Adaptive fuzzy petri nets for dynamic knowledge representation and inference

X. Li, F. Lara-Rosano*

Centre for Instrumentation Research, National University of Mexico (UNAM), Apartado Postal 70-418, Mexico City, D.F. 04510, Mexico

Abstract

Knowledge in some fields like Medicine, Science and Engineering is very dynamic because of the continuous contributions of research and development. Therefore, it would be very useful to design knowledge-based systems capable to be adjusted like human cognition and thinking, according to knowledge dynamics. Aiming at this objective, a more generalized fuzzy Petri net model for expert systems is proposed, which is called AFPN (Adaptive Fuzzy Petri Nets). This model has both the features of a fuzzy Petri net and the learning ability of a neural network. Being trained, an AFPN model can be used for dynamic knowledge representation and inference. After the introduction of the AFPN model, the reasoning algorithm and the weight learning algorithm are developed. An example is included as an illustration.

Keywords: Petri nets; Knowledge-based systems; Fuzzy reasoning; Knowledge representation; Adaptive expert systems; Neural learning

1. Introduction

Petri Nets (PN) models and net theory have become an important computational paradigm to represent and analyze a broad class of systems. As a computational paradigm for intelligent systems, net theory provides a graphical language to visualize, communicate and interpret engineering problems, as well as a specification and engineering language which can be used as a development, simulation and implementation tool (Pedrycz & Gomide, 1994). PN have the ability to represent and analyze in an easy way concurrently and synchronization phenomena, like concurrent evolutions, where various processes that evolve simultaneously are partially independent. Furthermore, PN approach can be easily combined with other techniques and theories such as object-oriented programming, fuzzy sets, neural networks, etc. These combined PN are widely used in computer systems, manufacturing systems, robotic systems, knowledge-based systems, process control, as well as other kinds of engineering applications.

Because normal PN cannot deal with vague or fuzzy information such as “very high” and “good”, several Fuzzy Petri Nets (FPN) have been introduced. As a model of knowledge-based systems, FPN are used for fuzzy knowledge representation and reasoning. In fact, by implementing the FPN model, major features offered by the PN model, such as correctness, circular rules, consistency, and completeness checking, can also be applied. PN have an inherent quality in representing logic in an intuitive and visual way and also can be implemented to simulate systems in operation. Therefore, a complex fuzzy expert system reasoning path can be reduced to a simple sprouting tree when applying a FPN-based reasoning algorithm as an inference engine. Besides these applications, FPN theory also provides means to manipulate imprecise and vague information. So, many FPN models are proposed to support fuzzy reasoning and decision making (Bugarn & Barro, 1994; Cao & Sanderson, 1995; Chen, Ke & Chang, 1990; Looney, 1994; Scarpelli & Gomide, 1993; Scarpelli, Gomide & Yager, 1996; Yeung & Tsang, 1994, 1998). There are some other applications of FPN in knowledge-based systems, such as inconsistency checking (Scarpelli & Gomide, 1994), uncertainty management (Konar & Mandal, 1996), and knowledge learning (Looney, 1994). For more details, a brief summary and discussion of FPN and its application are given in Section 2.

In this paper, we pay attention to knowledge representation (by weighted fuzzy production rules) and inference with FPN. The proposed model is called Adaptive Fuzzy Petri Nets (AFPN). This model can not only be implemented to do knowledge inference, but also has a learning ability like a neural network, implying that the system knowledge can be learned.

* Corresponding author. Tel.: +52-5622-8601; fax: +52-5622-8620.
E-mail address: lararf@servidor.unam.mx (F. Lara-Rosano).
2. Fuzzy petri nets

Generally, a FPN structure is defined as a 8-tuple (Chen et al., 1990)

\[ FPN = \{ P, T, D, I, O, f, \alpha, \beta \} \]

where \( P = \{ p_1, p_2, ..., p_n \} \) denotes a set of places, \( T = \{ t_1, t_2, ..., t_m \} \) denotes a set of transitions, \( D = \{ d_1, d_2, ..., d_n \} \) denotes a set of propositions, \( P \cap T \cap D = \phi \) and \( |P| = |D| \). \( I(O) : T \rightarrow P^\ast \) is the input (output) function, a mapping from transitions to bags of places. \( f : T \rightarrow [0, 1] \) is an association function which assigns a certainty value to each transition. \( \alpha : P \rightarrow [0, 1] \) is an association function which assigns a real value between zero to one to each place, and \( \beta : P \rightarrow D \) is a bijective mapping between the proposition and place label for each node. (Chen et al., 1990).

Many results prove that FPN are suitable to represent misty logic implication relations. FPN can deal with different types of compound fuzzy production rules. In Jeffrey, Lobo and Murata (1996) and Chaudhury, Marinescu and Whinston (1993), FPN are also introduced to represent Horn clauses or Non-Horn clauses. For this reason, the majority of FPN models are used for fuzzy reasoning and decision making (Bugarn & Barro, 1994; Cao & Sanderson, 1995; Chen et al., 1990; Looney, 1994; Scarpelli & Gomide, 1993; Scarpelli et al., 1996; Yeung & Tsang, 1994, 1998). On the other hand, FPN can also be used for inconsistency checking and uncertainty management (Garg, Ahson & Gupta, 1991; Konar & Mandal, 1996; Scarpelli & Gomide, 1994). As a model for checking inconsistency, FPN has the advantage of supporting the representation and execution of fuzzy rules, and the verification process that allows the development of automatic verifiers.

By combining with some other techniques, FPNs are proved to be very useful in some actual applications. For example, in Hanna (1996)FPN with neural networks (NN) are used to model products’ quality from a CNC-milling machining centre. In Andreu, Pascal and Valette (1997), FPN-based programmable logic controllers (PLC) are developed, based on a combination of PN with Possibility theory. Also, taking G2, a real-time expert system development package, a continuous FPN (CFPN) has been implemented for intelligent process monitoring and control (Tang, Pang & Woo, 1995).

However, as Pedrycz and Gomide (1994) pointed, “the primary thrust of these attempts is in a proper representation of the semantics of the underlying reasoning mechanisms. They lack of an adjustment (learning) mechanism to cope with potential numerical deficiencies of these models”. Yeung also mentioned this problem (Yeung & Tsang, 1998). Henceforth, Hirota and Pedrycz (1994) changed the modeling perspective, and developed a NN with OR/AND logic neurons to model fuzzy set concatenations. Later, Pedrycz and Gomide (1994) proposed a generalized FPN model (GFPN) which can be transformed easily into the NN proposed by Hirota and Pedrycz. The introduced architecture supports an explicit rather than an implicit form of knowledge representation. Moreover, this approach is a constructive one as parameters of the corresponding NN can be learned (trained) rather than assigned via guess prior to the use of the net. Another FPN model with learning ability was proposed by Looney (1994). An algorithm for adjusting thresholds is presented, but adjusting the weights is done by testing, and without assurance of convergency of the algorithm. Lara-Rosano (1994b) introduced the Weighted Production Networks (WPN) to model complex knowledge where the antecedents in a production rule were differentially weighted to express “the relevance of the single factors in producing the result”. In this paper, Lara-Rosano also defines the weight of a Weighted Conjunction (WC) and the weight of a Weighted Disjunction (WD), but he does not consider thresholds. Lara-Rosano (1994a) models complex soft systems by syncretizing a fuzzy causal impact NN with a PN modeling approach. However, his model is used to model fixed parameters systems. A learning algorithm is not provided.

To overcome this problem, Yeung proposed a multilevel weighted fuzzy reasoning algorithm, where a FPN is enhanced to include a set of threshold values and weights. He assigns a threshold value to every proposition in the antecedent, so that it fires only when its certainty value is greater than the threshold. He also assigns a weight to each conditional proposition in a production rule of the type IF...THEN, for determining the consequent. Based on this FPN model, Yeung proposes an algorithm for fuzzy reasoning. The results and comparisons reveal that this method is much better than others (Yeung & Tsang, 1998).

Although Yeung has made his model more flexible, the weights (or the parameters) in the model are still fixed. For a fuzzy knowledge (or an expert) system, however, the experiences and learning are very important. This means that the model should be able to be modified according to new incoming data. Our AFPN model tackles this problem by making weights adjustable like those of a neural network. Therefore, our FPN model has both the dynamic abilities of an ordinary fuzzy PN and the learning ability of a NN.

3. Weighted fuzzy production rules and adaptive fuzzy petri nets

In this paper, we assume that a fuzzy knowledge-based system is described by weighted fuzzy production rules (WFPR) and the system modeling is realized by mapping these rules into an Adaptive FPN (AFPN).

3.1. Definition of weighted fuzzy production rules

The definition of a weighted fuzzy production rule (WFPR) that we give here is similar to those given in
Weights learning

\begin{itemize}
\item [(a)]
\begin{center}
\includegraphics[width=0.5\textwidth]{fig1a.png}
\end{center}
\item [(b)]
\begin{center}
\includegraphics[width=0.5\textwidth]{fig1b.png}
\end{center}
\item [(c)]
\begin{center}
\includegraphics[width=0.5\textwidth]{fig1c.png}
\end{center}
\end{itemize}

Fig. 1. Mapping from WFPRs’ to FPNAWs’.

(Lara-Rosano, 1994b) or (Yeung & Tsang, 1998) but differs from them because it also considers negative weights, that is, negative propositions are also permitted to appear in a rule. WFPRs are categorized into three types defined as follows: Let \( a_i \) (or \( \neg a_i \)) the \( i \)th antecedent proposition of the rule, and \( c \) the consequent proposition. Each proposition \( a_i \) can have the format \( "x \in f_i^a" \), where \( f_i \) is an element of a set of fuzzy sets \( S \). Let \( \lambda_i \) be the threshold of the proposition to fire the rule, \( w_i \) the weight of the antecedent \( a_i \) and \( \mu \) the certainty factor of the rule:

**Type 1.** A simple fuzzy production rule
\[
R: \text{IF } a \ (\text{or } \neg a) \ \text{THEN } c, \lambda, w \ (\text{CF} = \mu) 
\]
For this type of rule, since there is only one proposition \( a \) (or \( \neg a \)) in the antecedent, the weight \( w \) is 1 (or -1).

**Type 2.** A composed conjunctive rule
\[
R: \text{IF } a_1 \ (\text{or } \neg a_1) \ \text{AND} \ a_2 \ (\text{or } \neg a_2) \ \text{AND} \ldots \ \text{AND} \ a_n \ (\text{or } \neg a_n) \ \text{THEN } c, \lambda_1, \lambda_2, \ldots, \lambda_n, w_1, w_2, \ldots, w_n \ (\text{CF} = \mu), 
\]

**Type 3.** A composed disjunctive rule
\[
R: \text{IF } a_1 \ (\text{or } \neg a_1) \ \text{OR} \ a_2 \ (\text{or } \neg a_2) \ \text{OR} \ldots \ \text{OR} \ a_n \ (\text{or } \neg a_n) \ \text{THEN } c, \lambda_1, \lambda_2, \ldots, \lambda_n, w_1, w_2, \ldots, w_n \ (\text{CF} = \mu), 
\]

In these types, negative antecedent propositions are considered. When an antecedent proposition is a negative one, a negative real number is assigned to the corresponding weight of its FNPN model.

3.2. Definition of adaptive fuzzy petri nets

Chen’s FPN definition is only a basic structure, it cannot represent the above WFPRs completely. Considering these two aspects, we define an Adaptive Fuzzy Petri Net as follows:

**Definition 1.** An AFPN is a 9-tuple
\[
\text{AFPN} = \{P, T, D, I, O, \alpha, \beta, Th, W\} 
\]
where \( P, T, D, I, O, \alpha, \beta \) are defined the same as Chen et al., 1990; \( Th : P \rightarrow [0, 1] \) is the function which assigns a threshold value \( \lambda_i \) from zero to one to each place \( i \). \( Th = \{\lambda_1, \ldots, \lambda_i, \ldots, \lambda_m\} \). For any transition \( t \), if the certainty factors associated with the tokens of all its input places are all greater than their thresholds, then the transition is enabled and fires instantly.

\( W_f : I \rightarrow [-1, 1] \) and \( W_o : O \rightarrow [-1, 1] \), are sets of input weights and output weights which assign weights to all the arcs of a net \( W = W_f \cup W_o \). \( w_{ij} \in W_{ij} \) indicates how much a place (or an antecedent condition) impacts a following transition (or an event) connected by \( w_{ij} \). A positive value means a positive impact and a negative value means a negative impact. For a transition \( t \), assume \( I(t) = \{p_1, p_2, \ldots, p_n\} \). The corresponding input weights to these places are \( w_{i1}, w_{i2}, \ldots, w_{in} \). It is reasonable to let \( w_{i1} + w_{i2} + \ldots + w_{in} = 1 \). Since all these conditions result in one consequent, the sum of their impacts is one. In the same way, \( w_{oj} \in W_{oj} \) indicates how much a transition impacts its output places, if the transition fires.

Compared to Yeung and Tsang’s (1994) definition, our structure combines the advantages of the two definitions of Chen and Yeung, it has a simpler structure (Yeung’s is a 13-tuple) but almost the same description ability (except for the negative weights in our model). The comparison results in the following five differences:

1. For any description, just as Chen does, we only give the mapping without the target set. For example, \( \alpha, Th \).
2. In our definition, \( Th \) is defined as a mapping from each place to a threshold value rather than to a set of thresholds; it equals to \( Th \) and \( \gamma \) in Yeung’s definition.
3. The set of weights \( W \) in our model is composed of two parts: a set of input weights, and a set of output weights. An input weight is assigned to an arc from a place to a transition, and an output weight is assigned to an arc from a transition to a place, meaning the certainty value of the corresponding proposition (or rule). It equals to \( W \) and \( f \) in Yeung’s definition. On the other hand, weights are added on arcs, meanwhile in Yeung’s definition they are added on places.
4. In our model, the uncertain reasoning process integrates the impact of each weighted branch. However, this reasoning process is taken simply as the
the certainty value into each of its output places.

5. We consider negative weights, Yeung does not.

3.3. Mapping WFPR into AFPN

Arguments in favor of modeling FPRs by FPNs can be found in some papers, for example, Hanna (1996), Chen et al. (1990) and Yeung and Tsang (1994). A knowledge-based system can be modeled as an AFPN by mapping its WFPRs into small AFPNs and then connecting them through common places.

To map WFPR into AFPN, we map propositions as places; the connections “THEN” between antecedent and consequent propositions as transitions and arcs (for a composed disjunctive rule, more transitions are needed); the weights of the antecedent propositions and the certainty factor of the rule as the weights of the input and output arcs of the corresponding transition, respectively. The mapping of the three types of weighted fuzzy production rules into the FPNs is given by step by step as follows.

Type 1. A simple fuzzy production rule
R: IF a (or ￢ a) THEN c, λ,w (CF = μ)
Rule R is represented in terms of FPN by Fig. 1a

Type 2. A composed conjunctive rule
R: IF a₁ (or ￢ a₁) AND a₂ (or ￢ a₂) AND ... AND aₙ (or ￢ aₙ) THEN c, λ₁, λ₂, ..., λₙ, w₁, w₂, ..., wₙ (CF = μ),
Rule R is represented in terms of FPN by Fig. 1b

Type 3. A composed disjunctive rule
R: IF a₁ (or ￢ a₁) OR a₂ (or ￢ a₂) OR ... OR aₙ (or ￢ aₙ) THEN c, λ₁, λ₂, ..., λₙ, w₁, w₂, ..., wₙ (CF = μ),
Rule R is represented in terms of FPN by Fig. 1c.

4. Implementation of AFPN model

When implemented, an AFPN can be used as an inference engine. The implementation of an AFPN is realized by firing transitions. A transition t fires instantly as soon as it is enabled. t is enabled if all its input places have tokens whose certainty factors are greater than their thresholds.

Let I(t) = {p₁₁, p₁₂, ..., pₘ₁}, with the corresponding input weights w₁₁, w₁₂, ..., wₘ₁, and thresholds λ₁, λ₂, ..., λₘ, and let O(t) = {p₁₂, p₂₂, ..., pₘₙ} with the corresponding output weights w₁₂, w₂₂, ..., wₘₙ.

Definition 2. ∀t ∈ T, t is enabled and fired if ∀pᵢⱼ ∈ I(t), the certainty value α(pᵢⱼ) ≥ λⱼ, j = 1, 2, ..., n holds. After firing t, the token in pᵢⱼ is removed, and a token with certainty factor wᵢⱼ ∗ (minαⱼ + ∑(α(pᵢⱼ) − λⱼ)wᵢⱼ) is put into each of its output places pₖₗ, k = 1, 2, ..., m. If a place pₖₗ has more than one input transitions (as Fig. 1d) and more than one of its input transitions fire, then the new certainty factor of pₖₗ is the new token which is produced by the transition with the maximum output weight.

5. AFPN training

Before training an AFPN, we should be clear as to what are the system outputs and what its inputs. Assume that a system is described by WFPRs. We define all the right hands of the rules as system outputs. If we have enough
training data, the parameters can be adjusted well enough. According to different types of system inputs, an AFPN is divided into four types of sub-structures, which are shown in Fig. 1a–d. Therefore, the learning of a whole net can be decomposed into several simpler learning processes referring to these small subnets. This fact greatly reduces the complexity of the learning algorithm.

Given a WFPR \( R \), we suppose that the thresholds of all its antecedent propositions and its certainty factors are known. But we are not sure about the input weights \( W_I \). These weights are to be learned, but only in the situations where there are enabled transitions. On the other hand, for Type 1 FPRs, the input weights are 1, thus only Type 2 and 3 FPRs’ weights need to be learned. Furthermore, according to Definition 2, if we have a set of training data, by comparing the weights of the input arcs, it is not difficult to know which transition is the dominating one at this place; thus if the data are available, then training can be implemented.

Type 2 FPRs can be translated into a standard neural network:

\[
y(k) = W[(k)^TP(k) + b]
\]

where \( k \) is time, input vector \( P(k) = [\alpha(p_1)(k) - \lambda_1, \alpha(p_2)(k) - \lambda_2]^T \), the weight vector \( W(k) = [w_{11}(k), w_{12}(k)]^T \), bias \( b = \min(\lambda_1, \lambda_2) \), the output \( y(k) \) is the certainty factor of the conclusions as shown in Fig. 2. So, Widrow–Hoff learning law (Least Mean Square) can be applied as

\[
W(k + 1) = W(k) + 2\delta e(k)P(k), \quad e(k) = t(k) - y(k),
\]

where \( t(k) \) is the goal output (teacher) and the weight vector \( W(k) \) is calculated recursively. It is known that for a small enough positive constant \( \delta \), the updating law (2) converges to real values (Hagan, Demuth & Beale, 1996).

5.1. Example

\{A, B, C, D, E, F, G\} are related propositions of a knowledge-based system \( I \). Between them there exist the following rules:

R1: If A and B Then E, \( \lambda_A, \lambda_B, w_A, w_B \) (CF = \( \mu_1 \))
R2: If C Then F, \( \lambda_C \) (CF = \( \mu_2 \))
R3: If F Then G, \( \lambda_F \) (CF = \( \mu_3 \))
R4: If D and E Then G, \( \lambda_D, \lambda_E, w_D, w_E \) (CF = \( \mu_4 \)).

Based on above translation principle, we map this system \( I \) into an AFPN which is shown in Fig. 3.

For this knowledge-based system \( I \), we assume we know the following real data:

\( \lambda_A = 0.5, \lambda_B = 0.8, \lambda_C = 0.3, \lambda_D = 0.8, \lambda_E = 0.1, \lambda_F = 0.4, \mu_1 = 0.8, \mu_2 = 0.9, \mu_3 = 0.6, \mu_4 = 0.7 \)
As the desired weights
\[ w^*_A = 0.73, \quad w^*_B = 0.27, \quad w^*_D = 0.1, \quad w^*_E = 0.9 \]  
are unknown, we will use neural networks technique to learn these weights from actual data. Given any set of inputs \( Y = \{ \alpha(p_A), \alpha(p_B), \alpha(p_C), \alpha(p_D) \} \), the human expert can give the corresponding output \( \Psi = \{ \alpha(p_E), \alpha(p_F), \alpha(p_G) \} \). For example, suppose that for \( Y_1 = \{ 0.7, 0.9, 0.5, 0.9 \} \), the desired output is \( \Psi = \{ 0.5148, 0.45, 0.3406 \} \); and for \( Y_2 = \{ 0.4, 0.2, 0.9, 0.7 \} \), the desired output is \( \Psi = \{ 0.81, 0.648 \} \). The adaptable parts of AFPN (see Fig. 4) may be transformed as Fig. 3b and c.

Let the initial a priori weights be:
\[ w_A(0) = 0.5; \quad w_B(0) = 0.5; \quad w_D(0) = 0.2; \quad w_E(0) = 0.8 \]

To apply Widrow–Hoff learning law, we make \( t(k) \) the desired outputs \( t_{\alpha(p)}(k) = [0.5148, 0], \quad t_{\alpha(p)}(k) = [0.3406, 0.648], \quad y(k) = [\alpha(p_E), \alpha(p_G)], \quad P_1(k) = [\alpha(p_A), \alpha(p_B)], \quad P_2(k) = [\alpha(p_D), \alpha(p_F)], \quad \delta = 1.7 \). The learning results are shown in Fig. 4. One can see that after 20 steps, the weights converge to their real values, i.e. AFPN mathematically models the knowledge-based system \( \Gamma \).

6. Conclusion

The advantages and drawbacks of using FPN in fuzzy knowledge systems are listed in detail (Yeung & Tsang, 1994). Our work has the following features which are different from those of earlier works:

1. We consider the contribution of an antecedent proposition to its consequent proposition as a weight taken as an input weight in the AFPN model.
2. Negative impacts are also permitted in our model. This fact is very important to model social or economic systems.
3. The fuzzy token transfer rule of AFPN is based on the accumulation of the contributions of all its inputs (see Definition 2), while most others are based on Min–Max calculation.
4. AFPN can be trained like neural networks, by learning the weights from the data given by experience and/or experts. This is an innovation over other models.

References


