ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

Petri nets design based on neural networks

Edouard LECLERCQ, Souleiman OULD EL MEDHI, Dimitri LEFEBVRE

GREAH - Université du Havre - 25, rue Philippe LEBON - BP 540 - 76058 LE HAVRE Cedex

Abstract. Petri net faulty models are useful for reliability analysis and fault diagnosis of discrete event systems. Such models are difficult to work out as long as they must be computed according to alarm propagation. This paper deals with Petri net models synthesis and identification based on neural network approaches, with regard to event propagation and to state propagation dataset. A learning neural algorithm is proposed to build Petri net models, these models are suitable for the diagnosis of discrete event systems.

1 Introduction

Fault diagnosis is an important issue for industrial processes, because faults often lead to unaccepted waste of productivity [1]. Diagnosis methods involve the analysis of collected measures, the estimation of critical parameters, and the comparison with reference models. For discrete event systems (DES) modelled by ordinary Petri nets (PN), diagnosis may be achieved with two distinct approaches. On the one hand, faults are modelled by forbidden states and we suppose that events are observed. It is about to estimate the forbidden markings from a partial observation of events [2]. On the other hand, faults are represented by events and we suppose that states are partially observed [3]. We focus on model synthesis and identification in sight of diagnosis [4], [5], [6], [7], [8]. In order to obtain the PN diagnoser, the first goal is to build a PN model of a process without a priori knowledge about the structure of the PN. This paper deals with the PN structure design from measured data.. The design is made by a neural network and the error back propagation learning algorithm. Other learning methods to build PN have been investigated such as genetic algorithm [9] or fuzzy logic [10].

Our work concerns the design of the PN from observations. These observations could be either events time sequences or markings time sequences. From the point of view of events observations, several methods are used for the investigation of PN properties: analysis and reduction methods like transitive matrices investigation [11]. This paper presents the way to derive and identify PN from events datasets. Such datasets are for example alarms sequences. From the point of view of states observations, the paper concerns the way to derive and identify PN from markings datasets. Such datasets are for example the value of critical parameters that are usually recorded by supervision systems. The contribution of the paper is also to present both approaches in a framework.

Our study is organized as follows. Section 2 concerns ordinary PN. In section 3, neural network used to model PN is presented, two approaches of the PN design are developed, the first one from events observations and the second one from states observations.

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

2 Ordinary Petri Net

An ordinary Petri Net (PN) is defined as $<P, T, W_{PR}, W_{PO}>$ where $P=\{P_i\}$ is a not empty finite set of n places and $T=\{T_j\}$ is a not empty finite set of p transitions. $W_{PR} = (w^{PR}_{ij}) \in \{0, 1\}^{n \times p}$ is the pre-incidence matrix and $W_{PO} = (w^{PO}_{ij}) \in \{0, 1\}^{n \times p}$ is the post-incidence one [12]. The PN incidence matrix W is defined as:

$$W = W_{PO} - W_{PR} \in \{-1, 0, 1\}^{n \times p}. \quad (1)$$

The PN marking M is an application from the set of places P to the set of non negative integer numbers $\mathbb{Z}^+$ such that, for each place $P_i \in P$, $M(P_i)$ is the number of tokens in place P_i. A firing sequence $\sigma = T_i T_j \ldots T_k$ is defined as an ordered series of transitions that are successively fired from marking M to marking M' such that equation (*2*) is satisfied:

$$M \xrightarrow{T_i} M_1 \xrightarrow{T_j} M_2 \rightarrow \ldots \xrightarrow{T_k} M' \quad (2)$$

Such a sequence is represented by its characteristic vector $F = (f_j) \in (\mathbb{Z}^+)^p$ where f_j stands for the number of T_j firings. The marking M' resulting from the marking M after firing the sequence F is given by:

$$\Delta M = M' - M = W.F \quad (3)$$

3 Learning algorithm for PN design

In this section, a learning algorithm, inspired from error back propagation, is investigated to obtain minimal size PN models.

The basic idea is to consider PN as a multi layered neural network: the hidden layer is composed of *nc* neurons that either correspond to *n* places or *p* transitions depending on the learning data set (sequences of events or sequences of markings) and the *ne* input and *ns* output layers both correspond either to the *p* transitions or to the *n* places (*ns* = *ne*). The weight matrix Q between input and hidden layers corresponds either to the connexion from transitions to places (i.e. matrix W_{PO}) or to the connexion from places to transitions (i.e. W_{PR}^T). The weight matrix between hidden and output layers V corresponds either to matrix W_{PR}^T or to matrix W_{PO}. This structure is represented in fig 1.

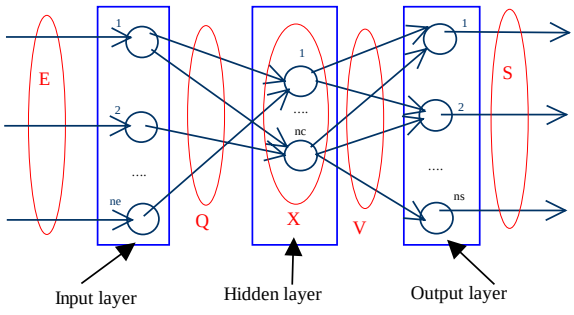

Fig. 1: Neural network structure, Q and V correspond to the incidence matrices.

The learning is computed with the dataset (*E, S*) obtained from a sequence of *nt* successive events or states.

For each input e_k, network output $y_k = (y_{ik})$ is compared to the desired output $s_k = (s_{ik})$ and the square error is computed:

$$\varepsilon = \sum_{i=1}^{ns}\sum_{k=1}^{nt}\left(y_{ik} - s_{ik}\right)^2 \quad (4)$$

The aim of the learning phase is to minimize error ε by updating weight matrices *V* and *Q* with (6). Output *Y* may be worked out with equation (5):

$$X = Q * E, \quad x_{ik} = \sum_{j=1}^{ne}\left(q_{ij} * e_{jk}\right) \text{ and } Y = V * X, \quad y_{ik} = \sum_{j=1}^{nc}\left(v_{ij} * x_{jk}\right) \quad (5)$$

$$\Delta v_{ij} = -\eta\frac{\partial\varepsilon}{\partial v_{ij}} = -\eta\sum_{k=1}^{nt}\left(y_{ik} - s_{ik}\right) * x_{kj} \text{ and } \Delta q_{lk} = -\eta\frac{\partial\varepsilon}{\partial q_{lq}} = -\eta\sum_{i=1}^{ns}\sum_{k=1}^{nt}\left(y_{ik} - s_{ik}\right) * v_{il} * e_{kq} \quad (6)$$

Updating equations (6) depend on the learning rate η. During updating, weight matrices take their values in a real set. Once error ε converges, binarisation of the weight matrices is computed thanks to function *Br*(.) in order to obtain the PN that has the closest behaviour from neural network (NN) behaviour:

$$\begin{aligned} &IR \rightarrow \{0, 1\} \\ &x \rightarrow Br(x) = 0 \text{ if } x < 0.5 \\ &\quad\; Br(x) = 1 \text{ if } x \geq 0.5 \end{aligned}$$

Once the binarisation computed, we test if the obtained PN verifies every propagation relationships, if it is not the case, another learning phase is started.

The drawback of the gradient algorithm is to reach local minima. Such minima can be detected when error increases or reaches a non zero stationary value. In that case, a partial initialisation of matrix *V* is proposed (matrix *Q* remains unchanged) .

As a conclusion, a two-loop algorithm is obtained. The first loop is a real one that corresponds to the back propagation algorithm (each execution is called "iteration", *limit_ite* is the maximal number of iterations per epoch). The second loop is a discrete one that gives the PN structure according to an equality test between desired matrix *S* and output matrix *Y* (each execution is called "epoch", *limite_epo* is the maximal number of epochs allowed, every 100 epochs a total random initialisation of matrices *Q* and *V* is done).

Two approaches are presented, the first one from the event set observations and the second one from the state set observations.

3.1 From event set

Let's assume that *Seq_E* is the event time sequence *Seq_E ={Seq_E(1) Seq_E(2)… Seq_E(nt)}* with *nt* observations where *Seq_E(i)* stands for the rank *i* event.

The idea is to compute the event directed paths (EDP), between all events of the considered DES according to the measured sequence of events, then to train the network in order to learn the EDP. Let define $E = \{e_1, e_2, \ldots, e_p\}$ as the set of events in *Seq_E*. An EDP exists from e_k to e_j if and only if the subsequence $[e_k e_j]$ exists in the sequence *Seq_E* (the causality relationships result from chronology). EDP are easily determined, as long as event are directly observed [3][13].

The input learning set is *E*. Each event e_i is considered as a *p* vector that satisfies $e_i(j) = 1$ if $i = j$ else $e_i(j) = 0$. The output learning set $S = \{s_1, s_2, \ldots, s_p\}$ is also a *p*

vectors set such as each vector s_i satisfies $s_i(j) = 1$ if an EDP exists from e_i to e_j else $s_i(j) = 0$.

The network is trained with input/output couples thanks to supervisory learning. Incidence matrices are obtained from $Q \in \mathbb{R}^{nxp}$ and $V \in \mathbb{R}^{pxn}$ with $W_{PO} = Br(Q)$ and $W_{PR} = Br(V^T)$.

In order to obtain the minimal number of places necessary to represent all EDP, the previous algorithm is associated with a pruning method that eliminates useless nodes in the hidden layer. The initial number of nodes (first stage of the pruning method) can be arbitrary chosen equal to the number of transitions p [14][15]. While all EDP are learnt the number of nodes is decreased: $p \leftarrow p-1$, and a new learning stage is computed. If during the learning stage, the number of epochs reaches the limit *limite_epo*, we consider that the number of nodes p becomes insufficient, and we retain $p+1$ nodes.

3.2 From state set

Let's assume that *Seq_S* is the state time sequence *Seq_S ={ Seq_S(1) Seq_S(2)…. Seq_S(nt)}* with *nt* observations where *Seq_S(i)* stands for the rank *i* state, represented by a *ne* vector.

The idea is to compute the state directed paths (SDP), between all states of the considered DES according to the measured sequence of states, then to train the network in order to learn the SDP.

3.2.1 SDP identification and definition of learning data sets

SDP have to be identified from the observed sequence. We have to distinguish several cases according to *Seq_S*, because there must be a bijection between input and output sets to make the network correctly learn all SDP.

To obtain this bijective relationship, a particular computation of the vectors *E(i)* and *S(i)* has to be made:

Each state in sequence *Seq_S* is considered as a *n* vector, *Seq_S*, *E* and *S* are considered as matrices : $Seq\_S \in (\mathbb{Z}^+)^{n \times nt}$, $E \in (\mathbb{Z}^+)^{n \times nt-1}$ and $S \in (\mathbb{Z}^+)^{n \times nt-1}$. Let *Vect(i)* refers to the i^{th} column of matrix *Vect*.

1 - *For every i within [1, nt-1]*

$E(i) \leftarrow (Seq\_S(i) - Seq\_S(i+1)) > 0$, the required input marking

$S(i) \leftarrow (Seq\_S(i) - Seq\_S(i+1)) < 0$, the obtained output marking

End For

2- If more than one output *S(j)* correspond to the same input vector *E(i)*, (if conflicts exist) then replace all the different outputs *S(j)* by the logical OR, between all of them. Only different vectors are conserved. An example of this computation is given below. From this algorithm we obtain the input and output learning sets *E* and *S*.

3.2.2 Learning and constraints

The network is trained with input/output couples thanks to supervisory learning. Incidence matrices are obtained from $Q \in \mathbb{R}^{p \times n}$ and $V \in \mathbb{R}^{n \times p}$ with *Br*(.) function: $W_{PR} = Br(Q^T)$ and $W_{PO} = Br(V)$.

To be sure to obtain a valid PN, (ie a PN with particular incidence matrices), constraints are applied to the matrices W_{PR} and W_{PO}. To limit the solution space for these matrices we have only considered PN without synchronisation and parallelism, and whose marking is bounded: each column of W_{PR} and W_{PO} has to have only one value 1, in the other case PN will absorb (synchronisation) or produce (parallelism) more than one token.

The number of hidden neurons corresponds to the number of transitions, this number *nc* is initialised by the number of different states in the observed sequence minus 1. A pruning algorithm (i.e. a test which consists in eliminating sink or source transitions and to eliminate all duplicated transitions) is also applied to obtain the nominal number of transitions.

4 Example

Let us consider the learning of a sequence of several states as an example. Examples with sequences of events can be founded in [14].

By running NN algorithm a PN is obtained according to the following parameters: η = 0.01, *error_ threshold* = 0.01, *limit_epo* = 1000 with a total random initialisation of matrices Q and V every 100 epochs and *limit_ite* = 200. The initial number of transitions is set to 9.

Considering the following state sequence *Seq_S* which presents more than one token and different conflicts, vectors *E* and *S* have to be computed, in fact only four relations have to be learnt to obtain the suitable PN.

$$Seq\_S = \left\{ \begin{bmatrix}2\\1\\0\\0\end{bmatrix} \begin{bmatrix}1\\2\\0\\0\end{bmatrix} \begin{bmatrix}0\\3\\0\\0\end{bmatrix} \begin{bmatrix}0\\2\\1\\0\end{bmatrix} \begin{bmatrix}1\\1\\1\\0\end{bmatrix} \begin{bmatrix}2\\0\\1\\0\end{bmatrix} \begin{bmatrix}2\\0\\0\\1\end{bmatrix} \begin{bmatrix}3\\0\\0\\0\end{bmatrix} \begin{bmatrix}2\\0\\0\\1\end{bmatrix} \begin{bmatrix}1\\0\\1\\1\end{bmatrix} \right\},\ E = \left\{ \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \right\}\ S = \left\{ \begin{bmatrix}0\\1\\1\\1\end{bmatrix} \begin{bmatrix}1\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \right\}$$

Algorithm has converged after 127 epochs and a total of 11147 iterations. Only 7 transitions are retained, the firing sequence is $\sigma = T_1T_1T_2T_3T_3T_4T_5T_6T_7$. The obtained incidence matrices and the PN are:

$$W_{PR} = \begin{bmatrix}1&0&0&0&0&1&1\\0&1&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\end{bmatrix} \text{ and}$$

$$W_{Po} = \begin{bmatrix}0&0&1&0&1&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&1\\0&0&0&1&0&1&0\end{bmatrix}$$

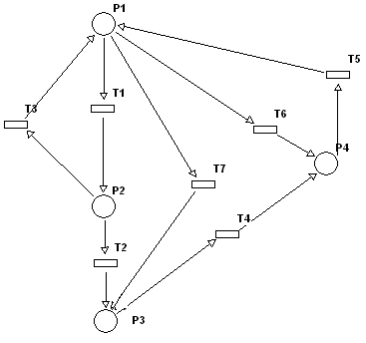

Fig 2 : PN model from state observations

5 Conclusion

An algorithm able to build PN models from observed sequences is presented. Two approaches of the PN design are developed, the first one from the observation of the events and the second one from the observation of the states. Such models are helpful for the diagnosis of DES.

Different kinds of PN may be built, all of them are ordinary PN with a bounded marking and a finite atteignability graph, but some of them can present conflicts (structural conflict) or have more than one mark. Synchronisation problems and real time applications will concern our further works.

References

[1] Y. Power, and P. A Bahri., A two-step supervisory fault diagnosis framework, WA 6150, 4-06 Juin, Murdoch, Australia, 2004.

[2] A. Giua and C. Seatzu, Observability of place / transition nets, IEEE – TAC, vol. 47, no. 9, pages. 1424–1437, 2002.

[3] Lefebvre D., Delherm C., Fault detection and isolation of discrete event systems with Petri net models, IEEE – TASE, Vol. 4, nu. 1, pp. 114–118, January 2007.

[4] T. Bourdeaud'huy and P. Yim, Synthèse de réseaux de Petri à partir d'exigences, Actes de la 5me conf. francophone de Modélisation et Simulation, pages 413-420, Nantes, France, September 2004.

[5] A. Giua and C. Seatzu, Identification of free-labeled Petri nets via integer programming. In Proc. 44th IEEE Conf. on Decision and Control, Seville, Spain, December 2005.

[6] K. Hiraishi, Construction of a class of safe Petri nets by presenting firing sequences. In Jensen, K., editor, Lecture Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets 1992, Sheffield, UK, volume 616, pages 244-262. Springer-Verlag, June 1992.

[7] M.E. Meda-Campana and E. Lopez-Mellado, Incremental synthesis of Petri net models for identification of discrete event systems, In Proc. 41th IEEE Conf. on Decision and Control, pages 805-810, Las Vegas, Nevada USA, December 2002.

[8] M.E. Meda-Campana and E. Lopez-Mellado, Required event sequences for identification of discrete event systems. In Proc. 42th IEEE Conf. on Decision and Control, pages 3778-3783, Maui, Hawaii, USA, December.

[9] J. Reid, Constructing petri net models using genetic search, Mathematical and Computer Modelling, Volume 27, Issue 8, pages 85-103, April 1998

[10] W. Pedrycz and H. Camargo, Fuzzy timed Petri nets, Fuzzy Sets and Systems, Volume 140, Issue 2, 1, pages 301-330, December 2003.

[11] J. Liu, Y. Itoh, I. Miyazawa and T. Sekiguchi, A Research on Petri Net Properties using Transitive Matrix, Proc. IEEE-SMC, vol. 1, pages 888-893, Tokyo, Japan,1999.

[12] R. David and H. Alla, Petri nets and grafcet – tools for modelling discrete events systems, Prentice Hall, London, 1992.

[13] Lefebvre D., Sensoring and diagnosis of DES with Petri net models, IFAC Safeprocess 2006, invited session “Model based fault analysis during a system’s entire life cycle”, pages 1213-1218, Beijing, China, September 2006.

[14] S. Ould El Mehdi, E. Leclercq and D. Lefebvre, Apprentissage hors ligne et en ligne de modèles par réseaux de Petri pour le diagnostic des SED, Qualita 2007 – Tanger (Maroc), 20-22 mars 2007.

[15] S. Ould El Mehdi, E. Leclercq and D. Lefebvre, Petri nets design and identification for the diagnosis of discrete event systems, IAR-ACD Annual Meeting Nancy, workshop on Advanced Control and Diagnosis, 2006.