 10 inputs, 25 hidden neurons and one output (i.e., 301 parameters), for a maximum of 50,000 iterations. To take into account that the learning algorithm performs a local search of the minimum, 50 sets of initial parameters have been constructed using a modified Nguyen–Widrow algorithm [58].

D. Comparison of pruning algorithms

The three pruning algorithms use the same set of initial parameters. The algorithms under consideration are "N2PFA", "Engel" and "Engel_mod". All the results obtained on the industrial case are grouped and synthesized in Table 3. The first line presents the number of inputs retained by the algorithms, and the second line gives the number of hidden neurons retained. The third line gives the number of parameters (weights and biases) of the resulting models. Lines 4 and 5 give values of the sum square error (MSSE), obtained for the three algorithms using the learning and validation data sets, respectively. The last line presents the time spent to complete the algorithms.

For each line, the minimum, maximum and mean values of the parameters under consideration are noted for the different algorithms using the 50 initial weight sets. For each line, two percentages are shown. The first indicates the percentage of initial weight sets that give values lower than the mean value. The second is the percentage of initial weight sets that give values higher than the mean value. The columns correspond to the different tested algorithms.

First, the number of inputs remaining in the model is studied. The three algorithms give very different results. If the "Engel_mod" algorithm retains all the inputs, then the "Engel" algorithm may prune all the inputs in some cases. The "N2PFA" algorithm has a more realistic behavior for the pruning of input variables.

TABLE 3: COMPARISON OF THE 3 ALGORITHMS ON THE INDUSTRIAL CASE

		Engel			Engel_mod			N2PFA		
		min	mean	max	min	mean	max	min	mean	max
Nb_I	val	0	8.14	10	10	10	10	5	8.62	10
	%	62%	< >	38%		< >		38%	< >	62%
Nb_H	val	0	2.26	5	2	2.8	5	2	18.82	25
	%	72%	< >	28%	48%	< >	52%	40%	< >	60%
Nb_θ	val	1	25.08	61	24	34.2	61	21	202	301
	%	72%	< >	28%	48%	< >	52%	42%	< >	58%
NSSE_ID	val	268250	381138	538740	264590	367401	509920	154610	248417	424620
	%	58%	< >	42%	58%	< >	42%	64%	< >	36%
NSSE_val	val	265350	407081	639370	285100	393779	575600	175810	266351	502580
	%	58%	< >	42%	58%	< >	42%	68%	< >	32%
duration	val	12.45	104.51	519.31	53.53	150.58	512.42	167.74	457.74	1570.6
	%	64%	< >	36%	76%	< >	24%	56%	< >	44%

Next, the number of hidden neurons remaining in the models is studied for the different algorithms. Considering the previous results obtained with the OBS algorithm [8], it was expected that most hidden neurons would have been retained. Yet, if the "Engel" algorithm performs the pruning of all hidden neurons in some cases, then in most cases the three algorithms converge toward the optimal two hidden neurons. However, this optimum number of hidden neurons is not found very often for the "N2PFA" algorithm, compared with the two other algorithms. The number of hidden neurons for the "Engel_mod" algorithm has a mean value of 2.8, a minimum value of 2 and a maximum value of 5 when the "N2PFA" algorithm finds the two hidden neurons in only 6% of cases and prunes no hidden neurons in 28% of cases.

The number of parameters remaining in the models is now considered. The "Engel_mod" algorithm finds the smallest structure (the results for "Engel" are biased by the absurd cases where all parameters are pruned). This is because of the number of hidden neurons that are retained and because the parameters are pruned one by one. Therefore, even if the "Engel_mod" algorithm cannot prune an input completely, it prunes many parameters connecting the inputs and the hidden neurons.

The values of NSSE obtained for the learning and the validation data sets are used to confirm or invalidate a choice of structure. These values are very difficult to compare between algorithms, because "N2PFA" relearns for 50 iterations after each pruning of an input or a hidden neuron, while the two other algorithms do not perform this relearning process.

For the three algorithms, the structures that retain only two hidden neurons give the best values of NSSE. This fact confirms the choice of a structure using two hidden neurons.

The last line of Table 3 gives the time spent to complete the three algorithms. The "Engel" and "Engel_mod" algorithms take very similar computing times, but the "N2PFA" algorithm requires three times the computing time, so it can take up to half an hour.

In summary, the "Engel" algorithm leads to an absurd structure without input or hidden neurons in 18% of cases. The "N2PFA" algorithm is the only one that can select the input variables. However, it retains fewer than six hidden neurons in only 14% of the cases. Moreover, this algorithm is three times slower than the other two. Finally, even if the "Engel_mod" algorithm cannot perform the variable selection, it does allow the rapid calculation of an acceptable number of hidden neurons for inclusion into the network.

A. Association of the "N2PFA" and "Engel_mod" algorithms

When considering the previous results, it is interesting to consider the association of the two algorithms, "N2PFA" and "Engel_mod", in order to determine the structure of the network.

The "Engel_mod" algorithm, which is the fastest, may be used on the initial structure to quickly determine a good number of hidden neurons. Then, the "N2PFA" algorithm may be applied on this smaller structure to determine the useful inputs. This approach should reduce the computing time.

Table 4 groups the results obtained using the 50 different initial sets of parameters.

TABLE 4: RESULTS OF THE ASSOCIATION

		Engel_mod + N2PFA		
		min	mean	max
Nb_I	val	5	7.98	10
	%	62%	< >	38%
Nb_H	val	1	2.16	3
	%	72%	< >	28%
Nb_θ	val	10	15.42	25
	%	72%	< >	28%
NSSE_ID	val	179070	232323	463450
	%	80%	< >	20%
NSSE_val	val	189300	244752	487760
	%	80%	< >	20%
mean error_ID	val	2.28E-11	5.94	272.69
	%	98%	< >	0.02
standard deviation error_ID	val	423.2	478.5974	680.82
	%	80%	< >	20%
mean error_val	val	0.02	14.22	270.79
	%	84%	< >	16%
standard deviation error_val	val	435.06	490.69	698.31
	%	80%	< >	20%
duration	val	502.58	629.18	1189.5
	%	68%	< >	32%

The first three lines present the number of inputs, the number of hidden neurons and the number of parameters comprising the resulting model, respectively. The next two lines show the NSSE values for the learning and the validation data sets, respectively. The next four lines present the mean and the standard deviation of the residuals obtained on the learning and validation data sets, respectively. The last line shows the computing time.

The computing time is well reduced, compared with the computing time required for the "N2PFA" algorithm used alone. However, all pruning phases take more than 10 minutes on average.

In most cases, the number of retained hidden neurons tends toward two. The mean number of inputs retained in the model is eight.

To determine the best structure, the preferred structure is that which gives the smallest mean values of the residuals for the learning and validation data sets, and the lowest values of NSSE. The selected structure includes eight inputs and two hidden neurons and therefore has 21 parameters. With this structure, the mean errors obtained on the learning (1.9×10^{-9}) and the validation (0.018) data sets are close to zero. Moreover, the standard deviation of the residual obtained with this structure is among the smallest (437.56 for learning and 456.17 for validation).

Most of the tests evolve to this considered structure. Figure 9 presents the selected structure.

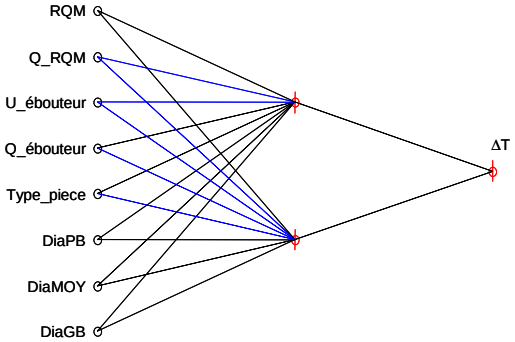

Figure 9. Structure of the network

Of the 10 initial inputs presented to the network, two have been pruned. These two inputs are the length of the log (Lg) and the type of product (ref). For this last variable, deletion could be predicted because the variable "type_piece" includes the information held by this variable. These two variables are strongly correlated, and the variable "type_piece" holds more information.

B. Results of the reduced model

The neural network is included in the reduced model. The performances of the reduced model and the complete model are investigated in this section.

Figure 10 shows the evolution of the input inventory of the trimmer as a function of the time (in seconds). This comparison is performed with two different data sets obtained under the same conditions. Figure 10 shows that the two models present the same type of queue evolution.

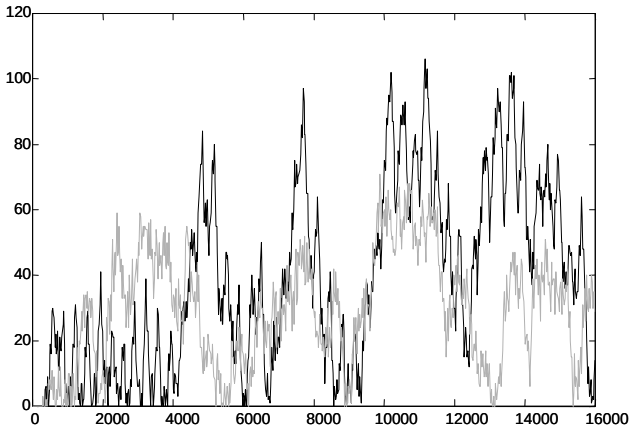

Figure 10. Input queue of the trimmer (black: complete model – grey: reduced model)

The differences between the two evolutions of the queues are due to the stochastic nature of the two models. The log arrival for the two models follows a homogeneous Poisson process with a mean of 20 seconds. The initialization of the models can produce an edge effect, which explains these differences in evolution of the two models [59][60].

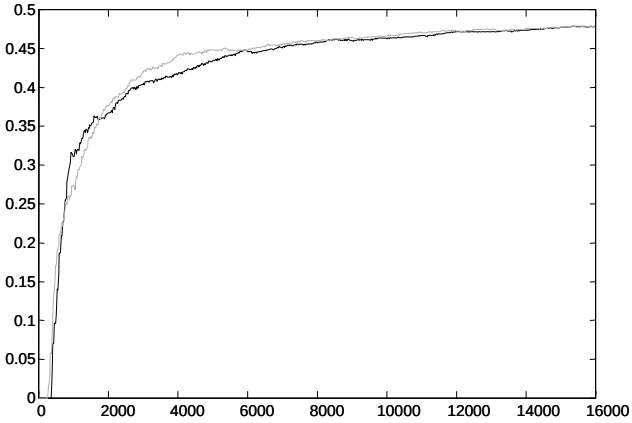

Figure 11. Utilisation rate of the trimmer (black: complete model – grey: reduced model)

Figure 11 shows the utilization rate of the trimmer as a function of the time (in seconds) for the two models (complete in black and reduced in gray). The two models present a similar evolution of the utilization rate and converge to the same value.

II. CONCLUSION

A new approach for simulation model reduction has been presented. This approach uses a neural network and, more specifically, a multilayer perceptron. The aim was to model the functioning of the part of a process that is not constrained in capacity.

This paper focused on one step of the network model

design: the determination of the structure of the network. In the first stage, a new pruning algorithm was proposed and compared with three others using two simulation examples.

Next, these pruning algorithms were applied to the reduction of a model of a sawmill's internal supply chain. The simulation examples and industrial case have highlighted the results of the different pruning algorithms.

The proposed algorithm allows rapid determination of a satisfactory number of hidden neurons, while the "N2PFA" algorithm is efficient for the determination of the useful inputs. Therefore, these two algorithms are associated and benefit from their different behaviors.

The results obtained for the industrial case have proved the good results of the pruning approach and the capacity to reduce the simulation model by using a neural network.

Future work could validate this approach using different application cases. One particular application could consider several external supply chains where at least one enterprise belongs to different supply chains.

REFERENCES

[1] P. Lopez, and F. Roubellat, *Ordonnancement de la production*, Hermès, Paris, 2001.

[2] E. Goldratt, and J. Cox, *The goal: A process of ongoing improvement*, North River Press; 2nd revised edition, Great Barrington, USA, 1992.

[3] A. Thomas, and P. Charpentier, "Reducing simulation models for scheduling manufacturing facilities", *European Journal of Operational Research*, vol. 161(1), pp. 111–125, 2005.

[4] E.H. Page, D.M. Nicol, O. Balci, R.M. Fujimoto, P.A. Fishwick, P. L'Ecuyer, and R. Smith, "An aggregate production planning framework for the evaluation of volume flexibility", *Proc. of the 1999 Winter Simulation Conference*, pp. 1509–1520, 1999.

[5] S.C. Ward, "Argument for constructively simple models", *Journal of the Operational Research Society*, vol. 40(2), pp. 141–153, 1989.

[6] R.J. Brooks, and A.M. Tobias, "Simplification in the simulation of manufacturing systems", *International Journal Production Research*, vol. 38(5), pp. 1009–1027, 2000.

[7] L. Chwif, R.J. Paul, and M.R. Pereira Barretto, "Discrete event simulation model reduction: A causal approach", *Simulation Modelling Practice and Theory*, vol. 14, pp. 930–944, 2006.

[8] P. Thomas, and A. Thomas, "Expérimentation de la reduction d'un modèle de simulation par réseau de neurones: cas d'une scierie", *7ème Conf. Int. de MOdélisation et de SIMulation MOSIM'08*, Paris, France, 30 March to 2 April, 2008.

[9] P. Thomas, G. Bloch, F. Sirou, and V. Eustache, "Neural modeling of an induction furnace using robust learning criteria", *Journal of Integrated Computer Aided Engineering*, vol. 6(1), pp. 5–23, 1999.

[10] P. Thomas, D. Choffel, and A. Thomas, "Simulation reduction models approach using neural network", *10th Int. Conf. on Computer Modelling and Simulation EUROSIM'08*, Cambridge, UK, 1 to 3 April, 2008.

[11] P.L. Bartlett, "For valid generalization, the size of the weights is more important than the size of the network", *Neural Information Processing Systems*, vol. 9, pp134-140, 1997.

[12] G.C. Cawley, and N.L.C. Talbot, "Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters", *Journal of Machine Learning Research*, vol. 8, pp. 841–861, 2007.

[13] H. Drucker, "Effect of pruning and early stopping on performance of a boosting ensemble", *Computational Statistics and Data Analysis*, vol. 38, pp. 393–406, 2002.

[14] C.M. Bishop, *Neural network for pattern recognition*, Oxford Univ. Press, Oxford, UK, 1995.

[15] P. Lauret, E. Fock, and T.A. Mara, "A node pruning algorithm based on a Fourier amplitude sensitivity test method", *IEEE Transaction on Neural Networks*, vol. 17(2), pp. 273–293, 2006.

[16] R. Chentouf, and C. Jutten, "Combining sigmoids and radial basis function in evolutive neural architecture", *European Symp. on Artificial Neural Network ESANN'96*, Bruges, Belgium, pp. 129–134, 1996.

[17] R. Setiono, "Feedforward neural network construction using cross-validation", *Neural Computation*, vol. 13(11), pp. 2865–2877, 2001.

[18] I. Rivals, and L. Personnaz, "Neural network construction and selection in nonlinear modeling", *IEEE Transaction on Neural Networks*, vol. 14(4), pp. 804–819, 2003.

[19] L. Ma, and K. Khorasani, "New training strategies for constructive neural networks with application to regression problems", *Neural Network*, vol. 17, pp. 589–609, 2004.

[20] B. Hassibi, and D.G. Stork, "Second order derivatives for network pruning: optimal brain surgeon", *Advances in Neural Information Processing Systems*, S.H. Hanson, J.D. Cowan and C.L. Gilles (Eds.), Morgan Kaufmann, San Mateo, CA, pp. 164–171, 1993.

[21] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, and C. Muller, "Neural modelling for time series: a statistical stepwise method for weight elimination", *IEEE Transaction on Neural Networks*, vol. 6(6), pp. 1355–1264, 1995.

[22] P. Thomas, and G. Bloch, "Robust pruning for multilayer perceptrons", *IMACS/IEEE Multiconference on Computational Engineering in Systems Applications CESA'98*, Nabeul-Hammamet, Tunisia, pp. 17–22, 1998.

[23] J. Xu, and D.W.C. Ho, "A new training and pruning algorithm based on node dependence and Jacobian rank deficiency", *Neurocomputing*, vol. 70, pp. 544–558, 2006.

[24] X. Zeng, and D.S. Yeung, "Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure", *Neurocomputing*, vol. 69, pp. 825–837, 2006.

[25] X. Liang, "Removal of hidden neurons in MLP by orthogonal projection and weight crosswise propagation", *Neural Computing and Applications*, vol. 16, pp. 57–68, 2007.

[26] E. Romero, and J.M. Sopena, "Performing feature selection with multilayer perceptron", *IEEE Transaction on Neural Network*, vol. 19(3), pp. 431–441, 2008.

[27] I.T. Jollife, *Principal component analysis*, Springer, New-York, 1986.

[28] P. Demartines, *Analyse de données par réseaux de neurones auto-organisés*, Ph.D. dissertation, Institut National Polytechnique de Grenoble, France, 1995.

[29] G. Dreyfus, J.M. Martinez, M. Samuelides, M.B. Gordon, F. Badran, S. Thiria, and L. Hérault, *Réseaux de neurones: méthodologies et applications*, Editions Eyrolles, Paris, 2002.

[30] H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar, "Ranking a random feature for variable and feature selection", *Journal of Machine Learning Research*, vol.3, pp. 1399–1414, 2003.

[31] T. Cibas, F. Fogelman Soulié, P. Gallinari and S. Raudys, "Variable selection with neural networks", *Neurocomputing*, vol. 12, pp. 223–248, 1996.

[32] P. Leray, and P. Gallinari, "Feature selection with neural networks", *Behaviometrika*, vol. 26(1), pp. 145–166, 1999.

[33] G. Castellano, and A.M. Fanelli, "Variable selection using neural network models", *Neurocomputing*, vol. 31, pp. 1–13, 2000.

[34] S. Gadat, and L. Younes, "A stochastic algorithm for feature selection in pattern recognition", *Journal of Machine Learning Research*, vol. 8, pp. 509–547, 2007.

[35] R. Setiono, and W.K. Leow, "Pruned neural networks for regression", *6th Pacific RIM Int. Conf. on Artificial Intelligence PRICAI'00*, Melbourne, Australia, pp. 500–509, 2000.

[36] A.P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information", *IEEE Transaction on Neural Networks*, vol. 12(6), pp. 1386–1399, 2001.

[37] B.P. Zeigler, *Theory of modeling and simulation*, Wiley, New York, 1976.

[38] G.S. Innis, and E. Rexstad, "Simulation model simplification techniques", *Simulation*, vol. 41, pp. 7–15, 1983.

[39] R.C. Leachman, *Preliminary design and development of a corporate level production planning system for the semi conductor industry*, Eds Optimization in industry, Chichester, UK, 1986.

[40] Y.F. Hung, and R.C. Leachman, "Reduced simulation models of wafer fabrication facilities", *International Journal Production Research*, vol. 37, pp. 2685–2701, 1999.

[41] J.S. Hwang, S. Hsieh, and H.C. Chou, "A Petri net based structure for AS/RS operation modeling", *International Journal Production Research*, vol. 36, pp. 3323–3346, 1999.
[42] G. Cybenko, "Approximation by superposition of a sigmoidal function", *Math. Control Systems Signals*, vol. 2(4), pp. 303–314, 1989.
[43] K. Funahashi, "On the approximate realisation of continuous mapping by neural networks", *Neural Networks*, vol. 2, pp. 183–192, 1989.
[44] R. Reed, "Pruning algorithm – A survey", *IEEE Transaction on Neural Network*, vol. 4(5), pp. 740–747, 1993.
[45] C. Jutten, and O. Fambon, "Pruning methods: a review", *Proc. of European Symp. on Artificial Neural Network ESANN'95*, Brussels, Belgium, pp. 129–140, 1995.
[46] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage", *Adv. Neural Inf. Process. Syst*., vol. 2, pp. 598–605, 1990.
[47] M. Norgaard, *System identification and control with neural networks*, Ph.D. dissertation, Institute of Automation, Technical University of Denmark, 1996.
[48] C.S. Leung, K.W. Wong, P.F. Sum, and L.W. Chan, "A pruning method for the recursive least squared algorithm", *Neural Networks*, vol. 14, pp. 147–174, 2001.
[49] H.S. Tang, S.T. Xue, R. Chen, and T. Sato, "H^{∞} Filtering method for neural network training and pruning", *J. Comp. in Civ. Engineering*, vol. 21(1), pp. 47–58, 2007.
[50] M.E. Ricotti, and E. Zio, "Neural network approach to sensitivity and uncertainty analysis", *Reliability Engineering and System Safety*, vol. 64, pp. 59–71, 1999.
[51] D. Sabo, and X.H. Yu, "A new pruning algorithm for neural network dimension analysis", *Proc. of the Int. Joint Conf. on Neural Network IJCNN'08*, Hong Kong, PRC, pp. 3313–3318, 2008
[52] H. Chandrasekaran, H.H. Chen, and M.T. Maury, "Pruning of basis functions in nonlinear approximators", *Neurocomputing*, vol. 34, pp. 29–53, 2000.
[53] A. Saltelli, K.S. Chan, and E.M. Scott, *Sensitivity analysis*, Wiley, New York, 2000.
[54] F. Gruau, "A learning and pruning algorithm for genetic boolean neural networks", *European Symp. on Artificial Neural Network ESANN'93*, pp. 57–63, 1993.
[55] D.W. Ruck, S.K. Rogers, and M. Kabrisky, "Feature selection using a multilayer perceptron", *Neural Network Computing*, vol. 2(2), pp. 40–48, 1990.
[56] G. Tarr, *Multilayered feedforward networks for image segmentation*, Ph.D. dissertation, Air Force Inst. Technol. Wright-Patterson AFB, 1991.
[57] L. Ljung, *System identification: theory for the users*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
[58] P. Thomas, and G. Bloch, "Initialization of one hidden layer feed-forward neural networks for non-linear system identification", *Proc. of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics WC'97*, Berlin, Germany, pp. 295–300, 1997.
[59] H. El Haouzi, *Approche méthodologique pour l'intégration des systèmes contrôlés par le produit dans un environnement de juste-à-temps: Application à l'entreprise TRANE*, Ph.D. dissertation, Université Henri Poincaré Nancy 1, France, 2008.
[60] T. Klein, *Le kanban actif pour assurer l'intéropérabilité décisionnelle centralisé/distribué: Application à un industriel de l'ameublement*, Ph.D. dissertation, Université Henri Poincaré Nancy 1, France, 2008.